A Weak Supervision-based Approach to Improve Chatbots for Code Repositories

Farbod Farhour

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Computer Science) at

Concordia University

Montréal, Québec, Canada

May 2022

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Farbod Farhour**

Entitled: **A Weak Supervision-based Approach to Improve Chatbots for Code Repositories**

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

Master of Applied Science (Computer Science)

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. Juergen Rilling

_____ Examiner
Dr. Juergen Rilling

_____ Examiner
Dr. Weiyi Shang

_____ Supervisor
Dr. Emad Shihab

_____ Co-supervisor
Dr. Essam Mansour

Approved by _____
Lata Narayanan, Chair
Department of Computer Science and Software Engineering

_____ 2022

_____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

A Weak Supervision-based Approach to Improve Chatbots for Code Repositories

Farbod Farhour

Software chatbots are growing in popularity and have been increasingly used in software projects due to their benefits in saving time, cost, and effort. Through natural language, users communicate with chatbots to perform various tasks (e.g., monitor and control services). Natural Language Understanding (NLU) component is vital for chatbots as it enables them to understand the users' queries. NLUs need to be trained on various ways a user formulates a query (typically different paraphrases of the same intent). Nevertheless, when implementing a chatbot using an NLU, chatbot practitioners face a challenge in training the NLUs as labeled training data is scarce or unavailable. Typically, such training is done manually and prohibitively expensive.

In this thesis, we propose a weak supervision-based approach to automate the query annotation and chatbot retraining process. Specifically, we leverage weak supervision to label users' queries posted to a software repository-based chatbot. To evaluate the proposed approach, we perform a case study to assess our approach on the NLU's performance. We use a software repository-based chatbot dataset that contains 749 queries, with 52 intents in our evaluation. The results show that using our approach yields to an average increase of 17.16% in the NLU's performance in terms of F1-score. Also, we find that our approach labels, on average, 99% of users' queries correctly. Finally, our results show that applying more labeling functions improves the NLU's performance in classifying the user's query. Our work helps software engineering (SE) practitioners improve their chatbot's performance while requiring minimal training by automating the labeling process of users' queries.

Statement of Originality

I, Farbod Farhour, 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.

Dedication

To my beloved mother and father

and

my best friend, Mahjabin

Acknowledgments

First, I would like to thank my parents, sister, and beloved grandmothers for their continued inspiration, support, and motivation in all of my endeavors. Your aid is invaluable for helping me reach my goals. None of this would be possible without you.

I express my appreciation to my supervisors, Dr. Emad Shihab and Dr. Essam Mansour, who shared their experience and knowledge and patiently guided me through this research from beginning to end. I sincerely thank you for your time and efforts and for giving me this opportunity to finish this study together. Also, I thank Dr. Shihab for his support through my difficult time. I could not have done it without you. Your support means the world.

I would also thank my colleague, Dr. Ahmad Abdellatif, for working alongside me while conducting this research, submitting the related paper, and completing this study. I thank you for your dedication and hard work on this.

I must thank my closest friend, Mahjabin, who supported me throughout this journey from day one through to completion and was with me and inspired me every step of the way. Finally, I thank Eric, AmirHossein, MohammadReza, and Kiana, who continue to encourage and motivate me with your friendship, advice, and incredible dedication to your craft.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chatbots are becoming more popular in the software engineering (SE) domain. Numerous chatbots assist software practitioners in daily development tasks such as code conflict resolution (Paikari et al., 2019), answering software development questions using Stack Overflow (Romero, Parra, & Haiduc, 2020), managing tasks (Toxtli, Monroy-Hernández, & Cranshaw, 2018), and assisting newcomers in the onboarding to new projects (Dominic, Houser, Steinmacher, Ritter, & Rodeghero, 2020). The broad adoption of chatbots in the SE domain is mainly due to the cost and time savings, increasing task completion rate, and recent advances in artificial intelligence and natural language processing (NLP) (Abdellatif, Costa, Badran, Abdelkareem, & Shihab, 2020).

Natural language understanding (NLU) is the backbone component in every chatbot as it allows chatbots to understand user's input (Abdellatif, Badran, Costa, & Shihab, 2021). NLU uses machine learning (ML) and NLP to extract structured information (the user's intent from the query and related entities) from the input query. There are many off-the-shelf NLUs that developers can easily use and integrate into their chatbots implementation instead of developing one from scratch. For example, MSRBot (Abdellatif, Badran, & Shihab, 2020) uses Dialogflow NLU to answer software project-related questions. MSABot (Lin, Ma, & Huang, 2020) uses Rasa NLU to assist developers in managing microservices.

The main challenge of using any NLU is the need for training data. Specifically, chatbot developers need to train the NLU using various queries where a user can express her

intention. For example, the queries "What is the fixing commit for issue 5?" and "Show me the resolution commit for bug 5" have the same semantic (determining the fixing commits for a certain bug) but different structures. Training NLUs with many queries is of paramount importance since it allows the NLU to better classify the user's query (Abdellatif, Badran, Costa, & Shihab, 2021; Abdellatif, Badran, & Shihab, 2020). Proper NLU training improves the chatbot's ability to respond to queries correctly (Ask, Facemire, & Hogan, 2016; GoodRebels, 2021; Lebeuf, Storey, & Zagalsky, 2018). Previous studies point out that crafting and collecting training data is a challenging task (Abdellatif, Badran, & Shihab, 2020; Abdellatif, Costa, et al., 2020; Dominic et al., 2020). For example, Dominic et al. (2020) reported that the main limitation of training their chatbot is the small size of the training dataset. Moreover, collecting more training queries is a time-consuming task. There are several posts on Stack Overflow where chatbot practitioners ask about sources to train the NLU (jsphdnl, 2017; Sheri, 2020; Tran, 2020). One way to obtain more training data is to learn directly from the human-chatbot conversation. In other words, the chatbot developers need to monitor and annotate the user's input manually and re-train the NLU on this new annotated data. In fact, many chatbot practitioners and NLU owners recommend using this process, especially in the early releases of the chatbot as it is trained on a few training queries (Microsoft, 2021a; Rasa, 2021b). Applying this process in practice is a time-consuming task and introduces additional costs because it requires the dedication of a domain expert (e.g., annotator) to label users' queries manually.

Providing labeled training data has become the main bottleneck in deploying machine learning-related approaches (Oramas, Quadrana, & Gouyon, 2021a; Rao, Bansal, & Guan, 2021; Ratner et al., 2017; Shu et al., 2020). Creating such training sets is quite expensive, specifically when domain expertise is needed. For instance, some large companies hire teams to label training data for their products (C., 2016; Davis et al., 2013; Ng, 2017). Based on years of collaboration with companies, agencies, and research labs, Ratner et al. (2017) reported that many practitioners have been turning to weak supervision: cheaper sources of noisier (i.e., heuristic) but larger-scale training sets created via techniques such as using cheaper annotators, programmatic scripts, or more creative and high-level input from domain

experts. Therefore, Ratner et al. (2020), proposed Snorkel, a system that enables users to train their models without hand-labeling any training data. Instead, users write labeling functions that are arbitrary heuristics with unknown accuracies and correlations. The main purpose of Snorkel is that it denoises the labeling functions' outputs without access to ground truth.

In recent years, researchers have proposed approaches to label the user's input in the user-chatbot conversation (Hancock, Bordes, Mazare, & Weston, 2019a; Mallinar et al., 2019a; Oramas et al., 2021a; Yu, Ding, & Bach, 2021). For example, Hancock et al. (Hancock et al., 2019a) developed a self-feeding approach to train the chatbot on the posed query if the user's satisfaction with the chatbot response is above a certain threshold. Mallinar et al. (2019a) developed a framework that allows chatbot developers to search for training queries from the previous conversations, then assign labels to them using weak supervision. They evaluated the proposed approach on a customer service chatbot that has six intents. While these approaches help address the issue of manually labeling the data, none of these approaches targeted chatbots that operate in the SE domain. Moreover, some approaches are applied to limited datasets with few numbers of intents.

To address this gap, this thesis introduces a weak supervision-based approach, called AlphaBot, to automatically annotate user's queries for the SE chatbots. AlphaBot contains three main components, a **data preprocessing component**, meant to preprocess and remove the noise from the user's query; a **query information extractor component**, which extracts important information from the query that helps to identify the intent of the query; and an **intent labeler component**, that uses weak supervision (i.e., labeling functions) and information extracted from the query by the previous component to assign an intent for the query. We created and wrote the labeling functions in this component based on the domain and our expertise. It is noteworthy to mention that all the steps in the main components of AlphaBot are done automatically. To evaluate the proposed approach, we perform an empirical study by obtaining a dataset of a chatbot, called AskGit (Abdellatif, Badran, & Shihab, 2021), that answers questions related to software repositories (e.g., "How many commits in the dev branch?"). The dataset contains 749 queries that represent 52

intents. To the best of our knowledge, this dataset is the most comprehensive (regarding the size) and available dataset for software repository-based chatbots. More specifically, this thesis answers two research questions (RQ).

First, **"Does AlphaBot improve the NLU's performance?"** We find that using AlphaBot improves the NLU's performance up to an average of 17.16% in the F1-score. Moreover, the findings from our evaluation show that AlphaBot, which uses labeling functions (i.e., rules and heuristics), annotates more than 99% of queries correctly on average. This leads us to our next RQ where we examine the impact of applying labeling functions progressively on the NLU's performance.

Second, **"What is the impact of the number of labeling functions on performance?"** The results show that NLU's performance is proportional to the number of applied labeling functions (e.g., heuristics). In other words, adding more labeling functions tends to increase the NLU's performance in terms of F1-score. This is clearly shown when the NLU is trained with a small-sized dataset compared to when it is more robust, i.e., trained with many queries.

1.1 Thesis Overview

Chapter 2 provides the background and related works of this study. Also, the background section of Chapter 2 focuses on an overview of the chatbot's architecture and discusses a motivating example. With the background and related works of this study set in place, the core of this thesis is then spread out across the four following chapters:

- Chapter 3: This chapter details our approach and its components. In this chapter, we provide the details of our approach and its three main components.
- Chapter 4: We describe our case study design to evaluate AlphaBot in this chapter.
- Chapter 5: To evaluate our approach, we have to apply it to NLUs and assess its effectiveness. Therefore, we continue our study by asking **"Does AlphaBot improve the NLU's performance?"** and answering this question in this chapter.

- Chapter 6: While evaluating our approach by applying it to NLUs, we measure the impact of the number of labeling functions on performance. So, in this chapter, we ask **"What is the impact of the number of labeling functions on performance?"** and answer this question.

We discuss our findings in Chapter 7. Chapter 8 discusses the threat to validity. Chapter 9 concludes the thesis.

1.2 Thesis Contributions

This thesis makes the following contributions:

- We propose a weak supervision-based approach to automate the labeling process of the user's queries for chatbots in the SE domain.
- We conduct an empirical study to evaluate our approach where 1) we examine the NLU's performance when applying our approach. 2) We explore the impact of adding more labeling functions on the performance trend of the NLU.
- We make our approach implementation and datasets publicly available (Farhour, Abdellatif, Mansour, & Shihab, 2021) to enable replication and accelerate future research in the field.

Chapter 2

Background and Related Works

In this chapter, we describe the fundamentals and background of our study and dive into the related works that investigate weak supervision, and chatbots in the SE domain.

2.1 Background: Challenges of Building a Chatbot

Before diving deep into our study, we provide an overview of how a chatbot works. Figure 2.1 presents a simplified summary of a user-chatbot interaction for illustration purposes. Initially, the user asks the chatbot a question via natural language (e.g., "Who fixed issue 5?"). The chatbot forwards the user's query to the NLU to extract the intent (represents the user's intention of the question) and the entities (represents pieces of information in the query). In the query "Who fixed issue 5?", the intent is to determine the developer that fixed the bug (*DeveloperFixIssue* intent), and the entity is "bug ticket 5" of type IssueNumber. Next, the NLU sends all extracted information back to the chatbot. Finally, the chatbot uses this information to query the database/API to answer the user's query and returns the response to the user. We note that the chatbot's responses are fully dependent on the extracted information by the NLU. In other words, if the NLU misclassifies the intent of a user's query, then the chatbot executes a wrong command, which leads to an incorrect response. Thus, it could negatively impact the user experience with the chatbot (Lebeuf et al., 2018).

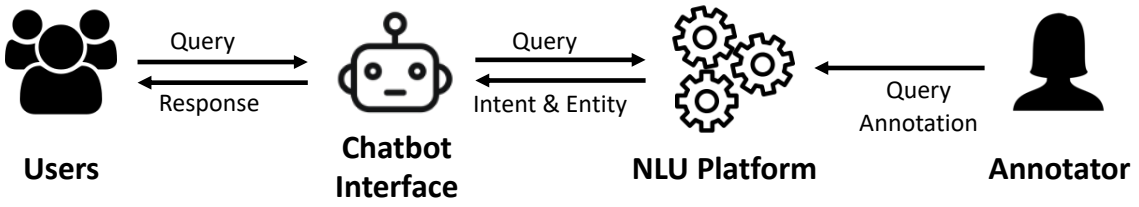


Figure 2.1: Motivating example of user-chatbot interactions overview.

As discussed in the previous chapter, chatbot practitioners suffer from a lack of data to train their chatbots. Moreover, prior work shows that NLUs perform better when they are trained on more data (Abdellatif, Badran, Costa, & Shihab, 2021). Therefore, to improve the NLU's accuracy, many NLUs record the users' queries and their extracted intents and entities (e.g., Google Dialogflow (Dialogflow, 2021) and Microsoft LUIS (Microsoft, 2021b)) to be examined by chatbot developers. In other words, the annotator (e.g., developer, domain expert) monitors the posed queries and the NLU extracted information (i.e., intent and entities) to ensure the correct extraction of the intent. In case of misclassified queries, the annotator manually annotates them with their correct intents. Then, those annotated queries are augmented to the queries in the original training dataset. Finally, the NLU is retrained on the queries in the augmented dataset. Many NLUs recommend this process to improve the NLU's performance in classifying the queries correctly (Microsoft, 2021a; Rasa, 2021b), especially at early releases of the chatbot by training the NLU on the data from chatbot users. Moreover, NLUs provide an interface to ease the manual annotation and retraining of the NLU (e.g., Dialogflow). However, this process is a burden and time-consuming because the chatbot developers need to dedicate time and effort for annotating users' queries posed to the chatbot. Moreover, it is a costly process as the labeling in some cases is done by domain experts, e.g., in chatbots that answer medical questions, the annotation is performed by medical practitioners (Ravì et al., 2017).

AlphaBot aims to help chatbot practitioners improve the NLU's performance by automating the annotation process of user's queries. Thus, it saves time, effort, and reduces the cost of chatbot development. Moreover, it allows practitioners to focus on the core and chatbot's critical tasks rather than annotating the user's queries.

This thesis proposes a weak supervision-based approach that labels the user's queries for chatbots in the SE domain. Thus, we divide the prior work into two areas; work related to weak supervision and work related to chatbots in the SE domain. We discuss these two areas in the following sections.

2.2 Weak Supervision and Dataset Annotation

Preparing labeled training data is one of the key development bottlenecks in supervised learning models (Oramas et al., 2021a; Rao et al., 2021; Ratner et al., 2017; Shu et al., 2020). This dependence of supervised learning models on labeled training data is nothing new. The manually-labeled training datasets are expensive and also time-consuming to create. It may take months for large benchmark sets. Moreover, in some problems, domain expertise is required for hand-labeling the training data. In practice, the cost, time, and the effort needed of hand-labeling such training datasets is the key bottleneck to deploying supervised learning models. By collaborating with companies, agencies, and research labs, Ratner et al. (2017) discovered that many practitioners have been turning to a weak supervision approach for their machine learning tasks: cheaper sources of noisier and heuristic but larger-scale training sets. These sets are created using methods, such as cheaper annotators, programmatic scripts, or more creative and high-level input from domain experts. Therefore, Ratner et al. (2020) proposed Snorkel, a system that enables users to train their models without hand-labeling any training data. Instead, users write labeling functions that are arbitrary heuristics with unknown accuracies and correlations. The main purpose of Snorkel is that it denoises the labeling functions' outputs without access to ground truth.

Labeling functions are heuristics that take as input a data point and either assign a label to it or abstain (not assigning any label). Labeling functions may be noisy. It means that they do not have perfect accuracy. Moreover, they do not have to label every data point. Due to their only requirement, which is mapping a data point with a label (or abstain), they cover a wide variety of forms (Snorkel-Team, 2020). Examples include (but not limited to):

- **Keyword search:** Searching for specific words in a text.

- **Pattern matching:** Searching for specific syntactical patterns in the text.
- **Third-party model:** Using a pre-trained model (mostly a model for a different task than the current one).
- **Distant supervision:** Utilizing an external knowledge base.
- **Crowdworker labels:** Treating each crowdworker as a black-box function that allocates labels to subsets of the dataset.

The goal of labeling function development is to make a high-quality set of training labels for the unlabeled dataset, not to label everything or directly build a model for inference using the labeling functions (Snorkel-Team, 2020). The training labels are used to train a separate discriminative model (in this study, the NLU model) in order to generalize to new, unseen queries (i.e., data points). Using an NLU, we can make predictions for queries that our labeling functions cannot cover.

Recently, many researchers use weak supervision for text classification (Meng, Shen, Zhang, & Han, 2018; Rao et al., 2021; Ruffolo. & Visalli., 2020; Shi et al., 2020; Shu et al., 2020). For example, Shu et al. (2020) used weak supervision to identify the intent of emails. They reported that their approach effectively leverage the weak supervision to enhance intent detection in emails. Rao et al. (2021) proposed a weak supervision model to classify the intent of user's search queries. They reported the efficacy of the weak supervision-based approach in improving the accuracy of the models. There is a number of studies that use weak supervision to improve the NLU's performance in chatbots (Hancock et al., 2019a; Mallinar et al., 2019a; Oramas, Quadrana, & Gouyon, 2021b). Hancock et al. (2019a) proposed a self-feeding chitchat chatbot that adds the users' queries to the chatbot's training dataset if the user is satisfied with its response. They declared that using weak supervision by learning from dialogue with a self-feeding chatbot enhances accuracy, nevertheless of the amount of traditional supervision. Mallinar et al. (2019a) presented a framework that enables annotators to search for user's queries in the customer service chatlog that are related to a certain intent. Then, the annotator verifies the labels for the retrieved queries. Finally,

the framework extends the labels to the rest of the queries in the set. They evaluated the proposed approach on a customer service chatbot that has six intents. Oramas et al. (2021b) developed a weak supervision-based approach to label the entities in the voice transcribed queries in the music domain. They concluded that using their approach and training with large amounts of weakly-supervised data created from unlabeled voice queries outperforms smaller amounts of hand-annotated data. While these studies help address the problem of manually labeling the data, none of these methods targeted chatbots that operate in the SE domain. Moreover, some approaches are applied to limited datasets with few numbers of intents.

Unlike these studies, to the best of our knowledge, AlphaBot is the first to use weak supervision to boost NLU's performance for SE chatbots. Software Engineering is a specialized domain that contains special terminologies used in a particular way. For example, in the SE domain, the word 'bug' refers to an issue in a bug tracking system (e.g., Jira) while in other domains it is related to an insect. In AlphaBot, we develop labeling functions that use SE terms to identify the intent of a query rather than general purposes labeling functions that are not tailored for the SE domain. We believe that our work complements prior work in applying weak supervision to a new domain using two popular NLU platforms (i.e., Rasa and Dialogflow). Moreover, we highlight some of the important features, such as entity (i.e., IssueNumber, IssueStatus, FileName, and DateTime) and Part of Speech (POS) that contribute to identifying the intent of a query in the SE domain.

2.3 Chatbot in Software Engineering

Several studies developed chatbots in different domains such as healthcare (Bharti et al., 2020), financial (Okuda & Shoda, 2018), and education (Clarizia, Colace, Lombardi, Pascale, & Santaniello, 2018). Recently, several studies proposed chatbots to assist software practitioners in their daily development tasks (Abdellatif, Badran, & Shihab, 2020; Bradley, Fritz, & Holmes, 2018; Dominic et al., 2020; Lin et al., 2020; Romero et al., 2020; Şerban, Golsteijn, Holdorp, & Serebrenik, 2021; Zhang et al., 2020). Abdellatif, Badran,

and Shihab (2020) developed MSRBot using Google Dialogflow to answer questions related to the software project (e.g., "Which commit fixes bug-5281?"). Lin et al. (2020) leveraged Rasa to implement MSABot that helps practitioners maintain microservices. Dominic et al. (2020) developed a chatbot using Rasa to assist newcomers in the onboarding process to new projects. Toxtli et al. (2018) developed TaskBot using Microsoft Language Understanding Intelligent Service (LUIS) to help software practitioners manage their tasks (e.g., task reminder).

The increased attention in SE chatbots and challenges to collect data to train chatbots (Abdellatif, Badran, & Shihab, 2020; Dominic et al., 2020) motivates our study; to help practitioners enhance the chatbot performance in intents classification and save resources by automating the annotation process of users input to the chatbot. That said, our study differs in that we aim to support chatbot practitioners and we do not develop chatbots.

Chapter 3

Study Approach

In this section, first, we provide an overview of our approach. Then, we discuss the three main components of this approach in detail.

Figure 3.1 presents an overview of our approach, which automates the labeling of users' queries to their corresponding intents. AlphaBot is divided into three main components, namely 1) Data preprocessing: eliminates the noise in the posed query, 2) Query information extractor: extracts helpful information (i.e., entities, part of speech, etc.) to be used for labeling the query, and 3) Intent labeler: uses the weak supervision-based approach to identify the intent of the query. The output of AlphaBot is the query with its corresponding intent (i.e., labeled query). Finally, we augment the labeled query to the chatbot's software repository-based training dataset and retrain the NLU on the augmented dataset to improve the NLU's performance. We elaborate on each of these parts in the following sections.

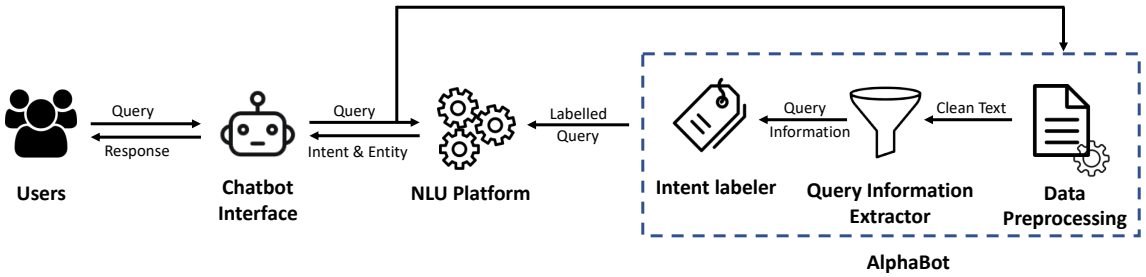


Figure 3.1: An overview of AlphaBot and its components.

3.1 Data Preprocessing

Data preprocessing is a critical step for NLP tasks (Vijayarani, Ilamathi, Nithya, et al., 2015). Since weak supervision applies labeling functions (i.e., heuristic) to the input data, transforming raw text into a more digestible form is critical to remove the noise from the user's query. Therefore, the main goal of this component is to provide a clean text for the next component (Query information extractor). To achieve that, we perform a series of steps as follows:

Text Preprocessing: As the user's text comes from user-chatbot interaction (i.e., chat forum), users might mistakenly add extra spaces or capitalize certain characters. Stop words and punctuation are other sources of noise that might affect the intent classification of the query. Thus, we remove extra white spaces, stop words, and punctuation from the query if they exist. This is a common step for NLP tasks (Abdellatif, Costa, et al., 2020; Bhadane, Dalal, & Doshi, 2015; Gomes et al., 2017; Jianqiang & Xiaolin, 2017).

Remove HTML tags: Since chatbots are integrated with third-party chat platforms, these platforms might append some tags that bias the intent's labeler component. For example, Slack adds a User ID tag (e.g., '<@U023BECGF>') to the user message. Thus, we clean all tags from the user's question. Although this step removes the noise introduced by the chatting platforms (e.g., Slack), the AlphaBot users need to be careful when applying this step because it depends on the context of the chatbot. For example, if the chatbot answers software development questions, then the questions might include HTML tags (e.g., <iframe>,).

Expand contractions: Users tend to use contractions on their chat messages that might mislead the Intent labeler component. This is because it could not extract the exact match of the word in the message. For example, if a user asks the chatbot: "Who doesn't have a lot of work to do in the project", then the labeling function would not detect the word 'not', which labels the query with *OverloadedDev* instead of *IdleDev* intent. Consequently, in this step, we expand all the contractions in the user's query.

After eliminating the noise from the input query, we forwarded the clean query to the

Query information extractor component. We detail this component in the following section.

3.2 Query Information Extractor

To classify the intent of a query, we need to extract useful information that refers to the correct intent of a user's query. In particular, the query information extractor component extracts the following information:

Entity: Represents an important piece of information in the query (e.g., FileName, DateTime, DeveloperName, and ProjectName). The entity that exists in a query helps identify the intent, especially if the intents share the same characteristics (e.g., have mutual words in their queries). For example, if a software repository-based chatbot supports the *GetNumberOfCommits* and *GetNumberOfCommitsOnSpecificDate* intents, and the user asks: "How many commits were pushed to the repository yesterday?", the chatbot should return the number of commits that happened yesterday (i.e., *GetNumberOfCommitsOnSpecificDate* intent) because there is a 'DateTime' entity (i.e., yesterday) in the query. Besides identifying the entities in the query, the entity type helps to filter out the irrelevant intents. For example, if a query has a 'CommitHash' entity type, then all the intents related to the repository are irrelevant (e.g., *NumberOfStars*, *NumberOfForks*, and *NumberOfDownloads* intents). This decreases the possibility of intents misclassification by reducing the number of possible intents that belong to the query.

Part-of-Speech (POS): POS for each word in the user's query is another valuable information source that helps identify the query's intent. For example, in the following two queries posted to a software repository chatbot, "List the commits that happened last week." and "Show me the developers that committed code last week.", the POS of the word 'commit' helps to identify the query's intent. In other words, if the POS of the word 'commit' is a noun, then the query is classified as *ListCommits* intent. Otherwise, if the word 'commit' is a verb, then the intent is *ListContributorsOnSpecificDate* intent.

Question Type: The type of WH question (e.g., who, what, or when) plays an important role in classifying the query's intent. For example, if the question type is 'Who', then the

user asks about a person (e.g., software developer). On the other hand, if the query is a 'How' question type, the user asks about a method to perform a specific task (e.g., "How to create CSS underline which partially covers the word?" (Knave, 2021)). In addition to the WH question, this step identifies if the query is a polar question (i.e., Yes/No question). For example, users want to verify something in their project (e.g., "Is the retailer API up?"). Therefore, we consider the question type in the user's query as another indicator of the query's intent.

After extracting the three pieces of information from the user's query, we forward them to the next component for labeling the query with its corresponding intent. Next, we describe the Intent labeler component.

3.3 Intent labeler

The main goal of the Intent labeler component is to label the user's query. The Intent labeler leverages a weak supervision-based approach to assign intent to the query. Weak supervision is a learning approach that combines knowledge from low-quality labeled data (e.g., keyword search, pattern matching, crowdsource labeling, etc.) to increase the accuracy of the training set (Ratner et al., 2017) and has been used by previous studies to improve the accuracy of chatbots operated on domains other than the SE domain (Hancock, Bordes, Mazare, & Weston, 2019b; Mallinar et al., 2019b). Among different types of weak supervision (Zhou, 2017), we formulate the task of labeling the user's queries with the corresponding intent as inaccurate weak supervision. In the inaccurate weak supervision, the given label (intent) to the query is not always the ground truth. This exactly matches our case where the NLU intent classification for a query is not always the ground truth. In fact, prior work shows that NLUs tend to misclassify intents with fewer training queries (Abdellatif, Badran, Costa, & Shihab, 2021).

Using a weak supervision approach, the Intent labeler component automates the labeling process for the user's queries posted to a software repository chatbot and keeps the abandoned queries manually to be labeled by the annotators. An abandoned query is a query

whose intent cannot be identified by the Intent labeler component. There are two reasons for a query to be classified as abandoned 1) the query does not match any of the defined heuristics, or 2) the chatbot does not support the intent. For example, if there is a chatbot that answers general development questions (e.g., "What is the difference between ArrayList and LinkedList?"), and the user asks a project-specific related question (e.g., "What is the number of opened bugs in the project?"), then, in this case, the question is labeled as abandoned because the chatbot does not support such kinds of questions. It is important to emphasize if the query is classified as abandoned, it will not be augmented to the training dataset because the Intent labeler component does not identify the intent of this query. The chatbot developers can examine the abandoned queries to expand the heuristics (i.e., labeling functions) or to support more intents that chatbot users demand. After the Intent labeler labels the query with its corresponding intent, we add the labeled query to the software repository-based training dataset and retrain the NLU on the augmented dataset.

3.4 Intent Labeling based on Weak Supervision

We reiterate that the Intent labeler component uses weak supervision to label the user's query. Ratner et al. (2020) proposed Snorkel, a system that enables practitioners to programmatically build and manage training datasets without manual labeling. In our study, we use Snorkel in the Intent labeler component. Snorkel is widely used by large IT companies (e.g., Microsoft, Google, IBM) and previous studies (Arachie & Huang, 2021; Bach et al., 2019; Fries et al., 2021; Parker & Yu, 2021). Practitioners through Snorkel can provide higher-level supervision (e.g., heuristics, distant supervision, etc.) through labeling functions (LFs) that take domain knowledge and resources. LFs are code snippets that label subsets of unlabeled data. For example, Figure 3.2 shows an LF that takes a user's query posted to a software repository chatbot as an input and returns the label (*ForksCount* intent) if the query satisfies all the conditions inside the LF. More specifically, it checks three conditions in the input query: 1) there are no entities in the query, 2) the existence of the 'fork' keyword, and 3) the presence of 'number' or 'count' keywords. In case one of the conditions is not fulfilled,

the LF returns 'ABSTAIN', which indicates the query is abandoned. In Snorkel, each intent should have at least one LF corresponding with that intent. Through LFs, practitioners can label large training datasets in hours or days rather than hand-label them over weeks or months.

```
# Condition 1: Has no entity.
# Condition 2: Has the word 'fork' in the clean text.
# Condition 3: Has the word 'number', or 'count' in the clean text.
# Intent: ForksCount
@staticmethod
@labeling_function(pre=[preprocess_command.__func__])
def forksCount_1(command):
    if LabelingFunctions.has_no_entity(command) \
            and \
            LabelingFunctions.check_in_clean_text(command, ['fork']) \
            and \
            LabelingFunctions.check_in_clean_text(command, ['number', 'count']):
        return LabelingFunctions._ForksCount
    else:
        return LabelingFunctions._ABSTAIN
```

Figure 3.2: An LF for the ForksCount intent.

Each query needs to pass through all implemented LFs even if it cannot be detected by the LFs. Then, each LF returns a label associated with the intent of that LF if the conditions are satisfied. Otherwise, it returns 'ABSTAIN'. Based on the returned labels, Snorkel determines the final label (intent) for the query. LFs may be noisy. It means that they do not have perfect accuracy. Moreover, they do not have to label every data point Snorkel-Team (2020). There are two methods that Snorkel uses to finalize the label of the query: 1) Majority vote: returns the most voted label among all LFs in case it is not 'ABSTAIN'. 2) Label model: Snorkel estimates the accuracies and correlation structure of the LFs using a generative model (Ratner et al., 2020) without accessing the ground truth (Bach, He, Ratner, & Ré, 2017). Thus, Snorkel weights the LFs by their true accuracies and eliminates the noise caused by some LFs outputs (Ratner et al., 2020). In case there are many LFs for the same intent and there are expected significant correlations/conflicts between the LFs, then the Label model is recommended (Ratner et al., 2020). Otherwise,

the Majority model is preferred when there are few LFs for each intent.

Chapter 4

Study Design

The main goal of AlphaBot is to improve the NLU's performance by auto-labeling the user's queries posted to the chatbot in the SE domain. To evaluate AlphaBot, we need to select a dataset for user-chatbot interactions in the SE domain and an NLU platform to measure the performance improvement after applying our approach. In this chapter, we detail our selection of the dataset, weak supervision framework, NLU platform, and the experiment design.

4.1 Dataset

To evaluate the performance of our approach, we opt for a dataset from the AskGit chatbot (Abdellatif, Badran, & Shihab, 2021). AskGit is a chatbot that answers questions related to software projects (e.g., "How many commits happened during March 2021?") on Slack. The AskGit dataset contains 749 queries distributed through 52 intents. To the best of our knowledge, this dataset is the most comprehensive (regarding the size) and available dataset for software repository-based chatbots. Table 4.1 presents a snapshot of ten intents with their descriptions. Also, we provide the distribution of queries across all 52 intents in Table A.3. We made the entire dataset publicly available (Farhour et al., 2021). Each intent in the dataset contains a set of training queries that present different ways a user could ask questions with the same semantic. For example, to know the developer who created the

bridge.py file in the project, a user could ask: "Who created this file bridge.py?", "Show me who created the file bridge.py", or "Who first added bridge.py file?". Those queries represent different ways of asking about the same semantic (i.e., *FileCreator* intent). On the other hand, the dataset contains four types of entities, namely 1) IssueNumber, 2) IssueStatus, 3) FileName, and 4) DateTime. Those entities cover the main artifacts in the repository. Table 4.2 shows the distributions for each entity type in the dataset.

Table 4.1: A snapshot of ten intents with their definitions from the AskGit dataset.

Intent	**Definition**	**No. of examples**
ProjectCollaborators	Presents the developer(s) who contributed to the project.	30
StarCount	Presents the number of stars for the repository.	29
IssueRelatedCommits	Determines the commits linked to a certain bug.	22
FileCreator	Identifies the developer(s) who creates a specific file in the repository.	21
IssueContributors	Determines the developer(s) who contributes in fixing a specific bug.	17
ModifiedFilesPR	Identifies the files that are touched by a specific pull-request.	12
ActivityReport	Presents statistics about the repository activities occurred in the last day (e.g., number of solved bugs).	10
LongestOpenPR	Identifies the longest pull-request which is still opened.	9
CommitsCountInBranch	Determines the number of commits in a certain branch.	7
OverloadedDev	Identifies overloaded developer(s) with the highest number of unfixed bugs.	6

Table 4.2: Entities distributions in the AskGit dataset.

Entity name	Definition	Total
IssueNumber	Issue ID number (e.g., issue 564)	192
FileName	Name of the file (e.g., map.json, response.java)	29
IssueStatus	The status of a GitHub issue (e.g., closed pull request, closed issue)	14
DateTime	specific period of time/date (e.g., last 24 hours, yesterday)	9

Several reasons motivate our selection of the AskGit dataset. First, this dataset reflects a realistic situation where software practitioners (e.g., project maintainers and managers) ask questions to get information about their software projects (e.g., "Who is assigned to the largest number of open issues?"). Second, it allows us to evaluate the applicability of the AlphaBot through applying it to a chatbot in production (AskGit). Moreover, the intents cover various types of queries related to a code repository (e.g., "Who last changed server.rb?"), issue tracker (e.g., "Please provide a report about closed issues in the repository"), a combination of both code and issue tracker (e.g., "What commits are linked to 3234?"), and software project (e.g., "When was this repository created?"). This helps to evaluate AlphaBot's performance for chatbots with many intents.

4.2 NLU Platform

As discussed in Chapter 2, every chatbot uses an NLU platform to understand the user's query, i.e., extract the intents and entities. There are many off-the-shelf NLUs available online that developers could use in their chatbot's implementation such as Google Dialogflow (Dialogflow, 2021). Among several NLUs, we select Rasa platform v2.2.3 (Rasa, 2021c) for several reasons. Rasa is an open-source NLU and can be run on a local machine. Therefore, Rasa implementation stays the same during our experiment. Moreover, it has been commonly used by prior work to develop chatbots (Dominic et al., 2020; Lin et al., 2020) which increases the usability of AlphaBot by the chatbot community. Our

approach requires the NLU to be retrained frequently (e.g., daily) as more users' queries are labeled. However, training NLUs is expensive in terms of time and resources, especially when the training dataset is large. Rasa overcomes the aforementioned issue through incremental learning (Rasa, 2021a). Incremental learning is a method to train the models on the new data and persist the knowledge gained from the original data (Polikar, Upda, Upda, & Honavar, 2001). Consequently, incremental learning saves time and computational resources in the case of training the NLU using large training sets (Geng & Smith-Miles, 2009). Using Rasa allows us (and we encourage other practitioners) to evaluate our approach on larger datasets with minimal computational resources and time. Finally, Rasa is free and supports multiple languages that enable our study's replicability by other researchers.

4.3 Experiment Settings

Before delving into AlphaBot performance evaluation, we describe the configurations in our approach used for the assessment. This allows the replication of the approach on other datasets. In the data preprocessing component, we leverage spaCy (spaCy, 2021) for text preprocessing and POS tagging in the query. SpaCy is a Python library for NLP tasks. We use Contractions Python library[1] to identify the closest expansion for the contraction in the input text.

For entity extraction in the query information extractor component, we use regular expressions to extract IssueNumber, IssueStatus, and FileName entities from the query. For example, to extract the issue number from the query, we check if the query has the word 'issue' or its synonyms (e.g., 'bug', 'ticket') followed by a number (e.g., 'issue 5938'). For the DateTime entity, we leverage Facebook Duckling (Facebook, 2021), which uses probabilistic context-free grammar through their pipeline to extract the DateTime entity. Prior work shows that Facebook Duckling performs well in extracting the DateTime entity from the user's query (Abdellatif, Badran, Costa, & Shihab, 2021).

[1] https://pypi.org/project/contractions

Backed by the extracted information (e.g., entity type) described in Section 3.2, the author of this thesis and one another domain expert used their chatbot development expertise to develop the LFs for all intents in the dataset (i.e., 52 intents). In particular, each domain expert independently examined three random queries for every intent in the dataset, and using the extracted information by the query information extractor component, they developed heuristics for each intent. Then, the authors discussed those heuristics and implemented them as LFs. As mentioned in Section 2.2, LFs may be noisy. It means that they do not have perfect accuracy. Moreover, they do not have to label every data point. Due to their only requirement, which is mapping a data point with a label (or abstain), they cover a wide variety of forms (Snorkel-Team, 2020), such as keyword search, pattern matching, third-party model, etc. In our approach, we mainly leveraged keyword search and pattern matching LFs. In addition to these LF types, we leverage spaCy (spaCy, 2021) for text preprocessing and POS tagging in the query. These NLP techniques are used to enhance the LFs and provide them more insights.

In total, we implemented 70 LFs using Snorkel. Some intents have more than one LF. For example, *OverloadedDev* intent has two LFs shown in Figure 4.1. The first LF in Figure 4.1a checks the query against three conditions (e.g., the query starts with 'Who') to be classified as *OverloadedDev* intent by this LF. Figure 4.1b demonstrates the second LF that classifies a query as *OverloadedDev* intent if a query satisfies both conditions. It is important to note that we configure Snorkel to use the Majority Vote model to identify the final label (intent) for the query since most of the intents (65%) have only one LF.

```
# Condition 1: has_and_has_only([issue_status]).
# Condition 2: is a wh question.
# Condition 3: text starts with who.
# Intent: OverloadedDev
@staticmethod
@labeling_function(pre=[preprocess_command.__func__])
def OverloadedDev_1(command):
    if LabelingFunctions.has_and_has_only(command, ['issue_status']) \
            and \
            LabelingFunctions.is_wh_question(command) \
            and \
            LabelingFunctions.text_starts_with('who', command):
        return LabelingFunctions._OverloadedDev
    else:
        return LabelingFunctions._ABSTAIN
```

(a) The first LF for OverloadedDev intent with three conditions.

```
    # Condition 1: has_and_has_only([issue_status]).
    # Condition 2: 'developer' in the clean text of command.
    # Intent: OverloadedDev
    @staticmethod
    @labeling_function(pre=[preprocess_command.__func__])
    def OverloadedDev_2(command):
        if LabelingFunctions.has_and_has_only(command, ['issue_status']) \
                and \
                LabelingFunctions.check_in_clean_text(command, ['developer']):
            return LabelingFunctions._OverloadedDev
        else:
            return LabelingFunctions._ABSTAIN
```

(b) The second LF for OverloadedDev intent with two conditions.

Figure 4.1: The two LFs for classifying queries as OverloadedDev intent.

4.4 Performance Evaluation

To evaluate Rasa's performance, we calculate the standard classification accuracy measures that have been used in prior works (e.g., (Abdellatif, Badran, Costa, & Shihab, 2021; Braun, Hernandez Mendez, Matthes, & Langen, 2017)) - precision, recall, and F1-score. In our study, recall is the percentage of the correctly classified queries to the total number of queries for that intent in the oracle, which is shown in Eq. 1.

$$Recall = \frac{TP}{TP + FN} \tag{1}$$

The precision is the percentage of the correctly classified queries to the total number of classified queries for the intent, as shown in Eq. 2.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Finally, we combine both precision and recall using the weighted F1-score that has been used in similar work (Abdellatif, Badran, Costa, & Shihab, 2021). More specifically, we compute the F1-score for each intent and aggregate all intents F1-score (i.e., Eq. 3) using the weighted average.

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3}$$

We consider the classes' support as weights to compute the weighted F1-score. Although we evaluate all three measures, we only present the weighted F1-score in the thesis. We add the precision, recall, and F1-score to the Appendix.

Chapter 5

Evaluation: AlphaBot and NLU's Performance

In this chapter, we evaluate our approach using an NLU platform and answer the question **"Does AlphaBot improve the NLU's performance?"** by assessing the gathered results from our experiments.

5.1 Motivation

Since user's queries are the primary source to train NLUs (Microsoft, 2021a; Rasa, 2021b), we want to assess AlphaBot's performance in labeling those queries. Having a practical and automated approach helps developers train their chatbots efficiently, especially when they interact with thousands of users' queries (Selvi, Saranya, Chidida, & Abarna, 2019). Moreover, automating the labeling process reduces the chatbot development costs (i.e., the cost of annotators to label all the queries from users). Therefore, the main goal of this research question is to determine the impact of using AlphaBot on the NLU's performance.

5.2 Evaluation Methodology

To achieve this, we use the AskGit dataset to evaluate Rasa's performance before and after applying AlphaBot. However, there is no original split in the AskGit dataset as users' queries might contain confidential information about their software repository (i.e., user data). To simulate a realistic situation where users pose questions to the chatbot, we randomly split the dataset into three blocks: training, validation, and test sets. The training and test sets are used to train and evaluate Rasa, respectively, this serves as the baseline for our study. We consider the queries in the validation split as the users' inputs to the chatbot. Therefore, we apply AlphaBot to all queries in the validation set. More specifically, each query in the validation set is provided as an input to AlphaBot to predict its intent. Then, we merge the output of AlphaBot (i.e., labeled queries) with the train set to train Rasa; we refer to this model as AlphaBot_Rasa. Finally, we use the test set to evaluate AlphaBot_Rasa's performance when applying AlphaBot.

To assess the AlphaBot effectiveness when it is applied at different stages of chatbot lifetime (i.e., from early releases to maturity), we evaluate the AlphaBot on different sizes of training, validation, and test sets. In particular, we split the AskGit dataset incrementally 10% each time where this split is used to train Rasa, and the rest are used as the validation and test sets. For example, we divide the AskGit dataset to 10% for training and 90% for test and validation sets (45% each). Table 5.1 presents the percentage (columns 1-3) for training, validation, and test sets, respectively. It is important to emphasize that the ground truth of the validation set is not exposed to our approach, and we use the same test set to evaluate both models, the baseline and AlphaBot_Rasa. Moreover, we use the stratified split method to maintain a consistent distribution of intents across all splits.

5.3 Experimental Results

Table 5.1 presents the baseline and AlphaBot_Rasa's performance and percentage of performance improvement on each split in terms of F1-score. The results show that using AlphaBot improves Rasa's performance across all splits compared to the baseline, as shown

in Table 5.1. The peak in the performance improvement (44%) is at the lowest training set size (split 10%). This shows an advantage of using our approach, especially when it has little training data. As the size of the training split increases, the percentage of improvement decreases. This is expected as the NLUs tend to have better performance if trained on more queries (Abdellatif, Badran, Costa, & Shihab, 2021). That said, with a high number of training queries, using AlphaBot leads to a 3.2% improvement in the F1-score compared to the baseline.

Table 5.1: Percentage improvement in F1-score of AlphaBot_Rasa compared to the baseline at different sizes of training, test, and validation sets.

Training split (%)	**Testing split (%)**	**Validation split (%)**	**Baseline F1-score (%)**	**AlphaBot_Rasa F1-score (%)**	**Percentage of improvement (%)**
10	45	45	36.33	80.52	44.19
20	40	40	46.24	86.02	39.78
30	35	35	51.54	86.1	34.56
40	30	30	73.59	92.16	18.57
50	25	25	82.37	92.83	10.46
60	20	20	83.63	91.82	8.19
70	15	15	81.94	88.58	6.64
80	10	10	87.25	93.24	5.99
90	5	5	82.05	85.25	3.2

To better understand the reasons for the improvement in the performance when using AlphaBot, we examine the labeled queries by AlphaBot. More specifically, we examine the accuracy of AlphaBot in labeling its input queries (i.e., queries in the validation set). Table 5.2 presents the number of correctly classified queries to the total number of queries in that split. Overall, we find that AlphaBot classifies the intents for most of the queries correctly across all splits. AlphaBot mislabels, in total, 13 cases (queries). Upon closer examination of those queries, we find that the main reason for the misclassification of queries is that they do not satisfy the conditions implemented in the LFs. Another reason is that there are misspellings in the queries, which causes them to be misclassified. For example, "Which

branch is defualt out of the ones we have?" is misclassified as *ListBranches* intent instead of *DefaultBranch* intent because the word 'defualt' is misspelled. Consequently, the LF of *DefaultBranch* intent returns 'ABSTAIN' as one of the conditions is not met (the existence of word 'default') in the query. This implies that practitioners need to consider typos when developing their LFs. Overall, the results show that the improvement in performance is due to training Rasa on queries that are correctly labeled by AlphaBot.

Table 5.2: Number of queries that are labeled correctly by our weak supervision approach.

Training split (%)	**Testing split (%)**	**Validation split (%)**	**Correctly labeled / total no. of queries (%)**
10	45	45	333/337 (98.8)
20	40	40	299/300 (99.7)
30	35	35	260/262 (99.2)
40	30	30	223/225 (99.1)
50	25	25	186/187 (99.5)
60	20	20	148/150 (98.7)
70	15	15	112/112 (100)
80	10	10	74/75 (98.7)
90	5	5	23/23 (100)

AlphaBot yields the best performance (44.19% improvement in F1-Score) when the training data is limited. Besides reducing the cost of annotating the human queries, AlphaBot labels, on average, more than 99% of queries correctly.

As we have observed that AlphaBot has helped improve the performance of NLU. The increase in performance is considerable, especially when the model has little training data. In the next chapter, we will investigate The impact of the number of labeling functions on

performance of the NLU under study and and answer the question “What is the impact of the number of labeling functions on performance?” by evaluating the gathered results from the conducted experiments.

Chapter 6

Evaluation: AlphaBot and Number of Labeling Functions

In this chapter, we assess the impact of the number of labeling functions and answer the question **"What is the impact of the number of labeling functions on performance?"** by evaluating the gathered results from the conducted experiments.

6.1 Motivation

Given that using AlphaBot significantly improves the NLU's performance, as shown in RQ1. Nothing comes free, crafting the LFs to automate the labeling process requires some effort and domain expertise, though they are cheaper than hand-labeling the user's queries regarding the time needed. Therefore, in this research question, we want to examine the impact of the number of implemented LFs on the NLUs' performance. In other words, we want to examine how many LFs are needed to achieve acceptable performance in the NLU. This helps chatbot developers to determine the number of required LFs to achieve the desired performance.

6.2 Evaluation Methodology

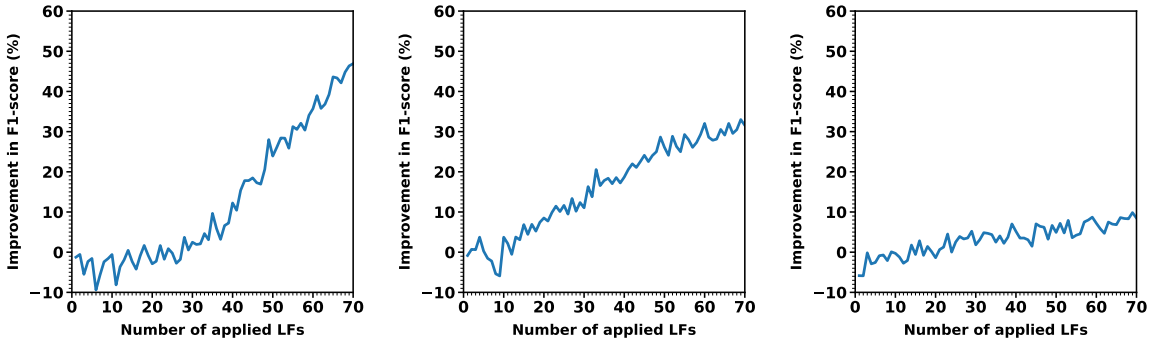
To accomplish this, we need to progressively assess the impact of adding LFs on Rasa's performance. Thus, we construct a set of LFs by randomly selecting one LF from our 70 LFs and adding it to the set. The set contains all the 70 LFs that are randomly shuffled. We use the same RQ1 splits to assess the impact of adding one LF at a time from the shuffled LFs list on Rasa's performance. More specifically, we apply the first LF in the LFs set to the queries in the validation set and merge the labeled queries to the training dataset, we refer to the merged dataset as the augmented dataset. Next, we retrain Rasa (AlphaBot_Rasa) on the augmented dataset and evaluate AlphaBot_Rasa's performance using the test set in the split. The impact of applying the LF is measured through the difference in the performance between the (AlphaBot_Rasa) and the baseline (Rasa's performance before applying any LFs for that split). Then, we additively choose the next LF from the shuffled LFs list to apply it to the queries in the validation set and repeat the same evaluation process to measure the impact of adding more LFs on Rasa's performance. We repeat the same steps till all LFs in the LFs list are applied for that split.

Since the LFs list is created randomly, the order of applying the LFs might impact the NLU's performance (e.g., applying the LFs associated with intents that have more queries in the test set). To reduce the randomness effect of the applied LFs order, we create another two randomly shuffled LFs lists and repeat the same experiment. In total, we have 210 runs for 70 LFs in the three LFs lists for each split. To make our study manageable, we perform our experiment using the 10%, 30%, and 50% training splits from RQ1. We believe that the selected splits aligned with our study goal, helping chatbot developers to improve the chatbot performance at the early releases of chatbot. For each split, we report the average of all runs of the three LFs lists.

6.3 Experimental Results

In this section, we discuss the experimental results of our evaluation. Figure 6.1 presents the performance trend in terms of F1-score when applying LFs additively on 10%, 30%, and

50% splits compared to the baseline. From the figure, we notice that adding more LFs yields a performance increase. This happens because Rasa is trained on more correctly labeled queries. In other words, applying more LFs leads to more labeled queries used to train Rasa. Another observation is that Rasa's performance falls in some cases even when more LFs are applied. For example, in Figure 6.1b, the performance decreases when applying the 41st LF that adds four correct training queries compared to the previous 40 LFs. One explanation for this behavior is that there is randomness when training Rasa, as it is based on deep learning models used for intents classification, which leads to a decrease in the performance.



(a) 10% training split (b) 30% training split (c) 50% training split

Figure 6.1: Analysis of performance trend in terms of F1-score when applying LFs in an additive manner.

Moreover, we observe that applying more LFs on splits with fewer training data achieves better performance improvement compared to splits with more training data. For example, Rasa shows more performance improvement in the 10% training split compared to 30% and 50% splits as shown in Figure 6.1. This is expected as Rasa becomes more robust when is trained on more data. This is aligned with our study goal to support practitioners in improving the performance of their chatbots when they are trained on fewer labeled queries (i.e., the early releases of the chatbot).

Although our results are promising when adding more LFs randomly, we believe that adding LFs for the main intents of the chatbot leads to higher performance. For example, if there is a chatbot that performs code refactoring, then it is unlikely that a user asks about the number of branches that the repository has. Therefore, practitioners need to focus on

adding LFs that label users' queries related to refactoring intents (e.g., refactoring a class, or a method).

Overall, applying more LFs leads to better NLU's performance. In case the NLU is trained on fewer training queries, the increase in the performance is higher compared to NLU trained with more queries.

In this chapter, we have evaluated the impact of the number of LFs on the performance of studied NLU. We now move our investigation towards evaluating our approach on another well-known NLU and repeat the same experiments discussed in Chapter 5 and this chapter using that NLU, in the next chapter.

Chapter 7

Discussion

We use the Rasa platform to evaluate AlphaBot as discussed in Chapter 4. In this chapter, we want to examine whether our approach could improve other NLUs' performance. Therefore, we rerun our experiment using Dialogflow platform (Dialogflow, 2021). Dialogflow is a cloud-based NLU platform developed by Google. Dialogflow supports more than 30 languages and can be integrated with popular communication channels (Dialogflow, 2021). In addition, it provides a graphical user interface (GUI) and API to facilitate training and testing of the NLU. Moreover, it has been used by prior works to develop the SE chatbots (Abdellatif, Badran, & Shihab, 2020; Paikari et al., 2019; Qasse, Mishra, & Hamdaqa, 2021).

7.1 Rerunning the First Experiment Using Google Dialogflow

To evaluate the impact of AlphaBot on Dialogflow's performance, we use the same experiment settings, dataset, and splits discussed in Chapter 5 and Chapter 6. In particular, we assess the Dialogflow's performance when using AlphaBot (Chapter 5) and evaluate the impact of adding more LFs on the performance trend (Chapter 6). It is worth noting that we assess the Dialogflow's performance on 10%, 30%, and 50% training splits. This is because it is expensive to run our experiment on all splits using a cloud-based NLU platform (i.e., Dialogflow). Another reason for selecting splits with a few training queries is to assess the

main goal of our approach, helping chatbot practitioners to boost the performance of the chatbot at early releases. Also, allow developers to focus on the core functionalities of the chatbot rather than annotating users' queries.

Table 7.1: Percentage of F1-score improvement when applying AlphaBot on Google Dialogflow.

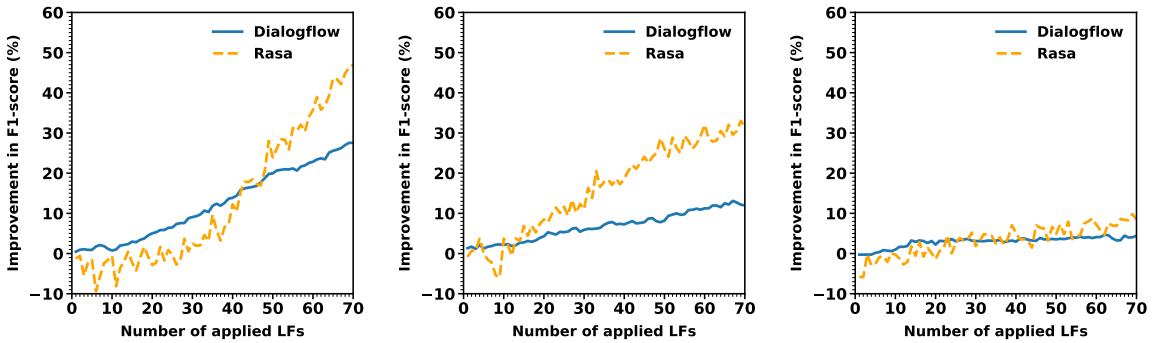
Training split (%)	**Testing split (%)**	**Validation split (%)**	**Baseline F1-score (%)**	**AlphaBot_Dialogflow F1-score (%)**	**Percentage of improvement (%)**
10	45	45	59.00	86.08	27.08
30	35	35	74.45	85.79	11.33
50	25	25	81.60	84.02	2.41

Table 7.1 presents F1-scores for Baseline (Dialogflow before applying AlphaBot), AlphaBot_Dialogflow, and the percentage of improvement after applying our approach. Similar to results in Chapter 5 with Rasa, AlphaBot increases the performance of Dialogflow in terms of F1-score for all splits. The peak performance increase is at splits with fewer training queries (i.e., 10% training split). Moreover, the improvement decreases as Dialogflow is trained on more queries (50% training split) as it becomes more robust.

Applying AlphaBot to Dialogflow yields to 27.08% improvement in F1-Score when the training data is limited. It also improves the performance of other splits on Dialogflow.

7.2 Rerunning the Second Experiment Using Google Dialogflow

Figure 7.1 shows the impact of applying LFs additively on the Dialogflow's performance. Overall, we notice that the performance trend increases as more LFs are applied. Another observation is that there is more performance improvement when applying more LFs on splits with fewer training queries. In fact, we have the same observations for Rasa as discussed in RQ2. Our findings show that our approach could be generalized for other NLU platforms.



(a) 10% training split (b) 30% training split (c) 50% training split

Figure 7.1: Results of applying the LFs additively on Dialogflow's F1-score.

Overall, applying more LFs leads to better Dialogflow's performance. Same as the finding described in Chapter 6, in case the NLU is trained on fewer training queries, the increase in the performance is higher compared to NLU trained with more queries.

In this chapter, we have evaluated our approach on another well-known NLU and repeated the same experiments discussed in Chapter 5 and this Chapter 6 using Google Dialogflow. In the next chapter, we discuss the threats of validity.

Chapter 8

Threats to Validity

As with other empirical studies, there are threats to validity, which may affect our study. In this chapter, we discuss the threats to the validity of our findings, broken down by internal, construct, and external validity.

8.1 Internal Validity

Concerns confounding factors that could have influenced our results. We develop LFs to label queries using the collected features discussed in Chapter 3, which might introduce human bias in crafting those LFs. To mitigate this bias, the author of this thesis and one another domain expert independently develop the LFs for all intents in the dataset (i.e., 52 intents) and then discuss each LF to reach a consensus on the best LFs that yield to best results. Another threat to internal validity is that the NLUs use deep learning models, which introduces randomness in the training process of these models. Thus, it might bias the results and conclusions in our study. To alleviate this threat, we repeat the experiment with the same settings twice and report the average of the two runs. Finally, we generate random orders of LFs lists in RQ2. Applying different orders of LFs might lead to different results. Therefore, we perform the same experiment using three randomly shuffled LFs lists and report the averages of those three experiments.

8.2 Construct Validity

To conduct this study and evaluate NLUs' performance, we calculate the standard classification accuracy measures - precision, recall, and F1-score. As described in Chapter 4, in our study, recall (shown using Eq. 4) is the percentage of the correctly classified queries to the total number of queries for that intent in the oracle.

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

Also, the precision shown in Eq. 5 is the percentage of the correctly classified queries to the total number of classified queries for the intent.

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

Finally, we combine both precision and recall using the weighted F1-score that has been used in similar work (Abdellatif, Badran, Costa, & Shihab, 2021). More specifically, we compute the F1-score for each intent and aggregate all intents F1-score (i.e., Eq. 6) using the weighted average.

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6}$$

We consider the classes' support as weights to compute the weighted F1-score. Although we evaluate all three measures, we only present the weighted F1-score in the thesis. The other metrics (i.e., precision, recall, and F1-score) are presented in Appendix. Although we utilize the same metrics that have been used in prior works (e.g., (Abdellatif, Badran, Costa, & Shihab, 2021; Braun et al., 2017)), other metrics also can be used to measure the changes in NLU's performance.

8.3 External Validity

Concerns about the generalization of our findings. We use the AskGit dataset to evaluate AlphaBot. Thus, our results may not generalize to other datasets. To the best of our knowledge, this is the only publicly available dataset that contains enough queries (749 queries) and intents (52 intents) for an SE chatbot, which enables us to evaluate AlphaBot performance in a large dataset. Since our approach achieves promising results, we are planning to put it into practice by integrating it with AskGit.

To evaluate the impact of using AlphaBot on NLU's performance, we perform a case study using the Rasa platform and replicate the same experiment using Dialogflow. This might affect the generalizability of our results. However, Rasa and Dialogflow are commonly used by chatbot developers and researchers. Also, the selected NLUs cover the open-source and closed-source NLUs which increase the generalizability of our results. That said, we plan (and encourage other researchers) to evaluate AlphaBot using more NLUs and datasets.

Chapter 9

Conclusion and Future Works

In this chapter, we summarize the conclusions and contributions of this thesis, and provide some possible future works for this study.

9.1 Conclusion

Chatbots are tremendously used to help software practitioners in their development tasks. Every chatbot relies on the NLU component to understand the user's input. Practitioners reported that they struggle to train NLUs due to the lack of data. One way to have more data is to train the NLUs on users' queries posed to the chatbot. However, annotating such queries requires dedicated effort and time. To help chatbot practitioners in this tedious and time-consuming task, we propose AlphaBot, an approach that annotates the user's queries using weak supervision. We evaluate AlphaBot using a dataset that composes of 749 queries representing 52 intents. Our results show that AlphaBot helps chatbot practitioners to boost the NLU's performance at early releases of their chatbots (i.e., fewer training queries). In particular, we find that our approach increases the NLU's performance up to an average of 17.16% compared to the baseline. Also, the results show that AlphaBot annotates, on average, 99% of queries correctly. Finally, we find that adding more labeling functions increases NLU's performance, especially when the NLU is trained on fewer training queries.

Chatbot practitioners can use our approach to annotate the queries posed to the chatbot. Thus, it reduces the cost and time for creating large labeled training datasets. Nevertheless, we plan in the future to evaluate AlphaBot using more NLUs. Moreover, we intend to propose an approach that creates LFs based on the input data and can be integrated with AlphaBot.

To sum up, our work contributes to practitioners and the research community by:

- Proposing a weak supervision-based approach to automate the labeling process of the user's queries for chatbots in the software engineering domain.
- Conducting an empirical study to evaluate our approach where 1) we examine the NLU's performance when applying our approach. 2) We explore the impact of adding more labeling functions on the performance trend of the NLU.
- Making our approach implementation and datasets publicly available (Farhour et al., 2021) to enable replication and accelerate future research in the field.

9.2 Future Works

While this thesis makes a new contribution to the current literature on chatbots in the software engineering domain, more avenues can be pursued to continue on this thread of research. We explain these avenues in the following subsections.

9.2.1 Evaluating Other NLU Platforms

While we use two well-known NLU platforms in this thesis (i.e., Rasa and Google Dialogflow), there are many other comprehensive NLU platforms available to use (e.g., IBM's Watson Conversation Service (IBM, 2021), Microsoft LUIS (Microsoft, 2021b), Wit.ai (Wit, 2021), etc.) by practitioners. One interesting avenue for future work could be a comparative study of these different NLU platforms to evaluate the effectiveness of AlphaBot on each of them.

9.2.2 An Approach for Automated Creation of Labeling Functions

In our study, we manually developed and implemented 70 LFs using Snorkel to cover all intents in the dataset for automated labeling the input queries. However, one exciting avenue as a future work could be proposing an approach for the automated creation of LFs using the features in the input data. This could be done by utilizing feature engineering and data mining techniques. By automated creation of some LFs for chatbots' datasets, the time needed for developing LFs could be significantly reduced.

9.2.3 Extending AlphaBot to Other Areas

In this thesis, we develop and implement AlphaBot to improve chatbots for code repositories. Although studying code repositories is one of the interesting research areas in software engineering, many other areas in this domain can be investigated and studied. Therefore, as another interesting future work, AlphaBot could be extended and applied to different areas in software engineering.

9.2.4 Using Neural Networks to Increase AlphaBot's Accuracy

As discussed in Section 3.4, there are two methods that Snorkel uses to finalize the label of the query: 1) Majority vote, and 2) Label model. As mentioned in Section 4.3, in this study, we configure Snorkel to use the Majority Vote model to identify the final label (intent) for the query since most of the intents (65%) have only one LF. However, in the Label model, Snorkel estimates the accuracies and correlation structure of the LFs using a generative model (Ratner et al., 2020) without accessing the ground truth (Bach et al., 2017). Thus, Snorkel weights the LFs by their true accuracies and eliminates the noise caused by some LFs outputs (Ratner et al., 2020). In case there are many LFs for the same intent and there are expected significant correlations/conflicts between the LFs, then the Label model is recommended (Ratner et al., 2020). Otherwise, the Majority model is preferred when there are few LFs for each intent.

Based on the mentioned difference between the two models implemented in Snorkel, in

a **larger dataset** with more correlated/conflicted implemented LFs, using the Label model in Snorkel is preferred. Therefore, an interesting future work could be extending AlphaBot by using larger datasets and utilizing the Label model for the automated labeling process. Also, the probabilistic training labels created by the Label model can be used to train a classifier weak supervision tasks. The output of the Snorkel Label model is just a set of predicted labels, which can be used with most popular machine learning libraries (e.g., such as TensorFlow (Abadi et al., 2016), Keras (Chollet et al., 2018), PyTorch (Paszke et al., 2019), etc.) for performing supervised learning. Therefore, a classifier trained using a weakly supervised dataset could exceed an approach based on the LFs alone as it learns to generalize above the noisy heuristics (i.e., the implemented LFs) created by domain experts using Snorkel.

9.2.5 An Empirical Study by Using AlphaBot in Production

As explained in Chapter 3, the queries labeled by AlphaBot can be added to the chatbot's training dataset to retrain the NLU using this augmented dataset and improve the NLU's performance. Therefore, one interesting future work is to use AlphaBot in production to label users' queries in a large code repository and extend the dataset for further NLU training and AlphaBot-related experiments. By using a larger dataset produced by a large number of users, a new empirical study can be conducted.

Appendix A

Supplementary Measures of the Experiments

In this appendix, we include the precision and recalls values for the experiments described in Chapter 5 and Chapter 7. Moreover, Table A.3 presents the distribution of queries across all 52 intents.

Table A.1: Precision and recall values of the second model compared to the baseline model for the Rasa experiment in Chapter 5.

Training split (%)	Testing split (%)	Validation split (%)	Baseline precision (%)	Baseline recall (%)	AlphaBot_Rasa precision (%)	AlphaBot_Rasa recall (%)
10	45	45	39.86	41.42	81.88	82.98
20	40	40	48.99	52	86.19	88
30	35	35	51.56	59.69	88.08	87.64
40	30	30	72.76	78.22	93.28	92.88
50	25	25	87.73	88.83	88.94	88.7
60	20	20	83.80	86	92.99	92.66
70	15	15	82.07	84.95	87.68	91.15
80	10	10	85.22	90.66	92.88	94.66
90	5	5	79.80	86.53	83.65	88.46

Table A.2: Precision and recall values of the second model compared to the baseline model for the first Dialogflow experiment in Chapter 7.

Training split (%)	Testing split (%)	Validation split (%)	Baseline precision (%)	Baseline recall (%)	AlphaBot_Dialogflow precision (%)	AlphaBot_Dialogflow recall (%)
10	45	45	63.14	55.38	87.24	84.96
30	35	35	77.88	71.33	89.76	82.16
50	25	25	82.51	80.73	85.93	82.20

Table A.3: The distribution of queries across 52 intents.

Intent	Definition	No. of examples
ProjectCollaborators	Presents the developer(s) who contributed to the project.	30
StarsCount	Presents the number of stars for the repository.	29
IssueRelatedCommits	Determines the commits linked to a certain bug.	22
FileCreator	Identifies the developer(s) who creates a specific file in the repository.	21
IssueContributors	Determines the developer(s) who contributes in fixing a specific bug.	17
ModifiedFilesPR	Identifies the files that are touched by a specific pull-request.	12
ActivityReport	Presents statistics about the repository activities occurred in the last day (e.g., number of solved bugs).	10
LongestOpenPR	Identifies the longest pull-request which is still opened.	9
CommitsCountInBranch	Determines the number of commits in a certain branch.	7
OverloadedDev	Identifies overloaded developer(s) with the highest number of unfixed bugs.	6
LargestFile	Demonstrates the largest file in the repository.	9
LastDeveloperTouchFile	Identifies the last developer who changed the file.	7
IssuesCount	Presents the number of issues in the repository.	12
IssueAssignees	Determines the assignee(s) of a specific issue.	6
IssueCreationDate	Identifies the creation date of a specific issue.	11
IssueClosingDate	Presents the closing date of a specific issue.	14
IssueCreator	Determines the person who created a specific issue.	14
IssueCloser	Determines the person who closed a specific issue.	15
MostRecentIssues	Presents the most recent issues of the repository.	13
LongestOpenIssue	Identifies the longest open issue of the repository.	10
CommitsCount	Presents the number of commits in the repository.	29
ForksCount	Presents the number of forks in the repository.	29
BranchesCount	Presents the number of branches in the repository.	28
SubscribersCount	Presents the number of subscribers of the repository.	29
WatchersCount	Presents the number of watchers of the repository.	15
DownloadsCount	Presents the number of downloads of the repository.	32
DefaultBranch	Determines the default branch of the repository.	15
BranchesList	Presents all the branches in the repository.	22
MainProgrammingLanguage	Provides the main programming language used in the repository.	15
RepositoryOwner	Determines the owner of the repository.	15
RepositoryLicense	Provides the license of the repository.	16
RepositoryTopics	Identifies all the topics of the repository.	18
RepositoryCreationDate	Determines the creation date of the repository.	19
LatestRelease	Provides the latest release in the repository.	9
ReleasesList	Provides all the releases in the repository.	6
CollaboratorsCount	Presents the number of collaborators of the repository.	8
LanguagesList	Provides all the languages used in the repository.	8
IntialCommit	Determines the initial commit in the repository.	11
IntialCommitInBranch	Determines the initial commit in a specific branch.	8
LatestCommit	Determines the latest commit in the repository.	9
LatestCommitInBranch	Determines the latest commit in a specific branch.	9
TopContributors	Identifies the top contributors in the repository.	13
ContributionsByDeveloper	Lists all the contributions by a specific developer.	8
PullRequestsCount	Presents the number of pull-requests in the repository.	11
PullRequestAssignees	Lists all the assignees for a specific pull-request.	7
PullRequestCreationDate	Determines the creation date of a specific pull-request.	11
PullRequestClosingDate	Determines the closing date of a specific pull-request.	13
PullRequestContributors	Provides the list of contributors for a specific pull-request.	14
PullRequestCreator	Provides the person who created a specific pull-request.	14
PullRequestCloser	Provides the person who closed a specific pull-request.	14
MostRecentPR	Lists the most recent pull-requests.	12
CommitsInPR	Lists the commits in a specific pull-request.	8

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... others (2016). {TensorFlow}: A system for {Large-Scale} machine learning. In *12th usenix symposium on operating systems design and implementation (osdi 16)* (pp. 265–283).

Abdellatif, A., Badran, K., Costa, D., & Shihab, E. (2021). 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., & Shihab, E. (2020). Msrbot: Using bots to answer questions from software repositories. *Empirical Software Engineering (EMSE)*, *25*, 1834-1863.

Abdellatif, A., Badran, K., & Shihab, E. (2021, August). *Askgit.* https://askgit.io/. ((Accessed on 08/03/2021))

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).

Arachie, C., & Huang, B. (2021). A general framework for adversarial label learning. *Journal of Machine Learning Research*, *22*(118), 1-33. Retrieved from http://jmlr.org/papers/v22/20-537.html

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

Bach, S. H., He, B., Ratner, A., & Ré, C. (2017). Learning the structure of generative models without labeled data. In *International conference on machine learning* (pp. 273–282).

Bach, S. H., Rodriguez, D., Liu, Y., Luo, C., Shao, H., Xia, C., ... Malkin, R. (2019). Snorkel drybell: A case study in deploying weak supervision at industrial scale. In *Proceedings of the 2019 international conference on management of data* (p. 362–375). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3299869.3314036 doi: 10.1145/3299869.3314036

Bhadane, C., Dalal, H., & Doshi, H. (2015). Sentiment analysis: Measuring opinions. *Procedia Computer Science*, *45*, 808-814. Retrieved from https://www.sciencedirect.com/science/article/pii/S1877050915003956 (International Conference on Advanced Computing Technologies and Applications (ICACTA)) doi: https://doi.org/10.1016/j.procs.2015.03.159

Bharti, U., Bajaj, D., Batra, H., Lalit, S., Lalit, S., & Gangwani, A. (2020). Medbot: Conversational artificial intelligence powered chatbot for delivering tele-health after covid-19. In *2020 5th international conference on communication and electronics systems (icces)* (p. 870-875). doi: 10.1109/ICCES48766.2020.9137944

Bradley, N. C., Fritz, T., & Holmes, R. (2018). Context-aware conversational developer assistants. In *Proceedings of the 40th international conference on software engineering* (p. 993–1003). 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

C., M. (2016, 11). *Google's hand-fed ai now gives answers, not just search results.* https://www.wired.com/2016/11/googles-search-engine-can-now-answer-questions-human-help/. ((Accessed on 07/04/2021))

Chollet, F., et al. (2018). Keras: The python deep learning library. *Astrophysics source code library*, ascl–1806.

Clarizia, F., Colace, F., Lombardi, M., Pascale, F., & Santaniello, D. (2018). Chatbot:

An education support system for student. In A. Castiglione, F. Pop, M. Ficco, & F. Palmieri (Eds.), *Cyberspace safety and security* (pp. 291–302). Cham: Springer International Publishing.

Davis, A. P., Wiegers, T. C., Roberts, P. M., King, B. L., Lay, J. M., Lennon-Hopkins, K., ... others (2013). A ctd–pfizer collaboration: manual curation of 88 000 scientific articles text mined for drug–disease and drug–phenotype interactions. *Database*, *2013*.

Dialogflow. (2021, August). *Dialogflow official website.* https://dialogflow.cloud.google.com/. ((Accessed on 08/03/2021))

Dominic, J., Houser, J., Steinmacher, I., Ritter, C., & Rodeghero, P. (2020). Conversational bot for newcomers onboarding to open source projects. In *Proceedings of the ieee/acm 42nd international conference on software engineering workshops* (p. 46–50). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3387940.3391534 doi: 10.1145/3387940.3391534

Facebook. (2021, July). *Duckling.* https://github.com/facebook/duckling. ((Accessed on 07/04/2021))

Farhour, F., Abdellatif, A., Mansour, E., & Shihab, E. (2021, Nov). *A weak supervision-based approach to improve chatbots for code repositories.* https://github.com/CoDS-GCS/AlphaBot. ((Accessed on 11/17/2021))

Fries, J. A., Steinberg, E., Khattar, S., Fleming, S. L., Posada, J., Callahan, A., & Shah, N. H. (2021). Ontology-driven weak supervision for clinical entity classification in electronic health records. *Nature Communications*, *12*(1), 2017. Retrieved from https://doi.org/10.1038/s41467-021-22328-4 doi: 10.1038/s41467-021-22328-4

Geng, X., & Smith-Miles, K. (2009). Incremental learning. In S. Z. Li & A. Jain (Eds.), *Encyclopedia of biometrics* (pp. 731–735). Boston, MA: Springer US. Retrieved from https://doi.org/10.1007/978-0-387-73003-5_304 doi: 10.1007/978-0-387-73003-5_304

Gomes, S. R., Saroar, S. G., Mosfaiul, M., Telot, A., Khan, B. N., Chakrabarty, A., & Mostakim, M. (2017). A comparative approach to email classification using naive bayes classifier and hidden markov model. In *2017 4th international conference on advances*

in electrical engineering (icaee) (p. 482-487). doi: 10.1109/ICAEE.2017.8255404

GoodRebels. (2021, July). *The impact of conversational bots in the customer experience.* https://www.goodrebels.com/the-impact-of-conversational-bots-in-the-customer-experience/. ((Accessed on 07/03/2021))

Hancock, B., Bordes, A., Mazare, P.-E., & Weston, J. (2019a). Learning from dialogue after deployment: Feed yourself, chatbot! *arXiv preprint arXiv:1901.05415*.

Hancock, B., Bordes, A., Mazare, P.-E., & Weston, J. (2019b, July). Learning from dialogue after deployment: Feed yourself, chatbot! In *Proceedings of the 57th annual meeting of the association for computational linguistics* (pp. 3667–3684). Florence, Italy: Association for Computational Linguistics. Retrieved from https://www.aclweb.org/anthology/P19-1358 doi: 10.18653/v1/P19-1358

IBM. (2021, August). *Ibm watson official website.* https://www.ibm.com/watson/. ((Accessed on 08/03/2021))

Jianqiang, Z., & Xiaolin, G. (2017). Comparison research on text pre-processing methods on twitter sentiment analysis. *IEEE Access*, *5*, 2870-2879. doi: 10.1109/ACCESS.2017.2672677

jsphdnl. (2017, 08). *nlp - conversational data for building a chat bot - stack overflow.* https://stackoverflow.com/questions/45821517/conversational-data-for-building-a-chat-bot. ((Accessed on 07/09/2021))

Knave, P. (2021, May). *reactjs - how to create css underline which partially covers the word? - stack overflow.* https://stackoverflow.com/questions/67694697/how-to-create-css-underline-which-partially-covers-the-word. ((Accessed on 05/25/2021))

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

Lin, C.-T., Ma, S.-P., & Huang, Y.-W. (2020). Msabot: A chatbot framework for assisting in the development and operation of microservice-based systems. In *Proceedings of the ieee/acm 42nd international conference on software engineering workshops* (p. 36–40). New York, NY, USA: Association for Computing Machinery. Retrieved from https://

doi.org/10.1145/3387940.3391501 doi: 10.1145/3387940.3391501

Mallinar, N., Shah, A., Ugrani, R., Gupta, A., Gurusankar, M., Ho, T. K., ... others (2019a). Bootstrapping conversational agents with weak supervision. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 33, pp. 9528–9533).

Mallinar, N., Shah, A., Ugrani, R., Gupta, A., Gurusankar, M., Ho, T. K., ... McGregor, B. (2019b, Jul.). Bootstrapping conversational agents with weak supervision. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 9528-9533. Retrieved from https://ojs.aaai.org/index.php/AAAI/article/view/5011 doi: 10.1609/aaai.v33i01.33019528

Meng, Y., Shen, J., Zhang, C., & Han, J. (2018). Weakly-supervised neural text classification. In *Proceedings of the 27th acm international conference on information and knowledge management* (p. 983–992). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3269206.3271737 doi: 10.1145/3269206.3271737

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

Ng, E. L. B. A. (2017, 1). *on the future of artificial intelligence.* https://time.com/4631730/andrew-ng-artificial-intelligence-2017/. ((Accessed on 07/04/2021))

Okuda, T., & Shoda, S. (2018, 04). Ai-based chatbot service for financial industry. *Fujitsu Scientific and Technical Journal*, *54*, 4-8.

Oramas, S., Quadrana, M., & Gouyon, F. (2021a). Bootstrapping a music voice assistant with weak supervision. In *Proceedings of the 2021 conference of the north american chapter of the association for computational linguistics: Human language technologies: Industry papers* (pp. 49–55).

Oramas, S., Quadrana, M., & Gouyon, F. (2021b, June). Bootstrapping a music voice

assistant with weak supervision. In *Proceedings of the 2021 conference of the north american chapter of the association for computational linguistics: Human language technologies: Industry papers* (pp. 49–55). Online: Association for Computational Linguistics. Retrieved from https://aclanthology.org/2021.naacl-industry.7 doi: 10.18653/v1/2021.naacl-industry.7

Paikari, E., Choi, J., Kim, S., Baek, S., Kim, M., Lee, S., ... van der hoek, A. (2019). A chatbot for conflict detection and resolution. In *2019 ieee/acm 1st international workshop on bots in software engineering (botse)* (p. 29-33). doi: 10.1109/BotSE.2019.00016

Parker, J., & Yu, S. (2021, August). Named entity recognition through deep representation learning and weak supervision. In *Findings of the association for computational linguistics: Acl-ijcnlp 2021* (pp. 3828–3839). Online: Association for Computational Linguistics. Retrieved from https://aclanthology.org/2021.findings-acl.335 doi: 10.18653/v1/2021.findings-acl.335

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... others (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, *32*.

Polikar, R., Upda, L., Upda, S., & Honavar, V. (2001). Learn++: an incremental learning algorithm for supervised neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *31*(4), 497-508. doi: 10.1109/5326.983933

Qasse, I. A., Mishra, S., & Hamdaqa, M. (2021). icontractbot: A chatbot for smart contracts' specification and code generation. In *Proceedings of the ieee/acm 43th international conference on software engineering workshops.* New York, NY, USA: Association for Computing Machinery.

Rao, N., Bansal, C., & Guan, J. (2021). Search4code: Code search intent classification using weak supervision. In *2021 ieee/acm 18th international conference on mining software repositories (msr)* (p. 575-579). doi: 10.1109/MSR52588.2021.00077

Rasa. (2021a, July). *Incremental training.* https://rasa.com/docs/rasa/next/migration

-guide#incremental-training. ((Accessed on 07/03/2021))

Rasa. (2021b, July). *Introduction to rasa x.* https://rasa.com/docs/rasa-x/. ((Accessed on 07/09/2021))

Rasa. (2021c, July). *Open source conversational ai | rasa.* https://rasa.com/. ((Accessed on 07/03/2021))

Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., & Ré, C. (2017). Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the vldb endowment. international conference on very large data bases* (Vol. 11, p. 269).

Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., & Ré, C. (2017, November). Snorkel: Rapid training data creation with weak supervision. *Proc. VLDB Endow.*, *11*(3), 269–282. Retrieved from https://doi.org/10.14778/3157794.3157797 doi: 10.14778/3157794.3157797

Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., & Ré, C. (2020). Snorkel: Rapid training data creation with weak supervision. *The VLDB Journal*, *29*(2), 709–730.

Ravì, D., Wong, C., Deligianni, F., Berthelot, M., Andreu-Perez, J., Lo, B., & Yang, G.-Z. (2017). Deep learning for health informatics. *IEEE Journal of Biomedical and Health Informatics*, *21*(1), 4-21. doi: 10.1109/JBHI.2016.2636665

Romero, R., Parra, E., & Haiduc, S. (2020). Experiences building an answer bot for gitter. In *Proceedings of the ieee/acm 42nd international conference on software engineering workshops* (p. 66–70). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3387940.3391505 doi: 10.1145/3387940.3391505

Ruffolo., M., & Visalli., F. (2020). A weak-supervision method for automating training set creation in multi-domain aspect sentiment classification. In *Proceedings of the 12th international conference on agents and artificial intelligence - volume 2: Icaart,* (p. 249-256). SciTePress. doi: 10.5220/0009165602490256

Selvi, V., Saranya, S., Chidida, K., & Abarna, R. (2019). Chatbot and bullyfree chat. In *2019 ieee international conference on system, computation, automation and networking (icscan)* (p. 1-5). doi: 10.1109/ICSCAN.2019.8878779

Şerban, D., Golsteijn, B., Holdorp, R., & Serebrenik, A. (2021, February 23). Saw-bot: Proposing fixes for static analysis warnings with github suggestions. In *Workshop on bots in software engineering.* United States: IEEE Computer Society.

Sheri. (2020, 07). *python - intent classification for chatbot - stack overflow.* https://stackoverflow.com/questions/62970861/intent-classification-for-chatbot. ((Accessed on 07/09/2021))

Shi, X., Hu, H., Che, W., Sun, Z., Liu, T., & Huang, J. (2020, Apr.). Understanding medical conversations with scattered keyword attention and weak supervision from responses. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(05), 8838-8845. Retrieved from https://ojs.aaai.org/index.php/AAAI/article/view/6412 doi: 10.1609/aaai.v34i05.6412

Shu, K., Mukherjee, S., Zheng, G., Awadallah, A. H., Shokouhi, M., & Dumais, S. (2020). Learning with weak supervision for email intent detection. In *Proceedings of the 43rd international acm sigir conference on research and development in information retrieval* (p. 1051–1060). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3397271.3401121

Snorkel-Team. (2020, 11). *Snorkel intro tutorial: Data labeling.* https://www.snorkel.org/use-cases/01-spam-tutorial. ((Accessed on 07/04/2021))

spaCy. (2021, July). *Industrial-strength natural language processing in python.* https://spacy.io/. ((Accessed on 07/03/2021))

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* (p. 1–6). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3173574.3173632 doi: 10.1145/3173574.3173632

Tran, K. (2020, 09). *nlp - building a chatbot about literary novel - stack overflow.* https://stackoverflow.com/questions/64007306/building-a-chatbot-about-literary-novel. ((Accessed on 07/09/2021))

Vijayarani, S., Ilamathi, M. J., Nithya, M., et al. (2015). Preprocessing techniques for text

mining-an overview. *International Journal of Computer Science & Communication Networks*, *5*(1), 7–16.

Wit. (2021, August). *Wit.ai official website.* https://wit.ai/. ((Accessed on 08/03/2021))

Yu, P., Ding, T., & Bach, S. H. (2021). Learning from multiple noisy partial labelers. *arXiv preprint arXiv:2106.04530*.

Zhang, N., Huang, Q., Xia, X., Zou, Y., Lo, D., & Xing, Z. (2020). Chatbot4qr: Interactive query refinement for technical question retrieval. *IEEE Transactions on Software Engineering*, 1-1. doi: 10.1109/TSE.2020.3016006

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