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

FARBOD FARHOUR, Concordia University, Canada
AHMAD ABDELLATIF, University of Calgary, Canada
ESSAM MANSOUR, Concordia University, Canada
EMAD SHIHAB, Concordia University, Canada

Software chatbots are growing in popularity and have been increasingly used in software projects due to their benefits in saving time, cost, and effort. At the core of every chatbot is a Natural Language Understanding (NLU) component that enables chatbots to comprehend the users' queries. Prior work shows that chatbot practitioners face challenges in training the NLUs because the labeled training data is scarce. Consequently, practitioners resort to user queries to enhance chatbot performance. They annotate these queries and use them for NLU training. However, such training is done manually and prohibitively expensive. Therefore, we propose AlphaBot to automate the query annotation process for SE chatbots. Specifically, we leverage weak supervision to label users' queries posted to a software repository-based chatbot. To evaluate the impact of using AlphaBot on the NLU's performance, we conducted a case study using a dataset that comprises 749 queries and 52 intents. The results show that using AlphaBot improves the NLU's performance in terms of F1-score, with improvements ranging from 0.96% to 35%. Furthermore, our results show that applying more labeling functions improves the NLU's classification of users' queries. Our work enables practitioners to focus on their chatbots' core functionalities rather than annotating users' queries.

CCS Concepts: • **Software and its engineering → Software libraries and repositories**; **Software notations and tools**.

Additional Key Words and Phrases: Software Chatbots, Weak Supervision, and Data Augmentation

ACM Reference Format:
Farbod Farhour, Ahmad Abdellatif, Essam Mansour, and Emad Shihab. 2024. A Weak Supervision-Based Approach to Improve Chatbots for Code Repositories. *Proc. ACM Softw. Eng.* 1, FSE, Article 105 (July 2024), 24 pages. https://doi.org/10.1145/3660812

1 INTRODUCTION

Numerous chatbots assist software practitioners in daily development tasks such as code conflict resolution [41], answering software development questions using Stack Overflow [54], managing tasks [65], and assisting newcomers in the onboarding to new projects [19]. 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) [4, 62].

Natural language understanding (NLU) is the backbone component in every chatbot as it allows chatbots to understand user's input [1]. NLUs use machine learning (ML) and NLP to extract structured information (the user's intent from the query and related entities) from the unstructured

Authors' Contact Information: Farbod Farhour, Concordia University, Montreal, Canada, farbod@farhour.com; Ahmad Abdellatif, University of Calgary, Calgary, Canada, ahmad.abdellatif@ucalgary.ca; Essam Mansour, Concordia University, Montreal, Canada, essam.mansour@concordia.ca; Emad Shihab, Concordia University, Montreal, Canada, emad.shihab@concordia.ca.

ACM 2994-970X/2024/7-ART105
https://doi.org/10.1145/3660812

input query. There are many off-the-shelf NLUs that developers can easily use and integrate into their chatbot implementation instead of developing one from scratch. For example, MSRBot [2] uses Dialogflow NLU to answer software project-related questions. MSABot [31] uses Rasa NLU to assist developers in managing microservices.

The main bottleneck of using any NLU is the need for training data. This is because 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. Prior work shows that proper NLU training improves the chatbot’s ability to respond to queries correctly [8, 23, 30]. In other words, training NLUs with many queries is of paramount importance since it allows the NLU to better classify the user’s query [1, 2]. However, previous studies point out that crafting and collecting training data is a challenging task [2, 4, 19, 24, 32]. For example, Dominic et al. [19] 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 [28, 58, 66]. One way to obtain more training data is to learn directly from the human-chatbot conversation. In other words, the chatbot developers need to manually monitor and annotate the user’s input 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 [35, 47]. 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., medical practitioners) to manually annotate users’ queries. In fact, there are large companies (e.g., Google, Amazon, IBM) who hire a full team to label training data for their products [15, 17, 38].

In recent years, researchers have proposed weak supervision approaches, part of machine learning that combines knowledge from weakly labeled data (user’s input to the user-chatbot conversation) to improve the chatbot’s performance. In particular, they use heuristics/rules to label the queries and re-train the machine learning model using these weakly labeled (noisy) data to improve the model’s performance [24, 32, 40, 69]. For example, Hancock et al. [24] developed a self-feeding approach to train a chit-chat chatbot on the posed query if the user’s satisfaction with the chatbot response is above a certain threshold. Mallinar et al. [32] developed a weak supervision based 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. Oramas et al. [40] proposed a heuristic weak supervision approach to label voice query logs. They evaluated the approach using actual voice traffic from a music streaming service. While these approaches help address the issue of manually labeling the data, none of these approaches target chatbots that operate in the SE domain. SE 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. Furthermore, some approaches require a lot of data to run since they depend on deep learning models, which makes these approaches inapplicable for SE chatbots due to the lack of training data [2, 4, 19]. On the other hand, there are approaches that require human interventions to run them. Finally, these approaches are evaluated to limited datasets with a few intents.

To address this gap, we present AlphaBot, a weak supervision-based approach to automatically annotate the intents of users’ 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

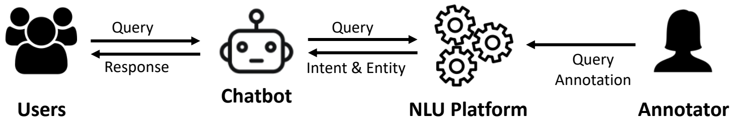


Fig. 1. Motivating example overview of user-chatbot interaction.

uses weak supervision (i.e., labeling functions) and information extracted from the query by the previous component to assign an intent for the query. It is important to note that all the steps in the AlphaBot's components are fully automated except constructing the labeling functions, which is done manually by a domain expert. To evaluate the proposed approach, we perform an empirical study using a dataset of a chatbot, called AskGit [3], which answers questions related to software repositories (e.g., "How many commits in the dev branch?"). The dataset contains 749 queries that represent 52 intents. More specifically, our study answers the following research questions:

RQ1: Does AlphaBot improve the NLU's performance? Our results show that AlphaBot, which uses labeling functions (i.e., rules and heuristics), annotates more than 99% of queries correctly on average. Furthermore, we find that using AlphaBot improves the NLU's performance in terms of F1-score (0.96% - 35% F1-score improvement), especially when the NLU is trained with a few number of queries. And the performance improvement decreases as the NLU becomes more robust, i.e., trained on a large dataset.

RQ2: 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 the F1-score. This is clearly shown when the NLU is trained with a small-sized dataset at early releases of the chatbot compared to when it is more robust (i.e., trained with many queries).

Our Contributions. To this end, this paper makes the following contributions:

- We propose a weak supervision-based approach to automate the labeling process of the user queries' intent for chatbots in the SE domain.
- We conduct an empirical study to evaluate our approach where we examine the performance of open-source (Rasa) and closed-source (Dialogflow) NLUs when applying AlphaBot. Furthermore, 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 [6] to enable replication and accelerate future research in the field.

Paper Organization. The rest of the paper is organized as follows. Section 2 provides an overview of the chatbots architecture and discusses a motivating example. Section 3 details our approach and its components. We describe our case study design to evaluate AlphaBot in Section 4. We report our results in Section 5. We discuss our findings in Section 6. Section 7 discusses the threat to validity. Section 8 presents the related work to our study and Section 9 concludes the paper.

2 BACKGROUND AND MOTIVATING EXAMPLE

Before diving deeper into our study, we provide an overview of how a chatbot works. Figure 1 presents an overview of a user-chatbot interaction. In this example, we opt to use a simplified chatbot architecture for illustration purposes, which has fewer components compared to full chatbot architecture [2]. Initially, the user asks the chatbot a question via natural language (e.g., "Who fixed

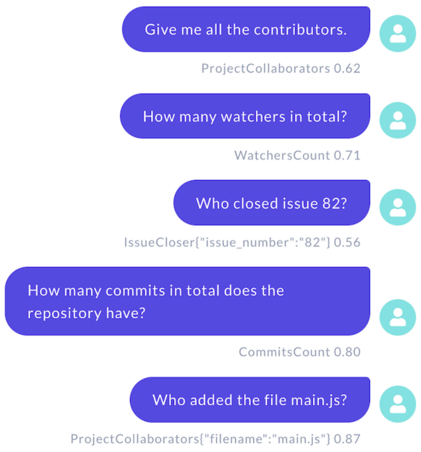

Give me all the contributors.
ProjectCollaborators 0.62
How many watchers in total?
WatchersCount 0.71
Who closed issue 82?
IssueCloser{"issue_number":"82"} 0.56
How many commits in total does the repository have?
CommitsCount 0.80
Who added the file main.js?
ProjectCollaborators{"filename":"main.js"} 0.87

Fig. 2. An example of user's queries recorded by Rasa with their classified intents.

issue 5?"). The chatbot forwards the user's query to the NLU to extract the intent (which represents the user's intention of the question) and the entities (which represent 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 return 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 [30].

As discussed in the previous section, 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 [1]. Therefore, to improve the NLU's accuracy, many NLUs record the users' queries and their extracted intents and entities (e.g., Google Dialogflow [18] and Microsoft LUIS [36]) 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 with 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 [35, 47], 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). Figure 2 presents a GUI provided by Rasa that displays the users' posed queries and their classified intents. The annotator needs to manually examine each query and confirm the classified intent in case it is correct, otherwise change the misclassified intent of a query to the correct intent. Also, the figure presents an example of queries stored by Rasa NLU with their classified intents and the classification confidence score, which presents how confident the NLU is in its intent classification, and it varies from 0 (i.e., not confident) to 1 (i.e., fully confident). For example, Rasa classified "How many commits in total does the repository have?" as *CommitsCount* intent with a 0.8 confidence score. From the figure, we observe that Rasa misclassifies "Who added the file main.js?" with *ProjectCollaborators* with a high confidence score of 0.87 while correctly classifying "Who closed issue 82?" as *IssueCloser* intent with a low confidence score (0.56). Consequently, the domain expert (annotator) needs to correct Rasa's intent classification of the "Who added the file main.js?" query to *FileCreator* intent and confirm the intent classification of the "Who closed issue 82?" query. This process is a burden and time-consuming because the chatbot developers need to dedicate time and effort to annotate each query posed to the chatbot to enhance the NLU's performance. 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 [52].

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.

2.1 Weak Supervision

Weak supervision is a part of machine learning that leverages noisy data to create larger training sets [49, 63]. More specifically, the practitioners leverage different forms of weak supervision to label the data. Then, they retrain their supervised models using the labeled data to enhance the performance of their models. There are different forms of weak supervision:

- **Heuristics:** Use rule-based technique to identify the class (or intent) to which the queries belong [71]. In other words, we determine the intent of a query if it has a specific keyword or pattern. For example, the intent of "Show me the refactor classes in my project" is *RefactoredClasses* intent because it has the word 'refactor', which is associated only with *RefactoredClasses* and not shared with other intents.
- **Distant supervision:** Identify the sentences that contain two entities that have a relation between them [5, 37]. For example, in the query "Fix NullPointerException in Android Studio", the entities 'NullPointerException' and 'Android Studio' enables us to identify that the user asks about fixing an exception in Android Studio.
- **Crowdsourced:** Outsourcing the data annotation task to the crowd where many domain experts manually label the data [25, 70]. Amazon Mechanical Turk platform[1] facilitates the crowdsourcing tasks.
- **Third-party models:** Use pretrained machine learning models to label the noisy data [33].

It is important to note that the different weak supervision forms are used to create or expand the training set from the noisy data to train supervised training models [50, 63]. Consequently, the trained model (the NLU in our study) becomes more robust because it has been trained on a large dataset and utilizes more features from the dataset. Also, it helps the model to generalize to new and unseen queries (i.e., data points), which differs from the rule-based classifiers, making it more rigid to classify the data points that are covered in their rule only.

3 ALPHABOT

Figure 3 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 (e.g., entities and part of speech) to be used for annotating the query, and 3) Intent labeler: uses the weak supervision-based approach to identify the intent of the query. AlphaBot takes the user's queries for the SE chatbot as input, and the output of AlphaBot is the query with its corresponding intent (i.e., labeled query). Finally, we augment the labeled query to

[1]https://www.mturk.com

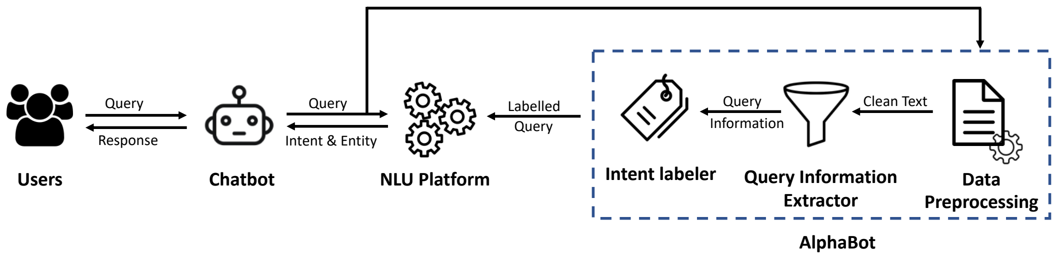


Fig. 3. An overview of AlphaBot and its components.

the chatbot's training dataset and retrain the NLU on the augmented dataset to improve the NLU's performance. We elaborate on each of these components in the following subsections.

3.1 Data Preprocessing

Data preprocessing is a critical step for NLP tasks [68]. 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 the steps described below:

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 [4, 11, 22, 27].

Remove unnecessary 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 platforms added 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>, <div>).

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 subsection.

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 [1], especially if the intents share the same characteristics (e.g., have mutual words in their queries). For example, if a 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 chatbot queries, “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 in a query 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?” [29]). 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. Among different types of weak supervision [73], 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, especially when it is trained with fewer training queries [1].

Ratner et al. [51] proposed Snorkel, a system that uses weak supervisiosn to enable 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 [7, 10, 20, 42]. Practitioners through Snorkel can provide higher-level supervision (e.g., heuristics, distant supervision, etc.) through labeling functions (LFs) that take domain knowledge and resources. Through LFs, practitioners can label large training datasets in hours or days rather than hand-label them over weeks or months. In Snorkel, each intent should have at least one LF corresponding with that intent. That said, the LFs do not have to label every data point [64]. LF is a code snippet that labels subsets of unlabeled data. For example, Figure 4 shows a LF for *ForksCount* intent that takes a user’s query posted to the 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

```python
# 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
```

Fig. 4. An LF for the ForksCount intent.

indicates the query is abandoned. 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.

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 or abstain) for the query. 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 [51] without accessing the ground truth [9]. Thus, Snorkel weights the LFs by their true accuracies and eliminates the noise caused by some LFs outputs [51]. The Label model is recommended if there are many LFs for the same intent and they have significant correlations/conflicts between these LFs [51]. Otherwise, the Majority model is preferred when there are few LFs for each intent.

In our study, we consider queries whose intent cannot be identified by the Intent Labeler component (i.e., all labeling functions return 'ABSTAIN' for the queries) as abandoned queries. It is important to note that AlphaBot will not augment the abandoned queries to the original training dataset[2] because the Intent Labeler component does not identify the intent of these queries. Instead, it will store the abandoned queries in a file to be manually labeled by the annotators. The chatbot developers can examine the abandoned queries in the file 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 original training dataset and retrain the NLU on the augmented dataset.

4 EVALUATION SETUP

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

[2]The initial dataset created by the chatbot developers to train the NLU.

improvement after applying our approach. In this section, we detail our selection of the dataset, weak supervision framework, NLU platform, and experiment design.

4.1 Dataset

To evaluate the performance of our approach, we opt for a dataset from the AskGit chatbot [3]. AskGit is a chatbot that answers questions related to software projects (e.g., "How many commits happened during March 2021?") on Slack. It is published on GitHub Marketplace[3] so that practitioners can install it on their software projects. The dataset used to train and evaluate AskGit was developed by the AskGit developers. Specifically, the developers brainstormed to create the initial training set for the intents supported by AskGit, representing various ways developers might inquire about each intent. Then, the AskGit developers piloted the chatbot with four practitioners from their network to gather additional training queries for each intent, expanding their final dataset. The AskGit developers employed this dataset to train and evaluate the NLU model of AskGit. The AskGit dataset contains 749 queries distributed across 52 intents. Furthermore, the size of the training examples varies among intents and for each individual intent, making this dataset suitable for evaluating the generalizability of AlphaBot on datasets with varying query sizes. Table 1 presents a snapshot of ten intents with their descriptions. We made the entire dataset, distribution of queries, and their length distribution across all 52 intents publicly available [6]. It is important to note that AskGit's users can ask the questions in different ways for each intent. 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 2 shows the distributions for each entity type in the dataset.

Several reasons motivate our selection of the AskGit dataset. To the best of our knowledge, this dataset is the most comprehensive (regarding the size) dataset that is publicly available for an SE chatbot. Furthermore, 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?"). 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. Finally, it allows us to evaluate the applicability of the AlphaBot by applying it to a chatbot in production (AskGit).

4.2 NLU Platform

As discussed in Section 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 [18]. Among several NLUs, we select Rasa platform v2.2.3 [48] for several reasons. Rasa is an open-source NLU and can be run on a local machine. Therefore, the Rasa implementation stays the same during our experiment. Moreover, Rasa has been commonly used by prior work to develop SE chatbots [19, 31] 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 in to be labeled. However, training

[3]https://github.com/marketplace/askgit-io

Table 1. A snapshot of ten intents with their definitions from the AskGit dataset.

Intent	Definition	Total	Avg.	Median	Min	Max
ProjectCollaborators	Presents the developer(s) who contributed to the project.	30	6.6	5.5	4	11
StarCount	Presents the number of stars for the repository.	29	6.7	6	2	12
IssueRelatedCommits	Determines the commits linked to a certain bug.	22	7.1	7	4	13
FileCreator	Identifies the developer(s) who creates a specific file in the repository.	21	6.3	6	3	11
IssueContributors	Determines the developer(s) who contributes in fixing a specific bug.	17	6.5	7	3	11
ModifiedFilesPR	Identifies the files that are touched by a specific pull-request.	12	7.4	7	4	13
ActivityReport	Presents statistics about the repository activities occurred in the last day (e.g., number of solved bugs).	10	5.7	6.5	2	9
LongestOpenPR	Identifies the longest pull-request which is still opened.	9	8	8	3	11
CommitsCountInBranch	Determines the number of commits in a certain branch.	7	7.1	8	4	8
OverloadedDev	Identifies overloaded developer(s) with the highest number of unfixed bugs.	6	8.3	8.5	5	11

Table 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

NLUs is expensive in terms of time and resources, especially when the training dataset is large. Rasa overcomes the aforementioned issue through incremental learning [67]. Incremental learning is a method to train the models on the new data and persist the knowledge gained from the original data [43]. Consequently, incremental learning saves time and computational resources in the case of training the NLU using large training sets [21]. Using Rasa enables 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 used in our approach for the assessment. This allows the replication of the approach on other datasets. In the data preprocessing component, we leverage spaCy [61] for text preprocessing and POS tagging in the query. SpaCy is a Python library for NLP tasks. We use Contractions Python library[4] 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 [46], 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 [1].

To develop LFs, the first two authors (domain experts) used their chatbot development expertise to develop the LFs for all intents in the AskGit dataset (i.e., 52 intents). In particular, each domain expert independently examined three random queries for every intent in the dataset and the extracted information (e.g., entity type) by the query information extractor component. Using the examined queries and the extracted information, they develop heuristics for each intent. For example, while examining the *RepositoryLicense* intent, the domain expert finds the intent has no entity associated with it, and all the examined queries contain the word ‘License’ in it. After each domain expert developed possible heuristics for each intent in the dataset, they discussed them to ensure that they are relevant to the intent and do not overlap with heuristics of other intents. Finally, they implement the heuristics as LFs. The annotators took less than two hours to devise all the LFs, which is significantly less time-consuming compared to labeling each query in our dataset under various experimental settings used in the evaluation (e.g., different dataset splits). Once the LFs for an intent are created, they can be applied to new data as needed. It is important to emphasize that we exclude the examined queries used to develop the LFs from our evaluation dataset.

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 5. The first LF in Figure 5a checks the query against three conditions (e.g., the query starts with ‘Who’) to be classified as *OverloadedDev* intent by this LF. Figure 5b 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 query’s final label (intent) since most of the intents (65%) have only one LF.

Although our LFs can be applied to a software repository-based chatbot (AskGit), it is important to note that this is the only manual step that needs to be done by AlphaBot’s users, which is less time-consuming and resource-intensive than manually labeling each data point in the log. In other words, the domain expert creates LFs once and applies them to each query that the chatbot receives, instead of manually labeling the queries on a daily basis.

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., [1, 14]) - 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 (i.e., Recall = $\frac{TP}{TP+FN}$). The precision is the percentage of the correctly classified queries to the total number of classified queries for the intent (i.e., Precision = $\frac{TP}{TP+FP}$). Finally, we combine both precision and recall using the weighted F1-score that has been used in similar work [1]. More specifically, we compute the F1-score for each intent and aggregate all intents F1-score (i.e., F1-score = $2 \times \frac{Precision \times Recall}{Precision + Recall}$) using the weighted average. We consider the classes’ support as weights to

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

```python
# 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.

```python
# 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.

Fig. 5. The two LFs for classifying queries as OverloadedDev intent.

compute the weighted F1-score. Although we evaluate all three measures, we only present the weighted F1-score in the paper. The precision and recall are available on [6].

5 RESULTS

In this section, we present the results of our case study. For each research question, we detail the motivation for the question, the approach that we use to answer the question, and the results.

5.1 RQ1: Does AlphaBot improve the NLU's performance?

Motivation: Since user's queries are the primary source for the chatbot developers to train NLUs [35, 47], 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 [56]. Moreover, automating the labeling process reduces the chatbot development costs (i.e., the cost of annotators to label all the queries from users) and allows developers to focus on critical tasks in their chatbot implementation rather than data annotation. Therefore, the main goal of this research question is to determine the impact of using AlphaBot on the NLU's performance.

Approach: To achieve this, we follow the approach used by prior work [26, 53], and evaluate Rasa's performance before and after applying AlphaBot using the AskGit dataset (discussed in Section 4.1). Specifically, we randomly split the dataset into three blocks: training, validation, and test sets. We use the stratified split method to maintain a consistent distribution of intents across all splits (i.e., training, validation, and test sets). We train Rasa on the training set only, without augmenting any queries. This serves as the baseline in our study. To simulate a realistic situation where users pose questions to the chatbot and the domain experts label the queries to retrain the NLU on them, we consider the queries in the validation split as users' inputs to the chatbot, which need to be annotated by the domain expert. Therefore, we apply AlphaBot to all queries in the validation set.

Table 3. 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	263/267 (98.5)
20	40	40	235/237 (99.2)
30	35	35	204/208 (98.1)
40	30	30	176/178 (98.8)
50	25	25	147/148 (99.3)
60	20	20	118/119 (99.2)
70	15	15	86/88 (97.7)
80	10	10	59/59 (100)
90	5	5	8/8 (100)

Each query in the validation set is provided as input to AlphaBot to predict its intent. Subsequently, we merge the output of AlphaBot (i.e., labeled queries) with the training set to train Rasa; we refer to this model as AlphaBot_Rasa. Finally, we use the test set to evaluate the performance of both the baseline and AlphaBot_Rasa.

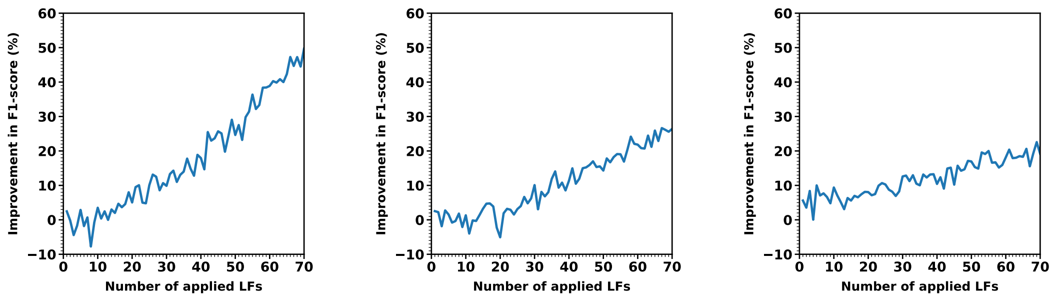
To assess the AlphaBot effectiveness when it is applied at different stages of chatbot lifetime (i.e., from early releases to maturity), we evaluate the AlphaBot on different sizes of training, validation, and test sets. In particular, we split the AskGit dataset incrementally 10% each time, where this split is used to train Rasa, and the rest are used as the validation and test sets. For example, we divide the AskGit dataset into 10% for training and 90% for test and validation sets (45% each). Table 4 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.

Results: Table 3 presents the number of correctly classified queries to the total number of queries in that split. From the table, we find that AlphaBot classifies the intents for most of the queries correctly across all splits. AlphaBot mislabels, in total, 13 unique 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 the word 'default') in the query. This implies that practitioners need to consider typos when developing their LFs. In addition to correctly labeling the queries, AlphaBot took less than a minute to label the users' queries for each split. This shows the amount of time saved compared to manually labeling user's queries as discussed in Section 2.

Next, we assess the impact of applying AlphaBot on the NLU's performance. Table 4 presents the performance of baseline and AlphaBot_Rasa and 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 4. Also, we notice that as the size of the training split increases, the percentage of improvement decreases. More specifically, the peak in the performance improvement (35.57%) occurs in the early releases of a chatbot with the lowest training set size (split 10%). Interestingly, we find that applying AlphaBot at 20% training split (AlphaBot_Rasa)

Table 4. 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	33.87	69.44	35.57
20	40	40	47.80	76.49	28.69
30	35	35	52.34	78.85	26.52
40	30	30	61.74	78.76	17.02
50	25	25	70.41	80.98	10.57
60	20	20	76.07	83.91	7.84
70	15	15	80.53	86.57	6.03
80	10	10	82.22	86.50	4.28
90	5	5	76.92	77.88	0.96



(a) 10% Training Split (b) 30% Training Split (c) 50% Training Split

Fig. 6. Analysis of performance trend in terms of F1-score when applying LFs in an additive manner.

achieves similar performance to the baseline at 50%. This is because, at 20%, AlphaBot correctly labels 99.2% of queries in the validation set. Another observation is that using AlphaBot leads to less performance improvement in the F1-score compared to the baseline when the chatbot becomes more mature (i.e., trained on a high number of queries), where there is a 0.96% improvement in terms of F1-score at split 90%. This is expected as the NLUs tend to have better performance if trained on more queries [1]. Our findings show an advantage of using AlphaBot in improving the NLU's performance, especially when it has little training data.

AlphaBot improves the NLU's performance, especially when it is applied at the early stages of chatbot lifetime (35.57% improvement in F1-Score), and the training data is limited. However, the improvement in the performance decreases as the chatbot becomes more mature (i.e., trained on a large data set).

5.2 RQ2: What is the impact of the number of labeling functions on performance?

Motivation: Given that using AlphaBot improves the NLU's performance, as shown in RQ1. There's no such thing as a free lunch; crafting the LFs to automate the labeling process requires 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.

Approach: 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 in the early releases of chatbot. For each split, we report the average of all runs of the three LFs lists.

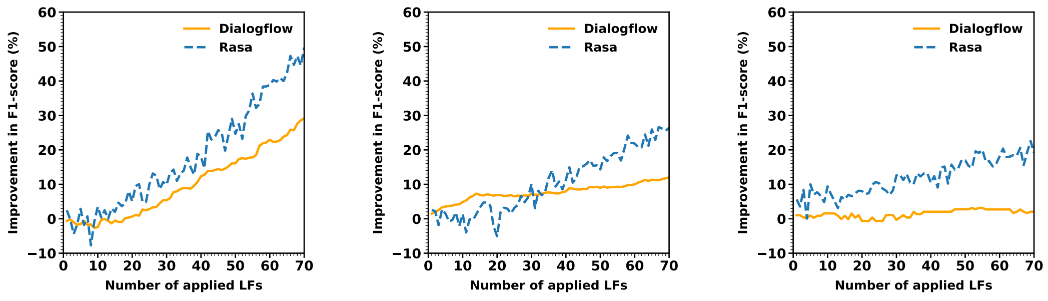
Results: Figure 6 presents the performance in terms of F1-score when applying LFs additively on 10%, 30%, and 50% splits compared to the baseline. From the figure, we notice that adding more LFs yields a performance increase. This happens because Rasa is trained on more correctly labeled queries. In 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 6b, the performance decreases when applying the 42st LF that adds four correct training queries compared to the previous 41 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 performance.

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. Our findings highlight that it is worth developing more LFs at the early stages of the chatbot lifetime as there is a substantial increase in the NLU's performance as more LFs are applied compared to when the chatbot becomes more mature, i.e. trained on more data. This is expected as Rasa becomes more robust when it 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).

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.

Table 5. 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	54.84	83.31	28.47
30	35	35	73.56	84.65	11.09
50	25	25	81.46	84.79	3.33



(a) 10% Training Split (b) 30% Training Split (c) 50% Training Split

Fig. 7. Results of applying the LFs additively on Dialogflow's F1-score.

6 DISCUSSION

6.1 Google Dialogflow

We use the Rasa platform to evaluate AlphaBot as discussed in Section 3. In this section, we want to examine whether our approach could improve other NLUs' performance. Therefore, we rerun our experiment using Dialogflow platform [18]. Dialogflow is a cloud-based NLU platform developed by Google. Dialogflow supports more than 30 languages and can be integrated with popular communication channels [18]. 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 SE chatbots [2, 41, 44].

To evaluate the impact of AlphaBot on Dialogflow's performance, we use the same experiment settings, dataset, and splits discussed in RQ1 and RQ2. In particular, we assess Dialogflow's performance when using AlphaBot (RQ1) and evaluate the impact of adding more LFs on the performance trend (RQ2). It is worth noting that we assess 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 boost the chatbot's performance at early releases. Also, allow developers to focus on the core functionalities of the chatbot rather than annotating users' queries.

Table 5 presents F1-scores for Baseline (Dialogflow before applying AlphaBot), AlphaBot_Dialogflow, and the percentage of improvement after applying AlphaBot. Similar to RQ1 results, AlphaBot increases the performance of Dialogflow in terms of F1-score for all splits. In particular, the peak performance increase is at splits with fewer training queries (i.e., 10% training split) and the improvement decreases as Dialogflow is trained on more queries (50% training split) as it becomes more robust. Figure 7 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

Table 6. The results of the ablation study on the number of queries labeled correctly.

Split			w/o Data Preprocessing		w/o Query Information Extractor		AlphaBot	
Training	Validation	Testing	Correctly labeled / total no. of queries (%)	Abandoned (%)	Correctly labeled / total no. of queries (%)	Abandoned (%)	Correctly labeled / total no. of queries (%)	Abandoned (%)
10	**45**	**45**	1/39 (0.02)	88.4	125/128 (97.7)	62.0	263/267 (98.5)	20.8
20	**40**	**40**	2/35 (0.06)	88.3	102/108 (94.4)	64	235/237 (99.2)	21
30	**35**	**35**	0/35 (0)	86.6	102/104 (98.1)	60.3	204/208 (98.1)	20.6
40	**30**	**30**	1/29 (0.03)	87.1	79/81 (97.5)	64	176/178 (98.8)	20.9
50	**25**	**25**	0/20 (0)	89.3	70/75 (93.3)	59.9	147/148 (99.3)	20.9
60	**20**	**20**	0/18 (0)	88	45/48 (93.6)	68	118/119 (99.2)	20.7
70	**15**	**15**	1/29 (0.03)	74.1	39/40 (97.5)	64.3	86/88 (97.7)	21.4
80	**10**	**10**	0/8 (0)	89.3	32/32 (100)	57.3	59/59 (100)	21.3
90	**5**	**5**	0/7 (0)	69.6	14/14 (100)	39.1	8/8 (100)	65.2

training queries. We have the same observations for Rasa as discussed in RQ2. Our findings show that our approach could be generalized to other NLU platforms.

6.2 Coldstart Scenario

In our evaluation of AlphaBot, the first two authors developed the LFs by manually examining three random queries for every intent in the dataset and the extracted information (e.g., entity type) by the query information extractor component to devise the LFs, as discussed in Section 4.3. However, the developers might have a list of intents for their chatbot, but they have not obtained any users' queries yet (i.e., a cold start). This scenario is where developers are about to start building their chatbot. This raises a question: Can practitioners still use our approach in the cold start scenario? The short answer is yes. AlphaBot's users can leverage their domain knowledge and expertise to develop LFs. For example, in the 'StarCount' intent, a practitioner might create an LF that checks if the query contains terms such as ('star' or 'popularity') and ('count' or 'how many'), anticipating these elements in all queries related to this intent. However, the accuracy and coverage of the developed LFs in a cold start scenario might be lower compared to developing LFs with access to users' queries posed to the chatbot. This is because users may ask questions that do not align with the heuristics specified in the LFs, resulting in the LFs abstaining from labeling such queries. For example, the query "What is the reputation of the repository?" does not match the heuristics defined in the LFs (e.g., 'star' or 'popularity'), leading the LF to return 'ABSTAIN'. Therefore, we recommend AlphaBot's users to add more LFs as their chatbot evolves by examining the abandoned queries file generated by AlphaBot, as discussed in Section 3.3. Overall, AlphaBot can save time and resources during a cold start scenario by minimizing the need for annotating users' queries. In other words, users only need to review abandoned queries (a portion of the dataset) to develop additional LFs.

6.3 Ablation Study

To evaluate the impact of the Data Preprocessing component and the Query Information Extractor component on AlphaBot's performance, we conducted an ablation study. Specifically, we reran AlphaBot by initially removing the Data Preprocessing component (w/o Data Preprocessing) and keeping the Query Information Extractor component to label the users' queries in the validation set. Subsequently, we ran the same experiment again by excluding the Query Information Extractor component (w/o Query Information Extractor) and keeping the Data Preprocessing component. It is important to note that we performed the experiment across different splits (e.g., 10% training, 45% validation, and 45% testing) used in RQ1. Furthermore, we did not exclude the Intent Labeler component as it is the core component that performs the labeling to the user's query. Table 6 presents the correctly labeled queries and abandoned queries (abstain) when including all the components (i.e., Data Preprocessing, Query Information Extractor, and Intent Labeler), excluding the Data Preprocessing component (w/o Data Preprocessing), and excluding the Query Information Extractor component (w/o Query Information Extractor). From the table, we observe that Data Preprocessing plays a major role in labeling queries. In other words, AlphaBot's performance is highly impacted by the Data Preprocessing component. Upon closer examination of the results, we find that chatbot users enters the same word in multiple ways, making them different from the heuristics used to develop LFs. For example, the "number_of_commits_in_branch" LF fails to label the query "how many commits are in the documentation branch" because the heuristic used in the LF is 'commit' and the query contains 'commits'. On the other hand, removing the Query Information Extractor Component from AlphaBot has a slight to negligible impact on the accuracy of labeling the user's queries compared to including all components (i.e., AlphaBot). However, when running AlphaBot without the Query Information Extractor component (w/o Query Information Extractor), its coverage in terms of abandoned queries is reduced. For example, in the Training split 10%, although AlphaBot and w/o Query Information Extractor achieve similar accuracy in labeling queries, the w/o Query Information Extractor labeled 38% of queries, while AlphaBot labeled 80% of queries. The Query Information Extractor component extracts information from queries such as Entities, which helps to identify the intent [1]. Our results show that using all components in AlphaBot achieves both higher accuracy and coverage.

7 THREATS TO VALIDITY

Internal Validity: Concerns confounding factors that could have influenced our results. We develop LFs to label queries using the collected features discussed in Section 3, which might introduce human bias in crafting those LFs. To mitigate this bias, the first two authors 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 the best results. In our study, we developed LFs using keyword searching and pattern matching. Developing LF using other weak supervision forms (e.g., distant supervision) might change our results. Nevertheless, our developed LFs correctly labeled more than 99% of queries as shown in RQ1. Moreover, we plan to evaluate the performance of other weak supervision forms in labeling queries.We use regular expressions in the Query Information Extractor component to extract entities from the users' queries. Using Named Entity Recognition (NER) models might lead to different results. However, upon closer examination of the results, the Query Information Extractor correctly extracted 97.9% of entities from the user's queries which makes us confident in the use of the regular expressions to extract entities.

In this work, we focus on single-intent queries, as the majority of publicly available datasets for SE chatbots predominantly consist of single intents [1, 2, 14, 19, 31]. Furthermore, as this study represents the first attempt to propose a weak supervision-based approach to automate the query annotation process for SE chatbots, concentrating on single-intent queries provides a controlled environment for the assessment and improvement of AlphaBot's capabilities in labeling users' queries. We plan in the future to evaluate our approach on datasets that contain multi-intent queries.

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 in Section 5. 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.

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. Furthermore, the developed LFs might not be generalizable to all queries that are semantically similar but syntactically different. To the best of our knowledge, this is the only publicly available comprehensive dataset that contains enough queries (749 queries) and intents (52 intents) for an SE chatbot, which enables us to evaluate AlphaBot's performance in a large dataset. Moreover, it enables us to evaluate the AlphaBot's applicability by applying it to a chatbot in production. Since AlphaBot achieves promising results, we plan 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 and Dialogflow platforms. This might affect the generalizability of our results. However, Rasa and Dialogflow are commonly used by chatbot developers and researchers to develop SE chatbots [2, 19, 31, 41, 44]. 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.

8 RELATED WORK

This paper 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.

8.1 Chatbots

Several studies developed chatbots in different domains such as healthcare [12], financial [39], and education [16]. Recently, several studies proposed chatbots to assist software practitioners in their daily development tasks [2, 13, 19, 31, 54, 57, 72]. Abdellatif et al. [2] developed MSRBot using Google Dialogflow to answer questions related to the software project (e.g., "Which commit fixes bug-5281?"). Lin et al. [31] leveraged Rasa to implement MSABot, which helps practitioners maintain microservices. Dominic et al. [19] developed a chatbot using Rasa to assist newcomers in the onboarding process to new projects. Toxtli et al. [65] developed TaskBot using Microsoft Language Understanding Intelligent Service (LUIS) to help software practitioners manage their tasks (e.g., task reminders).

The increased attention to SE chatbots and the challenges of collecting data to train chatbots [2, 19] motivates our study; to help practitioners enhance the chatbot performance in intents classification and save resources by automating the annotation process of user's input to the chatbot. That said, our study differs in that we aim to support chatbot practitioners, and we do not develop chatbots.

8.2 Weak Supervision

Recently, many researchers use weak supervision for text classification [34, 45, 55, 59, 60]. For example, Shu et al. [60] used weak supervision to identify the intent of emails. They use the weak supervision to enhance intent detection in emails. Rao et al. [45] 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 (more than 76%) of the models. There are a number of studies that use weak supervision to improve the NLU's performance in chatbots [24, 40]. Hancock et al. [24] proposed a self-feeding chit-chat 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, despite the amount of traditional supervision. Mallinar et al. [32] presented a framework that enables annotators to search for user's queries in the customer service chatlog 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 trained on six intents. The results show that the proposed approach improves the chatbot's performance. Oramas et al. [40] developed a weak supervision-based approach to label the entities in the voice transcribed queries in the music domain. They found that their approach 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, the proposed approaches are evaluated using a limited number of intents. To the best of our knowledge, AlphaBot is the first to use weak supervision to boost NLU's performance for SE chatbots. Also, our performance evaluation covers open-source (Rasa) and close-source (Dialogflow) NLUs. We examine the impact of the number of labeling functions on the NLU's performance, which have not been previously studied by prior work. We believe 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 (e.g., entity and POS) that contribute to identifying the intent of a query in the SE domain.

9 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 the proposed approach using a dataset composed 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 (0.96% - 35%) F1-score compared to the baseline. Furthermore, 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.

In the future, we plan in the future to evaluate AlphaBot using more NLUs and SE datasets (e.g., code related questions). Our approach achieves high performance in labeling queries, as discussed in RQ1 and RQ2. The knowledge and expertise required to develop LFs is the bottleneck of its use. Developing high-quality LFs is a resource-intensive task. Therefore, we intend (and encourage others) to investigate the impact of using different types of weak supervision forms (e.g., using pre-trained models) on the NLU's performance to save time in developing LFs. Furthermore, we plan to propose an approach that creates LFs based on the input data and can be integrated with AlphaBot. In addition to reduces the cost and time of creating LFs, it enables chatbot developers to focus on the critical tasks of their chatbot implementation rather than annotating data.

10 DATA AVAILABILITY

We make the source code of the approach, datasets, and results publicly available [6] to facilitate replication and accelerate future research in the area.

REFERENCES

[1] Ahmad Abdellatif, Khaled Badran, Diego Costa, and Emad Shihab. 2021. A Comparison of Natural Language Understanding Platforms for Chatbots in Software Engineering. *IEEE Transactions on Software Engineering (TSE)* (2021), 1–1.

[2] Ahmad Abdellatif, Khaled Badran, and Emad Shihab. 2020. MSRBot: Using Bots to Answer Questions from Software Repositories. *Empirical Software Engineering (EMSE)* 25 (2020), 1834–1863. Issue 3.

[3] Ahmad Abdellatif, Khaled Badran, and Emad Shihab. 2021. AskGit. https://askgit.io/. (Accessed on 12/07/2023).

[4] Ahmad Abdellatif, Diego Elias Costa, Khaled Badran, Rabe Abdelkareem, and Emad Shihab. 2020. Challenges in Chatbot Development: A Study of Stack Overflow Posts. In *Proceedings of the 17th International Conference on Mining Software Repositories (MSR'20)*. To Appear.

[5] Enrique Alfonseca, Katja Filippova, Jean-Yves Delort, and Guillermo Garrido. 2012. Pattern Learning for Relation Extraction with a Hierarchical Topic Model. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2* (Jeju Island, Korea) *(ACL '12)*. Association for Computational Linguistics, USA, 54–59.

[6] Anonymous. 2023. A Weak Supervision-based Approach to Improve Chatbots for Code Repositories. https://zenodo.org/records/10714394. (Accessed on 28/09/2023).

[7] Chidubem Arachie and Bert Huang. 2021. A General Framework for Adversarial Label Learning. *Journal of Machine Learning Research* 22, 118 (2021), 1–33.

[8] Julie Ask, Michael Facemire, and Andrew Hogan. 2016. The State Of Chatbots. *Forrester.com report* 20 (2016).

[9] Stephen H Bach, Bryan He, Alexander Ratner, and Christopher Ré. 2017. Learning the structure of generative models without labeled data. In *International Conference on Machine Learning*. PMLR, 273–282.

[10] Stephen H. Bach, Daniel Rodriguez, Yintao Liu, Chong Luo, Haidong Shao, Cassandra Xia, Souvik Sen, Alex Ratner, Braden Hancock, Houman Alborzi, Rahul Kuchhal, Chris Ré, and Rob Malkin. 2019. Snorkel DryBell: A Case Study in Deploying Weak Supervision at Industrial Scale. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 362–375.

[11] Chetashri Bhadane, Hardi Dalal, and Heenal Doshi. 2015. Sentiment Analysis: Measuring Opinions. *Procedia Computer Science* 45 (2015), 808–814. International Conference on Advanced Computing Technologies and Applications (ICACTA).

[12] Urmil Bharti, Deepali Bajaj, Hunar Batra, Shreya Lalit, Shweta Lalit, and Aayushi Gangwani. 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)*. 870–875.

[13] Nick C. Bradley, Thomas Fritz, and Reid Holmes. 2018. Context-Aware Conversational Developer Assistants. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 993–1003.

[14] Daniel Braun, Adrian Hernandez Mendez, Florian Matthes, and Manfred Langen. 2017. Evaluating Natural Language Understanding Services for Conversational Question Answering Systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*. Association for Computational Linguistics, Saarbrücken, Germany, 174–185.

[15] Metz C. 2016. 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 08/02/2023).

[16] Fabio Clarizia, Francesco Colace, Marco Lombardi, Francesco Pascale, and Domenico Santaniello. 2018. Chatbot: An Education Support System for Student. In *Cyberspace Safety and Security*, Arcangelo Castiglione, Florin Pop, Massimo Ficco, and Francesco Palmieri (Eds.). Springer International Publishing, Cham, 291–302.

[17] Allan Peter Davis, Thomas C Wiegers, Phoebe M Roberts, Benjamin L King, Jean M Lay, Kelley Lennon-Hopkins, Daniela Sciaky, Robin Johnson, Heather Keating, Nigel Greene, et al. 2013. A CTD–Pfizer collaboration: manual curation of 88 000 scientific articles text mined for drug–disease and drug–phenotype interactions. *Database* 2013 (2013).

[18] Dialogflow. 2021. Dialogflow Official Website. https://dialogflow.cloud.google.com/. (Accessed on 01/08/2023).

[19] James Dominic, Jada Houser, Igor Steinmacher, Charles Ritter, and Paige Rodeghero. 2020. Conversational Bot for Newcomers Onboarding to Open Source Projects. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (Seoul, Republic of Korea) *(ICSEW'20)*. Association for Computing Machinery, New York, NY, USA, 46–50.

[20] Jason A. Fries, Ethan Steinberg, Saelig Khattar, Scott L. Fleming, Jose Posada, Alison Callahan, and Nigam H. Shah. 2021. Ontology-driven weak supervision for clinical entity classification in electronic health records. *Nature Communications* 12, 1 (2021), 2017.

[21] Xin Geng and Kate Smith-Miles. 2009. *Incremental Learning*. Springer US, Boston, MA, 731–735.

[22] Sebastian Romy Gomes, Sk Golam Saroar, Md Mosfaiul, Alam Telot, Behroz Newaz Khan, Amitabha Chakrabarty, and Moin Mostakim. 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)*. 482–487.

[23] GoodRebels. [n. d.]. The impact of conversational bots in the customer experience. https://www.goodrebels.com/the-impact-of-conversational-bots-in-the--experience/. (Accessed on 12/07/2023).

[24] Braden Hancock, Antoine Bordes, Pierre-Emmanuel Mazare, and Jason Weston. 2019. Learning from dialogue after deployment: Feed yourself, chatbot! *arXiv preprint arXiv:1901.05415* (2019).

[25] Howe and Jeff. 2006. The Rise of Crowdsourcing. *Wired* 14 (01 2006).

[26] Mia Mohammad Imran, Yashasvi Jain, Preetha Chatterjee, and Kostadin Damevski. 2023. Data Augmentation for Improving Emotion Recognition in Software Engineering Communication. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (Rochester, MI, USA) *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 29, 13 pages. https://doi.org/10.1145/3551349.3556925

[27] Zhao Jianqiang and Gui Xiaolin. 2017. Comparison Research on Text Pre-processing Methods on Twitter Sentiment Analysis. *IEEE Access* 5 (2017), 2870–2879.

[28] jsphdnl. 2017. nlp - Conversational Data for building a chat bot - Stack Overflow. https://stackoverflow.com/questions/45821517/conversational-data-for-building-a-chat-bot. (Accessed on 08/14/2023).

[29] Pensive Knave. 2021. 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 08/25/2023).

[30] Carlene Lebeuf, Margaret-Anne Storey, and Alexey Zagalsky. 2018. Software Bots. *IEEE Software* 35, 1 (2018), 18–23.

[31] Chun-Ting Lin, Shang-Pin Ma, and Yu-Wen Huang. 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* (Seoul, Republic of Korea) *(ICSEW'20)*. Association for Computing Machinery, New York, NY, USA, 36–40.

[32] Neil Mallinar, Abhishek Shah, Rajendra Ugrani, Ayush Gupta, Manikandan Gurusankar, Tin Kam Ho, Q Vera Liao, Yunfeng Zhang, Rachel KE Bellamy, Robert Yates, et al. 2019. Bootstrapping conversational agents with weak supervision. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 9528–9533.

[33] Gideon S. Mann and Andrew McCallum. 2010. Generalized Expectation Criteria for Semi-Supervised Learning with Weakly Labeled Data. *Journal of Machine Learning Research* 11, 32 (2010), 955–984.

[34] Yu Meng, Jiaming Shen, Chao Zhang, and Jiawei Han. 2018. Weakly-Supervised Neural Text Classification. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (Torino, Italy) *(CIKM '18)*. Association for Computing Machinery, New York, NY, USA, 983–992.

[35] Microsoft. 2021. 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 08/12/2023).

[36] Microsoft. 2021. LUIS (Language Understanding) - Cognitive Services. https://www.luis.ai/. (Accessed on 12/09/2023).

[37] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant Supervision for Relation Extraction without Labeled Data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2* (Suntec, Singapore) *(ACL '09)*. Association for Computational Linguistics, USA, 1003–1011.

[38] Eadicicco L. Baidu's Andrew Ng. 2017. on the future of artificial intelligence. https://time.com/4631730/andrew-ng-artificial-intelligence-2017/. (Accessed on 07/24/2023).

[39] T. Okuda and S. Shoda. 2018. AI-based chatbot service for financial industry. *Fujitsu Scientific and Technical Journal* 54 (04 2018), 4–8.

[40] Sergio Oramas, Massimo Quadrana, and Fabien Gouyon. 2021. 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*. Association for Computational Linguistics, Online, 49–55.

[41] Elahe Paikari, JaeEun Choi, SeonKyu Kim, Sooyoung Baek, MyeongSoo Kim, SeungEon Lee, ChaeYeon Han, YoungJae Kim, KaHye Ahn, Chan Cheong, and André van der hoek. 2019. A Chatbot for Conflict Detection and Resolution. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. 29–33.

[42] Jerrod Parker and Shi Yu. 2021. Named Entity Recognition through Deep Representation Learning and Weak Supervision. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Association for Computational Linguistics, Online, 3828–3839.

[43] R. Polikar, L. Upda, S.S. Upda, and V. Honavar. 2001. Learn++: an incremental learning algorithm for supervised neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 31, 4 (2001), 497–508.

[44] Ilham A. Qasse, Shailesh Mishra, and Mohammad Hamdaqa. 2021. iContractBot: A Chatbot for Smart Contracts' Specification and Code Generation. In *Proceedings of the IEEE/ACM 43th International Conference on Software Engineering Workshops*. Association for Computing Machinery, New York, NY, USA.

[45] Nikitha Rao, Chetan Bansal, and Joe Guan. 2021. Search4Code: Code Search Intent Classification Using Weak Supervision. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. 575–579.

[46] Rasa. [n. d.]. Duckling. https://duckling.wit.ai/. (Accessed on 08/14/2023).

[47] Rasa. 2021. Introduction to Rasa X. https://rasa.com/docs/rasa-x/. (Accessed on 07/29/2023).

[48] RASA. 2021. Open source conversational AI | Rasa. https://rasa.com/. (Accessed on 12/09/2023).

[49] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, Vol. 11. NIH Public Access, 269.

[50] Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid Training Data Creation with Weak Supervision. *Proc. VLDB Endow.* 11, 3 (Nov. 2017), 269–282.

[51] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2020. Snorkel: Rapid training data creation with weak supervision. *The VLDB Journal* 29, 2 (2020), 709–730.

[52] Daniele Ravì, Charence Wong, Fani Deligianni, Melissa Berthelot, Javier Andreu-Perez, Benny Lo, and Guang-Zhong Yang. 2017. Deep Learning for Health Informatics. *IEEE Journal of Biomedical and Health Informatics* 21, 1 (2017), 4–21.

[53] Georgios Rizos, Konstantin Hemker, and Björn Schuller. 2019. Augment to Prevent: Short-Text Data Augmentation in Deep Learning for Hate-Speech Classification. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Beijing, China) *(CIKM '19)*. Association for Computing Machinery, New York, NY, USA, 991–1000. https://doi.org/10.1145/3357384.3358040

[54] Ricardo Romero, Esteban Parra, and Sonia Haiduc. 2020. Experiences Building an Answer Bot for Gitter. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (Seoul, Republic of Korea) *(ICSEW'20)*. Association for Computing Machinery, New York, NY, USA, 66–70.

[55] Massimo Ruffolo. and Francesco Visalli. 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,*. INSTICC, SciTePress, 249–256.

[56] V. Selvi, S. Saranya, K. Chidida, and R. Abarna. 2019. Chatbot and bullyfree Chat. In *2019 IEEE International Conference on System, Computation, Automation and Networking (ICSCAN)*. 1–5.

[57] Dragos Şerban, Bart Golsteijn, Ralph Holdorp, and Alexander Serebrenik. 2021. SAW-BOT: Proposing Fixes for Static Analysis Warnings with GitHub Suggestions. In *Workshop on Bots in Software Engineering*. IEEE Computer Society, United States.

[58] Sheri. 2020. python - Intent classification for Chatbot - Stack Overflow. https://stackoverflow.com/questions/62970861/intent-classification-for-chatbot. (Accessed on 09/05/2023).

[59] Xiaoming Shi, Haifeng Hu, Wanxiang Che, Zhongqian Sun, Ting Liu, and Junzhou Huang. 2020. Understanding Medical Conversations with Scattered Keyword Attention and Weak Supervision from Responses. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 05 (Apr. 2020), 8838–8845.

[60] Kai Shu, Subhabrata Mukherjee, Guoqing Zheng, Ahmed Hassan Awadallah, Milad Shokouhi, and Susan Dumais. 2020. *Learning with Weak Supervision for Email Intent Detection.* Association for Computing Machinery, New York, NY, USA, 1051–1060.

[61] Spacy. 2021. Industrial-strength Natural Language Processing in Python. https://spacy.io/. (Accessed on 01/09/2023).

[62] Margaret-Anne Storey and Alexey Zagalsky. 2016. Disrupting Developer Productivity One Bot at a Time. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Seattle, WA, USA) *(FSE 2016)*. Association for Computing Machinery, New York, NY, USA, 928–931.

[63] Snorkel Team. 2020. An Overview of Weak Supervision. https://www.snorkel.org/blog/weak-supervision. (Accessed on 07/17/2023).

[64] Snorkel Team. 2020. Snorkel Intro Tutorial: Data Labeling. https://www.snorkel.org/use-cases/01-spam-tutorial. (Accessed on 07/09/2023).

[65] Carlos Toxtli, Andrés Monroy-Hernández, and Justin Cranshaw. 2018. Understanding Chatbot-Mediated Task Management. Association for Computing Machinery, New York, NY, USA.

[66] Kieu Tran. 2020. nlp - Building a chatbot about literary novel - Stack Overflow. https://stackoverflow.com/questions/64007306/building-a-chatbot-about-literary-novel. (Accessed on 07/16/2023).

[67] RASA. 2021. Incremental Training. https://rasa.com/docs/rasa/next/migration-guide#incremental-training. (Accessed on 11/08/2023).

[68] S Vijayarani, Ms J Ilamathi, Ms Nithya, et al. 2015. Preprocessing techniques for text mining-an overview. *International Journal of Computer Science & Communication Networks* 5, 1 (2015), 7–16.

[69] Peilin Yu, Tiffany Ding, and Stephen H Bach. 2021. Learning from Multiple Noisy Partial Labelers. *arXiv preprint arXiv:2106.04530* (2021).

[70] Man-Ching Yuen, Irwin King, and Kwong-Sak Leung. 2011. A Survey of Crowdsourcing Systems. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social*

Computing. 766–773.

[71] Ce Zhang, Christopher Ré, Michael Cafarella, Christopher De Sa, Alex Ratner, Jaeho Shin, Feiran Wang, and Sen Wu. 2017. DeepDive: Declarative Knowledge Base Construction. *Commun. ACM* 60, 5 (apr 2017), 93–102.

[72] N. Zhang, Q. Huang, X. Xia, Y. Zou, D. Lo, and Z. Xing. 2020. Chatbot4QR: Interactive Query Refinement for Technical Question Retrieval. *IEEE Transactions on Software Engineering* (2020), 1–1.

[73] Zhi-Hua Zhou. 2017. A brief introduction to weakly supervised learning. *National Science Review* 5, 1 (08 2017), 44–53.

Received 2023-09-28; accepted 2024-04-16