Identifying Opinion Based Questions in Developer Chat Communication

by

Keerthana Muthu Subash

A Thesis submitted to
the Faculty of Graduate and Postdoctoral Affairs in partial fulfilment of
the requirements for the degree of
Master of Computer Science

Carleton University
Ottawa, Ontario

©2022
Keerthana Muthu Subash
The undersigned recommend to the Faculty of Graduate and Postdoctoral Affairs acceptance of the Thesis

Identifying Opinion Based Questions in Developer Chat Communication

Submitted by Keerthana Muthu Subash
in partial fulfilment of the requirements for the degree of

Master of Computer Science

Dr. Olga Baysal, Thesis Supervisor

Dr. Junfeng Wen, Internal Examiner

Dr. Nafiseh Kahani, External Examiner

Dr. Ahmed El-Roby, Chair of Defence

Dr. Michel Barbeau, Department Chair

Carleton University
2022
Abstract

Today, software developers work on complex and fast-moving projects that often require instant assistance from other domain and subject matter experts. Chat servers such as Discord facilitate live communication and collaboration among developers all over the world. With numerous topics discussed in parallel, mining and analyzing the chat data of these platforms would offer researchers and tool makers opportunities to develop software tools and services such as automated virtual assistants, chatbots, chat summarization techniques, Q&A thesaurus, and more.

Firstly in this thesis, we propose a dataset called DISCO consisting of the one-year public DIScord chat COversations of four software development communities. We have collected the chat data of the channels containing general programming Q&A discussions from the four Discord servers, applied a disentanglement technique to extract conversations from the chat transcripts, and performed a manual validation of conversations on a random sample (500 conversations). Our dataset consists of 28,712 conversations, 1,508,093 messages posted by 323,562 users.

Opinion questions are more abundant in chats than in other developer communications. Mining them can help developers quickly getting insights about a specific development topic (e.g., Is declaring functions inside another function actually help?). Especially, powerful virtual conversation assistants can be developed with chat opinion Q&As to help software engineers save their time to get help for bug reports, pair programming, and more. Chatterjee et al. created an opinion Q&A system, ChatEO, that uses online chat platforms as a resource for collecting developer opinions. Secondly in this thesis, we improve the existing ChatEO’s opinion-asking question identification process by replacing heuristics with Deep Learning (DL) architecture given the popularity of DL models in Natural Language Processing (NLP) tasks. We evaluate different DL models combined with various word embeddings to understand
the impact of each combination on the models’ performance in identifying opinion-based questions.

We use our DISCO dataset as the source in addition to the Slack dataset used in existing ChatEO as both have the same characteristics. A small random subset of the data (2,000 conversations) is manually labelled and validated to train and validate the DL models. The results show a better performance of DL models over heuristics and are validated with a manual qualitative study. As DL performance increases with an increase in training data, we have employed an automatic weak learner, Snorkel to label a larger dataset with 44,820 conversations. As the DL performance results of this larger data is not adequate due to class imbalance, we have used class balancing techniques - SMOTE (39,650 instances per class) and Near-Miss (4,472 instances per class). Near-Miss results are comparable to the DL performance with the 2,000 dataset, while SMOTE achieves the best performance in this study (0.95 recall). Multi-CNN performs the best with different dataset sizes due to the powerful deep convolutional architecture. GloVe-Twitter word embedding offers the best results across as Twitter data resembles chat conversations for its shorter and informal nature.
I dedicate this thesis to my mom and my husband, whom I love very much.
Acknowledgments

I, Keerthana Muthu Subash, would like to express my deepest gratitude to my amazing supervisor, Professor Olga Baysal, for her continuous guidance, advice, and friendly discussions throughout my thesis. I will always be grateful to you for your support and kindness through some tiring times I encountered this year. I am very grateful to Professor Preetha Chatterjee from the University of Drexel for trusting me with extending her work and supporting me in every way possible from day one.

I am truly blessed to have my beautiful mom, who never stops to amaze me with her strength and kindness; it reflects in everything I do till now and here on. I am beyond thankful to my amazing husband, who always believed in me and backed up my dreams even when I didn’t believe in them. Without you both, this thesis would not have been possible.

I am lucky to have found my little support system throughout this journey. My little brother and my grandparents for always having my back. And my wonderful friend Lakshmi and her family for moral support and for making me feel at home in this country.

I am thankful to all my Professors at Carleton University and the University of Ottawa for the knowledge they passed on. I thank everyone whom I worked with in our research group. I thank Lakshmi Prasanna Kumar for helping me with the annotation process. I thank Sri Lakshmi Vadlamani for working with us on our paper.
# Table of Contents

- **Abstract** .......................................................... iii
- **Acknowledgments** ................................................ vi
- **Table of Contents** .................................................. vii
- **List of Tables** .................................................... xi
- **List of Figures** ..................................................... xii
- **Nomenclature** ...................................................... xiv

## 1 Introduction

- 1.1 Discord Chat Mining ................................................. 1
- 1.2 Motivation .......................................................... 3
- 1.3 Research Questions ................................................. 4
- 1.4 Contributions ...................................................... 5
- 1.5 Thesis Organization ............................................... 6

## 2 Background and Related Work

- 2.1 Mining Software Developer Communications ................. 7
2.2 Chat Platforms as Knowledge Source ........................................ 9
  2.2.1 Popular chat platforms ...................................................... 9
  2.2.2 Conversation disentanglement .......................................... 11
  2.2.3 Presence of abundant information in chat platforms .......... 11
  2.2.4 Existing research on chat platforms ................................... 12

2.3 Opinion Mining in Chat Platforms ........................................... 14

2.4 NLP and DL for Chat Mining .................................................. 16
  2.4.1 Word embeddings .......................................................... 18
  2.4.2 DL models in NLP tasks .................................................. 18

2.5 Automated Labelling, Class Balancing & Hyper-parameter Tuning . 23
  2.5.1 Automated labelling with weak supervision ....................... 24
  2.5.2 Class balancing techniques ............................................ 24
  2.5.3 Hyper-parameter tuning ................................................ 26

3 Methodology ................................................................. 27
  3.1 Data Collection and Dataset .............................................. 27
    3.1.1 DISCO dataset ............................................................. 28
      3.1.1.1 Data selection ...................................................... 28
      3.1.1.2 Data collection and preprocessing .............................. 29
      3.1.1.3 Conversation disentanglement .................................. 30
    3.1.2 Slack dataset ............................................................ 32

3.2 Modified ChatEO for Question Identification .......................... 32
  3.2.1 Original ChatEO architecture ......................................... 33
  3.2.2 ChatEO - conversation disentanglement ............................ 34
3.2.3 ChatEO - heuristics based question identification  . . . . . . . . 34
3.2.4 ChatEO - DL based answer extraction  . . . . . . . . . . . . . . . 35
3.2.5 Modifications in ChatEO - DL models for question identification 37
3.2.6 Word embeddings as a feature for DL models  . . . . . . . . . . 37
   3.2.6.1 Word2Vec embedding  . . . . . . . . . . . 38
   3.2.6.2 GloVe embedding  . . . . . . . . . . . . . 40
3.2.7 DL models used in ChatEO  . . . . . . . . . . . . . . . . . . . . . 42
   3.2.7.1 LSTM  . . . . . . . . . . . . . . . . . . . 43
   3.2.7.2 Bi-LSTM  . . . . . . . . . . . . . . . . . . 45
   3.2.7.3 CNN  . . . . . . . . . . . . . . . . . . . . 47
3.3 Snorkel for Weak Supervision Labelling  . . . . . . . . . . . . . . . . 51
   3.3.1 Snorkel Architecture  . . . . . . . . . . . . . . . . . . . . . . . 52
   3.3.2 Class balancing (after Snorkel)  . . . . . . . . . . . . . . . . . . 54
   3.3.2.1 SMOTE  . . . . . . . . . . . . . . . . . . . 55
   3.3.2.2 Near-Miss  . . . . . . . . . . . . . . . . . . 55
3.4 Evaluation Metrics  . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
4 Results  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 58
   4.1 Impact of Deep Learning Models in ChatEO  . . . . . . . . . . . . . . 58
      4.1.1 Discord and Slack datasets  . . . . . . . . . . . . . . . . . . . . 58
      4.1.2 Evaluation dataset  . . . . . . . . . . . . . . . . . . . . . . . . . 60
      4.1.3 ChatEO’s heuristics model as baseline  . . . . . . . . . . . . . 61
      4.1.4 DL in ChatEO for opinion-asking question identification  . . . . 61
      4.1.5 Quality analysis: heuristics vs. DL models  . . . . . . . . . . . . 63
## List of Tables

1. Example of an opinion-asking question and answers in Discord python-general channel. ........................................... 15
2. Dataset of disentangled Discord conversations. ................... 29
3. Dataset of disentangled Slack conversations. ....................... 32
4. Example of a linguistic pattern for opinion-asking question and answers. 35
5. Slack and Discord dataset metrics. ........................................ 59
7. Opinion-asking question identification results: DL models. ...... 62
8. Opinion-asking question identification results: DL + larger dataset. .............................................................. 69
9. Opinion-asking question identification results: DL + NearMiss dataset. ............................................................. 71
10. Opinion-asking question identification results: DL + SMOTE dataset. .............................................................. 72
List of Figures

1. Research roadmap ........................................... 4
2. Discord chat conversation: an example ..................... 10
8. Overview of the data collection and disentanglement process .......... 28
9. Format of conversation data .................................. 31
10. Overview of ChatEO automatic extraction of opinion Q&A from developer chats [1] .................................. 33
15. LSTM - forget gate layer [3] ................................. 44
17. LSTM - cell state update [3] ................................. 45
# Nomenclature

## Abbreviations
This thesis uses some common abbreviations existing in the domain of computer science. The following table lists the abbreviations and their meaning:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRC</td>
<td>Internet Relay Chat</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>DL</td>
<td>Deep Learning</td>
</tr>
<tr>
<td>CL</td>
<td>Conversation Length</td>
</tr>
<tr>
<td>CQA</td>
<td>Community Question Answering</td>
</tr>
<tr>
<td>SE</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>SO</td>
<td>Stack Overflow</td>
</tr>
<tr>
<td>CBOW</td>
<td>Continuous Bag Of Words</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>Bi-RNN</td>
<td>Bidirectional Recurrent Neural Network</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short-Term Memory</td>
</tr>
<tr>
<td>Bi-LSTM</td>
<td>Bidirectional Long Short-Term Memory</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>VAE</td>
<td>Variational Autoencoders</td>
</tr>
<tr>
<td>GAN</td>
<td>Generative Adversarial Networks</td>
</tr>
<tr>
<td>LF</td>
<td>Labeling Functions</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>SMOTE</td>
<td>Synthetic Minority Oversampling</td>
</tr>
<tr>
<td>CSA</td>
<td>Conversational Search Assistants</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In this first chapter, we introduce Discord chat mining in Section 1.1 then uncover the motivation behind our work on opinion-asking question identification using Discord chats in Section 1.2. Motivation is followed by our research questions that we seek to find answers to in Section 1.3, then our contributions are summarized in Section 1.4 and lastly, a thesis structure concludes this chapter in Section 1.5.

1.1 Discord Chat Mining

In recent years, more and more software development communities adopt online chat platforms such as Discord, Slack, IRC, Gitter and Microsoft Teams for more effective collaboration and communication on their projects. These chat platforms serve as a vital resource for getting technical help, sharing knowledge with fellow developers, as well as facilitating real-time conversations among community members. Despite the wide adoption and benefits of chat platforms for software development communities, research offers only a few studies on mining these chat conversations compared to the studies on mining emails and bug reports [15], tutorials [16], and Q&A forums [17–20]. Chatterjee et al. [21,22] have mined and studied Slack chat conversations; their results show that these conversations contain valuable information such as code snippets’ description and APIs, bug debugging techniques, best programming practices, and causes of common errors/exceptions. Esteban et al. [23] have studied Gitter data to help new developers get familiar with software products.
To the best of our knowledge, the software engineering research community have not conducted any study on Discord server data. As a public chat platform with thousands of users all over the world, we believe that mining Discord conversation would provide numerous research opportunities and, in turn, help software development communities. The chat conversations in Discord follow an informal, unstructured and asynchronous format. The conversation length might range from 2 messages to 100s spanning numerous participating users. The conversations are not always continuous and are entwined with each other. For the researchers to mine and use Discord chat data, the conversations need to be subjected to a technique to separate or disentangle them.

As a part of our research, we present a disentangled dataset of chat conversations obtained from Discord servers of four programming language communities such as Python, Go, Clojure, and Racket. We have selected the following general technical help channels: python#python-general, gophers#golang, racket$general, and clojurians#clojure. The source chat transcripts from these channels are exported in JSON format and then converted into XML for the date range of November 2019 to October 2020 (12 months). The exception is the golang channel with the chat data being exported from November 2019 to September 2020 (11 months) due to the fair dealing policy [24]. Under this policy, researchers are allowed to use up to 10% of the copyrighted data.

This dataset was then subjected to the disentanglement process by applying the modified Elsner and Charniak’s algorithm used by Chatterjee et al. [22] for Slack chat mining. The resulting disentangled dataset has each XML node representing a chat message utterance. The message utterance has three tags including anonymized user name, a timestamp, the message text, and a computed conversation ID to identify the conversation. The DISCO (DIScord CONversations) dataset consists of 28,712 conversations, 1,508,093 utterances, and 323,562 participants. We have published this DISCO dataset in the proceedings of the MSR 2022 Data Showcase track to provide a starting point for Discord chat mining for the software engineering community.
1.2 Motivation

In software engineering, opinion mining is focused on API-related opinions and developer emotions from Q&A forums [25–29], developer sentiments from commit logs [30], developer intentions from emails and issue reports [31,32], and detecting software requirements and feature requests from app reviews [33,34]. These studies proved that mining opinions could help in increasing developer productivity, improving code efficiency [27], and building better recommendation systems [28]. Chatterjee et al. [22] showed that the Slack chat platform consists of a large number of opinion-asking questions compared to other developer communication. In literature, opinions was identified in software-related documents using lexical patterns [31] and sentiment analysis techniques [27,28,35]. However, these techniques are not directly applicable to identifying opinion-asking questions in chats for several reasons. Chat communities differ in format, with no formal structure, and informal conversation style. The natural language text in chats could follow different syntactic patterns and contain incomplete sentences [21], which could potentially inhibit the automatic mining of opinions.

To mitigate this, Chatterjee et al. [1] created an opinion-asking question answering system, ChatEO, for Slack chat data to explore the feasibility of building opinion-providing virtual assistants for software engineers. They argued that Conversational Search Assistants (CSAs) are one the virtual assistants that can benefit from opinions [36] that support information seekers who struggle to form good queries for exploratory search. In ChatEO, Chatterjee et al. have used a heuristics-based approach in their ChatEO architecture for identifying opinion-asking questions. Our research focuses on enhancing this heuristics-based approach by replacing it with Deep Learning (DL) models. One reason for this is that rule-based approaches are always expensive with the need for domain experts and they are not extensive for all the data points. With a lot of rules, the model gets complicated and can result in contradictions between the rules. We have used DL models to replace heuristics as they have proven to be effective for different Natural Language Processing (NLP) tasks [37] when used with word embeddings. Our Discord dataset described in Section 1.1 complements the Slack dataset [38] used for ChatEO by Chatterjee et al. for extracting opinion asking questions. This is because both platforms are used by software developers in the same way to discuss software needs. Thus, we have used both
Slack and Discord data in our research to enhance ChatEO’s question identification process by using DL methods.

1.3 Research Questions

The main goal of this thesis is to investigate whether deep learning models can improve the question identification part of the ChatEO’s architecture. Figure 1 illustrates the research road map we followed to answer the research questions described in this section.

- **RQ1**: How do deep learning models perform in comparison with the original heuristics-based question identification technique of ChatEO?

  The opinion-question answering system created by Chatterjee et al. [1] on Slack data, ChatEO uses a heuristic-based question identification process. We are improving their work in two folds. Firstly, we incorporate the new Discord dataset created along with the original Slack data as the source as they have the same characteristics with abundant opinion-asking questions [22, 38]. Secondly, we replace the heuristic-based question identification part with DL models with the combination of different word embeddings to understand how each combination impacts the results. We also study how DL performs in comparison to
the heuristics by including a manual quality analysis.

- **RQ2: What is the impact of dataset size and class balancing on the performance of DL models in detecting opinion-based questions?**
  The dataset used to train the DL models from RQ1 is a manually labelled small dataset. As we understand that DL models perform better with a larger training set [39], we want to explore how to increase the size. We have used an automatic weak learner for labelling, Snorkel [13] for the task as it has proven to perform well for automatic labelling [40]. As the class imbalance grows as any real-life dataset grows in size, we also employ different class balancing techniques and study their effect on identifying opinion-based questions.

### 1.4 Contributions

The major contribution of this research are as follows:

1. Creation of a new dataset on a collaborative effort from Discord chat platform which has over 150 million active monthly users [41] all over the world using a Disentanglement technique.

2. Improvement of ChatEO’s question identification process using hyper-parameter tuned DL models with word embeddings that replaces the original heuristics.

3. Manual labelling of a smaller dataset (Slack + Discord) to be used by the DL models for question identification task of modified ChatEO.

4. Quality analysis to compare the results of the heuristics-based approach with DL one.

5. Creating a larger labelled dataset using Snorkel to utilize the full potential of DL models.

1.5 Thesis Organization

The structure for the rest of this thesis is as follows: Chapter 2 discusses Background and Related Work, while Chapter 3 describes our Methodology. Results are presented in Chapter 4 followed by Discussion in Chapter 5. The thesis concludes with Chapter 6 outlining summary of contributions (Section 1.4) and future work (Section 6.2).
Chapter 2

Background and Related Work

This chapter outlines the necessary background knowledge needed for our work and presents all the related work. As introduced in Section 1, the focus of our research is on mining software developer chat communication for the application of creating an opinion question answering system. Section 2 presents the background knowledge needed to understand the scope of the research along with the related work in the domain. Section 2.1, Section 2.2, Section 2.3, Section 2.4, and Section 2.5 provide information on mining software developer communications, chat platforms as a source of knowledge, opinion mining in chat platforms, NLP & DL for chat mining, and, automatic labelling, class balancing & hyper-parameter tuning, respectively.

2.1 Mining Software Developer Communications

Communication among software developers has changed a lot in recent years due to distributed software development practices, i.e., developers contributing to a project while being in different physical locations across the globe. Documenting key discussions and information from the developer communications has become vital for subsequent development and research. Developer communications such as blog posts, bug reports/issues, documentation (e.g., manuals, tutorials, API docs), code reviews, Q&A forums (e.g., Stack Overflow, specialized online forums), public chats (e.g., Slack, Discord) offer specific insights on code discussions, error debugging, best practices, opinion answering, and may take place within the same space (e.g., GitHub). It is different from other written documents like research papers, and e-books, which
only focus on specific domains or topics.

With the abundance of information available about developer communication, there have been several research studies to facilitate the use of the knowledge shared between developers. Mined information from bug reports and mailing lists was used to re-document source code or for mentor recommendations for software projects [15, 12, 43]. Q&A forums such as Stack Overflow have been researched extensively and mined for various use cases. For IDE recommendations, there have been research on programming errors, exceptions, and linking relevant discussions to source code [17, 19, 20, 44, 46]. Chatterjee et al. studied how novice software engineers direct their efforts and what kinds of information they focus on within a post selected from the results returned in response to a search query on Stack Overflow [18]. For API recommendations, Treude et al. [47] proposed an approach for automatically augmenting API documentation with informative sentences from Stack Overflow. Personal blogs and tutorials provide API recommendations by mining the natural language text in them [48, 49]. Rahman et al. [25] proposed a technique that takes a natural language query as input and uses keyword-API associations mined from Stack Overflow to produce API recommendations as output. Other works for enriching API documentations include augmenting code examples [50, 51], identifying iOS and Android API classes for which developers regularly face usage obstacles [52], and integrating crowdsourced frequently asked questions (FAQs) from the web into API documents [53].

Automatically generating comments for open-source projects has been explored using code snippets and their descriptive text from Stack Overflow [54, 55]. Common error patterns in SQL patterns were identified by Nagy et al. [56]. The re-usability of code snippets from Stack Overflow has been explored by Yang et al. [57]. Badashian et al. [58] used developers’ contributions to Stack Overflow as a proxy for their expertise for consideration during bug triaging. Building thesauri and knowledge graphs of commonly used software terms in Stack Overflow were explored [59, 61]. Other Stack Overflow research includes topic analysis [62], mining domain-specific information [63], exploring gender bias [64, 66], and emotions [29, 67]. Stack Overflow includes built-in quality signalling in the form of up and down votes, accepted answers, and user reputation. These features were used to select the input for data mining in certain studies [19, 20, 44, 54, 55, 57].
2.2 Chat Platforms as Knowledge Source

Public chats is a type of developer communication that comprises multiple communities focused on particular topics such as technology (e.g., Python or Ruby on Rails). The specific channels within a given community are assigned to general discussion or to particular subtopics [68]. More recently, chat platforms serve as a main mode of communication for different software engineering tasks such as maintaining code quality, testing, conducting development operations, supporting customers, and creating documentation; chats are replacing the traditional communication platforms, including emails and mailing lists [66, 69, 70]. This happens due to their revolutionized approach in team communications and project coordination by providing a user-friendly way of managing and organizing conversations, facilitating knowledge sharing, and by integrating with external software development tools such as GitHub, Asana, and Jira [69]. With a lot of studies on software developer communications such as Stack Overflow, it is interesting to note that fewer number of studies are conducted on studying chat platforms such as Slack, Discord, and Gitter.

2.2.1 Popular chat platforms

There are several different chat platforms used by developers and practitioners. The most popular ones are as follows:

- **IRC**: Internet Relay Chat (IRC) is a text-based chat system that works on client-server networking model with private and group messaging capabilities between multiple users. It is one of the oldest chat platforms being still in use.

- **Slack**: Slack chat platform has over 10+ million daily active users and is a multi-server and multi-channel app that allows one-on-one as well as group chat options. The files can be shared between the users and groups. The integration with other apps and services through a set of conversation-based bots and apps is the highlight in using Slack.

- **Discord**: Discord was created for video game players that allowed them to find other players and talk in real time while playing. It has now expanded to many open source communities, research labs, as well as businesses spanning
across 20+ million active daily users. It combines all the best features from other platforms like Slack and Skype and is steadily overtaking them in terms of users, video, and voice capabilities. As a public chat platform, the transcripts can be downloaded for research unlike Slack with admin rights. There is no message limit in Discord compared to the different subscription plans and features that Slack offers.

- **Gitter**: Gitter is an open source instant messaging app for software developers. It does not have a message cap limit like Slack and is also public, similar to Discord. The main advantage of Gitter is its powerful integration facility for GitHub, which is the most popular collaborative software development platform.

Chat conversations in some channels follow a Q&A format, with information seekers posting questions and others providing answers, possibly including code snippets, debugging steps, or web links. An example of a Discord chat conversation is provided in Figure 2. It is taken from the Python programming server which contains different channels such as “data-science-and-ai”, “security”, “web-development”, etc. The chat snippet presented here is from the “python-general” channel.
2.2.2 Conversation disentanglement

Chat conversations are usually entwined with each other. As we can see in Figure 2, the second utterance is a part of another conversation. The task of deciding to which conversation an utterance can be linked to is called chat disentanglement. Liu et al. [76] argued that conversation disentanglement process can be divided into two-step methods and end-to-end methods. Two-step method identifies the relation between the message pairs that are clustered accordingly to form the conversation thread. In end-to-end methods, a global conversation flow is captured. Elsner and Charniak [77] created a popular disentanglement technique based on two-step process for IRC chat transcripts. A deep learning model, Siamese Hierarchical Convolutional Neural Network was leveraged by Jiang et al. [78] to find similarities between messages to identify the messages which are under the same conversation. A disentanglement technique based on Elsner and Charniak algorithm was developed by Riou et al. [79] for a French language corpus extracted from the IRC channel of French language Ubuntu platform. Lowe et al. [80, 81] proposed the Ubuntu Dialogue Corpus and performed heuristics-based disentanglement technique on the dataset. A manually annotated disentanglement dataset based on “reply-to” relation was proposed by Kummerfeld et al. [82]. Liu et al. [76] suggested a deep co-training algorithm for disentanglement with two classifiers such as a message pair classifier and a session classifier.

2.2.3 Presence of abundant information in chat platforms

Chat platforms contain a variety of themes according to the study by Silva et al. [2]. With these diverse themes available at a single place, chats can serve as an important resource to mine information to help developers. The high-level themes are depicted in Figure 3 and have four branches as follows:

- **SD — Software development**: themes related to software development and maintenance activities;
- **SA — Software architecture**: themes related to the structure of systems and technologies;
Figure 3: Software engineering themes in chat platforms [2].

- **SQ — Software quality**: themes related to quality of functionality and correctness;

- **PD — Professional development**: themes related to software engineering learning and training.

### 2.2.4 Existing research on chat platforms

With the wide variety of themes present in developer chats as presented in Section 2.2.3, there is a growing need to mine and detect them for chat improvements and services. By facilitating chat mining and extracting useful information, developers can [83]: (1) Avoid trouble finding answers within a long chat history and identifying the right community user for answering their question. (2) Avoid repetitive questions asked by new comers to the channel and thereby eliminate unnecessary distractions. (3) Reduce spending considerable time reading extensive wikis and pages of documentation for finding specific problem solutions. (4) Overcome newcomer struggles
during on-boarding to get up to speed with their colleagues. We discuss the existing studies on chat mining to understand the research performed on chat platforms in recent years.

IRC meeting logs have been analyzed to study the content, participants, their contribution and styles of communications by Shihab et al. [84] [85]. Yu et al. [86] conducted an empirical study to investigate the use of synchronous (IRC) and asynchronous (mailing list) communication mechanisms in global software development projects. The use of channels, email discussions, and community digests to mitigate and resolve conflicts was studied by Elliott et al. [87]. The impact of Slack on team dynamics was studied by Lin et al. [66] who showed that most developers use Slack for team-wide purposes including communication and collaboration with other team members. An empirical study to analyze the characteristics of the posted questions and the impact on the response behavior on Gitter developer chat platform was studied by Ehsan et al. [88]. The Slack use by distributed global development teams has been studied by Stray et al. [89]. Panichella et al. [90] investigated collaboration links identified through data from three different kinds of communication channels: mailing lists, issue trackers, and IRC chat logs. An approach to automatically filter out off-topic IRC discussion through Stack Overflow programming discussions and YouTube video comments has been proposed by Chowdhury and Hindle [91].

Exploratory studies on the frequency and completeness of available rationale in chat messages, contribution of rationale by developers, and the potential of automatic techniques for rationale extraction were conducted by Alkadhi et al. [92] [94]. Their results show that machine learning algorithms can be leveraged to detect rationale in IRC messages with 0.76 precision and 0.79 recall. Shi et al. [95] detected feature-request dialogues from chat messages using deep Siamese network to facilitate the requirements gathering process during software development. The automation in these analyses was used to learn about developer behavior, and the potential of machine learning techniques to extract specific types of information from developer chat communications.

Chatterjee et al. [96] explored which kinds of information regarding code snippets are embedded in different software-related documents, such as developer discussions in public chat rooms. They found that in public chat rooms there are more explanatory information about code snippets (i.e., why and how a functionality was implemented
and it excepted output) than, for example, about data structure or code efficiency. Chatterjee et al. [21], for example, looked for useful information in public Slack conversations in comparison to Stack Overflow. The authors found that the largest proportion of Slack conversations are “design”-related conversations (which involve discussions of API usage and recommendations), followed by “explanatory”-type of conversations (e.g., developers explaining to each other technologies and capabilities of different programming languages). Sahar et al. [97] studied how developers discuss issue reports and found that issue reports discussed in Gitter take more time to get resolved compared to issue reports that are not discussed. Using thematic analysis, Mezouar et al. [98] identified the reasons behind the use of Slack and Gitter, the perceived impact on the associated projects and the quality characteristics of these instant messaging tools. The authors found that developers seek knowledge from instant messaging to obtain timely feedback from experts who, in return, share their expertise with others. This two-way interaction helps build developer communities and increases the reputations of those who contribute in these communities. Likewise, using thematic analysis, Silva et al. [2] obtained software engineering themes from instant messaging communication of developers. Recently, Subash et al. [38] studied the Discord chat data to understand how it can compliment the Slack dataset created by Chatterjee et al. [22] for software engineering research.

### 2.3 Opinion Mining in Chat Platforms

An opinion is defined as a ’judgment or belief not founded on certainty or proof’ [99]. Developer communications often contains an exchange of opinions or sentiments expressed about software tools and services. Developer opinions originate from their use of the particular tools or services. It can be used to improve the software development process as it shapes decisions related to software development. Opinion extraction helped the software community in many ways. Relevant API recommendation based on crowdsourced knowledge of Stack Overflow was studied by Rahman et al. [55] whereas automatic summarization of API reviews was explored by Uddin et al. [27]. Stack Overflow was mined by Lin et al. [28] for opinions on APIs using natural language parsing and pattern-matching. Novielli et al. [29] studied Stack Overflow posts to understand the impact of the emotional style of a technical question on the
Table 1: Example of an opinion-asking question and answers in Discord python-general channel.

| Q: Guys, what do you think is better - Pycharm or Visual Studio? |
| A1: I prefer pycharm as ideaVim works better than the VSC equivalent. At least from what I saw. |
| A2: Visual studio guy here. The only reason I don’t use pycharm is that sometimes I need to hop over to html or js real quick to check something out. |
| A3: Pycharm because the debugger is easier to set up and use. |

probability of promptly obtaining a satisfying answer. Sinha et al. [30] conducted a study on GitHub commit logs to identify developer sentiment polarity. To classify the content of development emails according to their purpose (e.g. feature request, opinion asking, problem discovery, solution proposal, information giving etc), Sorbo et al. [31] used Natural Language Parsing and Huang et al. [32] used a CNN model. Opinions were used to detect feature requests and software requirements from app reviews [33, 34]. These studies suggest that, beyond reducing developers’ effort in manual searches on the web and facilitating information gathering, mining of opinions could help in increasing developer productivity, improving code efficiency, and building better recommendation systems.

Even though opinions are part of other developer communication, it was noted by Chatterjee et al. [21] that potentially more opinions are present in Slack developer chats. Motivated by this fact they have created a new Question Answering system for opinion questions called ChatEO [1] that can be a starting point for building opinion-providing virtual assistants for software engineers. They argued that one type of virtual assistant that can benefit form this kind of Opinion QA system would be Conversational Search Assistants (CSA) [36] as Wizard of Oz studies have explicitly shown the need for opinions within CSAs [100]. CSA support information seekers who struggle forming good queries for exploratory search, e.g., seeking recommendations on API, tools, or resources, by eliciting the actual need from the user through conversation. Example of an opinion asking question from a Discord server is presented in Table 1. Each of the answers has sufficient information to be standalone and so, they can be considered as separate Q&A pairs along with the question and answer(s).
ChatEO architecture developed by Chatterjee et al. [1] for automatic extraction of opinion-based Q&A. We use the Slack dataset presented in their study along with the Discord chat dataset collected and published by us [38] to replace the heuristic-based question answering process with a deep learning based framework.

### 2.4 NLP and DL for Chat Mining

Most developer chat communication consists of natural language that is text data. *Natural Language Processing (NLP)* is used to process and analyse textual data for any specific task. NLP is a branch of artificial intelligence that helps computers understand, interpret, and manipulate human language. NLP draws from many disciplines, including computer science and computational linguistics, in its pursuit to fill the gap between human communication and computer understanding [101]. NLP tasks are split into two areas: core and applications [102].

The tasks addressed in the **core areas** are:

- *Language Modeling* quantifies associations among naturally occurring words.
- *Morphology* deals with segmentation of meaningful components of words and identifying parts of speech words.
- *Parsing* also called syntactic processing that builds sentence diagrams as possible precursors to semantic processing.
- *Semantics* attempts to distil the meaning of words, phrases, and higher-level components in text.

The **applications** area involves topics such as:

- *Information Retrieval* is the process, methods, and procedures of searching, locating, and retrieving recorded data and information from a file or database.
- *Information Extraction* is the process of parsing through unstructured data and extracting essential information into more editable and structured data formats.
• **Text Classification** is the task of assigning a label or class to a given text. It can be single-label or multi-label classification.

• **Text Generation** leverages knowledge in computational linguistics and artificial intelligence to automatically generate natural language texts, which can satisfy specific communicative requirements.

• **Summarization** finds elements of interest in documents to produce an encapsulation of important content.

• **Question Answering** involves answering natural language questions posed by humans, using models that draw information from textual sources.

• **Machine Translation** involves mathematical and algorithmic techniques to translate documents from one language to another.

When applying NLP techniques to any text problem, we usually combine any combinations of core areas and applications. In our research on mining chat textual data for creating an Opinion Question Answering system, we use all the core areas of NLP for Question Answering and Text Classification applications.

In the past, NLP tasks were built upon machine learning techniques using shallow models (e.g., SVM and logistic regression) trained on very high dimensional and sparse features. In recent years, neural networks based on dense vector representations are producing superior results on various NLP tasks [37]. This trend was possible with the success of word embeddings [103] and deep learning methods [104] which allow multi-level automatic feature representation learning rather than hand-crafted features used by machine learning that was time-consuming and often incomplete. The first work showed how a simple deep learning network outperforms the state-of-art approaches in most NLP tasks such as named-entity recognition (NER), semantic role labelling (SRL), and POS tagging by Collobert et al. [105]. Since then, numerous complex deep learning algorithms were proposed to solve difficult NLP tasks. We explain certain DL concepts and models used for NLP tasks in this section.
2.4.1 Word embeddings

To avoid the *curse of dimensionality*\(^{106}\) while learning joint probability functions of language models, distributed representations of words existing in low-dimensional space were introduce\(^{107}\). These distributional vectors or word embeddings are built upon the concept that words with similar meanings tend to occur in a similar context. Word embeddings have become popular following the work of Mikolov et al.\(^{103}\) on continuous bag-of-words (CBOW) and skip-gram models called *Word2Vec* that efficiently construct high-quality distributed vector representations. In their model, when 2 word vectors are combined, it results in a vector that is a semantic composite of the individual words, e.g., “man” + “royal” = “king”. The CBOW model computes the conditional probability of a target word given the context words surrounding it across a window of size \(k\). On the other hand, the skip-gram model does the opposite of CBOW model by predicting the surrounding context words given the central target word. Pennington et al.\(^{108}\) created another famous word embedding called *GloVe* based on a “count-based” model. Here, the word co-occurrence count matrix was pre-processed by normalizing the counts and log-smoothing operation. It was then factorized to get lower dimensional representations by minimizing a “reconstruction loss”. Word embeddings such as fastText\(^{109}\), Elmo\(^{110}\), BERT\(^{111}\) have evolved since then helping in various NLP tasks.

2.4.2 DL models in NLP tasks

The popularity of word embeddings drove the research community to develop numerous DL models with the embeddings as their features for NLP tasks. In recent years, the DL models such as Recurrent Neural Networks (RNN)\(^{112}\), Convolutional Neural Networks (CNN)\(^{113}\), Variational Autoencoders (VAE)\(^{114}\) and Generative Adversarial Networks (GANs)\(^{115}\) are widely used for NLP tasks. We will explain each of them in this Section.

**RNN** processes sequential information in a recurrent way where the output is dependent on the previous computations and results\(^{116}\). This quality makes them a great candidate for natural language text processing like chat data which is sequential in nature. Conceptually RNN differs from a standard neural network as the standard input in an RNN is a word instead of the entire sample as in the case of a standard
neural network. It gives the flexibility to work with varying lengths of sentences, which is possible in a standard neural network due to its fixed structure. RNN also provides an additional advantage of sharing features learned across different sentence positions. RNN processes the sequence one element at a time in time steps. At any given time step, it produces the output and has the hidden state. It calculates the hidden state by combining the previous hidden state and current input and multiplies them with weight matrices to get the output. There are various types of RNN used for NLP tasks such as Long Short Term Memory (LSTM) [117], Bi-LSTM [118], Attention-based LSTM [119,120] and Gated Recurrent Unit (GRU) [121]. A simple RNN architecture is shown in Figure 4.

There are several studies that included RNN for NLP and text mining in the recent years. Tai et al. [122] created a Tree-LSTM model, a generalization of LSTM to tree-structured network typologies, to learn rich semantic representations. They argued that tree-based LSTM is better for NLP tasks because natural language exhibits syntactic properties that would naturally combine words to phrases. The same concept was followed by Zhu et al. [123] when they created a tree-based LSTM using a memory cell to store the history of multiple child cells or multiple descendant cells in a recursive process. They stated that this is a principled way of considering long-distance interaction over hierarchies, e.g., language or image parse structures. Cheng et al. [124] replaced the single memory cells of LSTM with a memory network that achieves promising results in language modelling, sentiment analysis, etc. A multitime scale LSTM was created by Liu et al. [125] to model long texts, such as sentences and documents, by capturing valuable information with different timescales. On the Bi-LSTM side, Zhou et al. [126] integrated a Bidirectional-LSTM (BiLSTM) model with two-dimensional max-pooling to capture text features. Wan et al. [127] explore semantic matching using multiple positional sentence representations generated by a
bi-directional LSMT model. On the GRU front, Jozefowicz et al. \cite{128} and Liu et al. \cite{129} proved that GRU outperforms LSTM in most of the NLP tasks.

**CNN** \cite{113} gets its name from the convolution operation in mathematics and signal processing. RNNs were trained to recognize patterns across time, whereas CNN’s learn to recognize patterns across space \cite{104}. CNN’s use functions, known as filters, allow for simultaneous analysis of different features in the data and create a feature map. Often, it is not important precisely where certain features occur, but rather whether or not they appear in particular localities. Therefore, pooling operations is needed to minimize the size of feature maps. The sizes of such pools are generally small to prevent the loss of too much precision. CNN consists of multiple pairs of these two layers and the result of this sequence of operations is then typically connected to a fully connected layer (traditional multi-layer perceptron neural network (MLP)). CNN for text processing was popularized by Collobert et al. \cite{105,130} when they proposed a general CNN-based framework to solve a plethora of NLP tasks. It was further improved for NLP tasks by Kalchbrenner et al. \cite{131} along with Kim et al. \cite{4}. The basic structure of CNN for text data is displayed in Figure 5. The static and non-static channels refer to the two-word embedding input - one static and one fine-tuned using backpropagation.

CNN works well when detecting local and position-invariant patterns is important. These patterns could be key phrases that express a particular sentiment like “I like”
or a topic like “endangered species”. Some modification to the CNN architecture proposed by Kim et al. [4] was done by Liu et al. [132]. First, a dynamic max-pooling scheme is adopted to capture more fine-grained features from different document regions. Second, a hidden bottleneck layer is inserted between the pooling and output layers to learn compact document representations that can reduce the model size and increase the performance. Character-level CNN was explored by Zhang et al. [133], while Kim et al. [134] took as input the characters in a fixed-sized, encoded and passed them through a deep CNN model. Prusa et al. [135] presented an approach to encoding text using CNNs that greatly reduces memory consumption and training time required to learn character-level text representations. Mou et al. [136] proposed a tree-based CNN to capture sentence-level semantics. Pang et al. [137] cast text matching as the image recognition task and used multi-layer CNNs to identify salient $n$-gram patterns. Wang et al. [138] proposed a CNN-based model that combines explicit and implicit representations of short text. Poria et al. [139] created a Deep CNN with an auxiliary support in the form of pre-trained networks trained on emotion, sentiment, and personality datasets that achieved state-of-the-art performance. Usually NLP tasks involving micro texts using CNN-based methods require the need for additional information and external knowledge to perform as per expectations. Chen et al. [140] proposed a modified pooling strategy: dynamic multi-pooling CNN (DMCNN). This strategy used a novel dynamic multi-pooling layer that, as the name suggests, incorporates event triggers and arguments to reserve more crucial information from the pooling layer.

**VAE** [114] is a deep generative model that inspired by both Autoencoders [141] and Variational Inference. Variational inference is widely used to approximate distributions in complex latent variables [142]. VAEs address the issue of non-regularized latent space in autoencoder and provide the generative capability to the entire space. The encoder in the normal autoencoder outputs latent vectors. Instead, the encoder of VAE outputs parameters of a pre-defined distribution in the latent space for every input. The VAE then imposes a constraint on this latent distribution forcing it to be a normal distribution. This constraint makes sure that the latent space is regularized. The difference between autoencoder and general VAE is presented in Figure 6.

Bowmen et al. [143] successfully transferred VAE from computer vision to natural language processing and proposed a novel variational method. They created an
RNN-based variational autoencoder incorporating distributed latent representations of entire sentences. Hu et al. [144] proposed generating sentences whose attributes are controlled by learning disentangled latent representations with designated semantics. The authors augmented the latent code in the VAE with a set of structured variables, each targeting a salient and independent semantic feature of sentences. The model incorporated VAE and attribute discriminators, in which the VAE component trained the generator to reconstruct real sentences for generating plausible text while the discriminators forced the generator to produce attributes coherent with the structured code. To enable VAE to model long-term text, variational hierarchical recurrent encoder-decoder (VHRED) was created by Serban et al. [145]. Under the assumption of VHRED, no linguistic label was supplied and the model works in a purely unsupervised way. SPHRED [146] (Hierarchical recurrent encoder-decoder with separated context) resolved this assumption by studying the conditional generation where partial or all linguistic attribute labels are incorporated into the latent variable. Following the idea of SPHRED, Zhao et al. [147] regard attribute labels as extra knowledge. They propose kgCVAE, incorporating linguistic features as extra knowledge for the latent variables.

GAN [115] is another class of generative models composed of two competing networks. A generative neural network decodes latent representation to a data instance, while the discriminative network is simultaneously taught to discriminate between instances from the true data distribution and synthesized instances produced by the generator. The GAN architecture is illustrated in Figure 7.

GAN was first used in NLP by Li et al. [148]. The model architecture is straightforward: a generative model to generate a response given a query and a discriminative model to distinguish the matching or mismatching between a given query and response. Xu et al. [149] developed an approximate embedding layer, directly converting the prediction of the next token into a mixed word embedding, whose mixing
weight is the probability given by the softmax layer of the decoder. A straightforward design of discriminator is a classifier, outputting a confidence score as result. However, Xu et al. [150] proposed a language based discriminator to avoid the saturation problems of the classifier based one. Adversarial Information Maximization (AIM) [151] explores another kind of discriminator, whose training objective for its discriminator is to maximize the cosine distance of a pair of positive examples and at the same time minimize the cosine distance of a pair of negative examples. The cosine distance is measured in a shared embedding space. Recently, Feng et al. [152] put forward a new framework with a forward discriminator and a backward discriminator. It is worth noting that this work introduces future information and reformulates the context-response pair to triplets in the format of (context, response, future). So the two discriminators are responsible for checking the forward pass (context $\rightarrow$ response) and the backward pass (future $\rightarrow$ response), respectively.

### 2.5 Automated Labelling, Class Balancing & Hyper-parameter Tuning

While using deep learning models, three concepts needs to be considered to obtain the highest performance. These are training the DL models on a large dataset, balancing the dataset, and hyperparameter tuning to the model. These concepts are discussed in Section 2.5.1, Section 2.5.2 and Section 2.5.3 respectively.
2.5.1 Automated labelling with weak supervision

One of the costly requirements of DL is that they need massive training sets of labelled examples to learn from, often tens of thousands to millions to reach peak predictive performance \[39\]. Such training sets are expensive, provided domain experts are available. So most of the research and industry experts are turning toward weak supervision \[13\] that is a cheaper source of noisy or heuristic labels. Weak supervision techniques used in literature are distant supervision, crowd-sourced labelling, rules, and heuristics.

Each of these weak supervision techniques lacks accuracy and coverage. To increase these metrics, Ratner et al. \[13\] have developed Snorkel, the first end-to-end system for combining weak supervision sources to rapidly create training data using data programming \[153\]. Snorkel is widely used in the industry as it enables users to use weak labels from all available weak label sources, supports different classifiers, and provides rapid results in response to the user’s input. It proves its efficiency with a case study at the industrial level \[40\]. First, Snorkel lets users write their labelling functions (LFs) to express a wide range of weak supervision sources such as patterns, heuristics, and external knowledge bases, and then combines these sources. The conflicts and correlation between the labels are then mapped and resolved using a generative model. At this point, The labels attached to data points are probabilistic labels that are used to train a discriminative model. The discriminative model aims at generalizing beyond the noisy LFs to predict the true labels.

In our work, we use Snorkel to generate a larger labelled dataset for the DL model in addition to a manually labelled smaller dataset. We compare the results on both datasets to understand the advantages of automatic labelling.

2.5.2 Class balancing techniques

In the real-world application of machine/deep learning, the target classes are most often disproportionate (i.e., one or more classes are highly represented against others). This class imbalance problem where the majority class has more samples than the minority class while training results in poor learning of the minority classes leading to inaccurate predictions. It was studied by Gou et al. \[154\] where they argued that a
balanced dataset helps in increasing the predicting capabilities of the machine/deep learning algorithms. There are different techniques to resolve this class imbalance problem.

- **Resampling**: Simple resampling techniques where the majority class is under-sampled or a minority class is over-sampled can resolve class imbalance problem. However, in practice, these simple sampling approaches have flaws. Undersampling the majority class can lead to leaving out instances that provide important differences between the classes. Oversampling the minority class can lead to model overfitting, since it will introduce duplicate instances, drawing from a pool of data that is already small. There is another oversampling method called Synthetic Minority Oversampling Technique (SMOTE) which synthesizes new examples from the minority class instead of duplicating them.

- **Cost-sensitive learning**: In regular learning, we treat all misclassifications equally, which causes issues in imbalanced classification problems, as there is no extra reward for identifying the minority class over the majority class. Cost-sensitive learning changes this, and uses a function \( C(p, t) \) (usually represented as a matrix) that specifies the cost of misclassifying an instance of class \( t \) as class \( p \). This allows us to penalize misclassifications of the minority class more heavily than we do with misclassifications of the majority class, in hopes that this increases the true positive rate. A common scheme for this is to have the cost equal to the inverse of the proportion of the data-set that the class makes up. It increases the penalization as the class size decreases.

- **Metrics**: Whenever we encounter an imbalance problem, we look at the trade-off between metrics. In situations where we want to detect instances of a minority class, we are usually concerned more with recall than precision. Because in the context of detection, it is usually more costly to miss a positive instance than to falsely label a negative instance. Thus, when comparing approaches to imbalanced problems, one need to consider using metrics beyond accuracy such as recall, precision, and AUROC. Switching metrics can most often than not help the models to learn the minority class better.
2.5.3 Hyper-parameter tuning

Deep learning models have a variety of hyper-parameters that need to be tuned to obtain the state-of-the-art or even better results using them. Some of these hyperparameters, such as the number of layers or the number of neurons per layer, are bound directly to the deep neural architecture. Others like drop-out rate are independent of the architecture. Hyperparameter tuning uses grid search/brute force in most cases, where all possible combinations of the hyperparameters with all parameter values form a grid, followed by training an algorithm on each combinations. As the number of hyperparameters grows, grid search becomes computationally expensive. To address this, Bergstra et al. [156] created a randomized parameter tuning and showed that for each of their datasets, there are only a few impactful parameters on which more values should be tried. But as each trial is independent of the other, they do not learn anything from the other. To rectify this, Snoek et al. [157] proposed the Bayesian optimization method using a statistical model for mapping hyperparameters to an objective function. As this method add another level of complexity to the already complex hyperparameter tuning problem, this has not become very popular. The most popular one is still the ad-hoc grid search created by Hutter et al. [158] despite the training time. Here, the authors manually tried the most correlated parameters on the same grid to iteratively find the most impactful set of hyperparameters with the best values.
Chapter 3

Methodology

This chapter describes the methodology we followed in identifying the opinion-based questions in developer chat conversations. Section 3.1 describes the methodology in collecting and disentangling developer chats, while Section 3.2 describes the modified ChatEO architecture for opinion question identification. Section 3.3 explains how Snorkel, a weak learner is adapted to labelling a bigger data set as a source for ChatEO’s DL models. Lastly, Section 3.4 describes the evaluation metrics used in this thesis work.

3.1 Data Collection and Dataset

This section explains our dataset collection process in detail. Our dataset was created by combining two similar datasets. We have published one of the datasets, called DISCO [38], with one-year public Discord chat conversations of four software development communities (python, go-lang, racket, clojure) in the Data Showcase track of the 2022 International Conference on Mining Software Repositories [159]. The other dataset was published by Chatterjee et al. [22] on software-related Q&A chat conversations, curated for two years from three open Slack communities (python, clojure, elm). The combined dataset consists of 67,667 conversations, 19,45,986 utterances, contributed by 3,35,733 users. The process of collecting a DISCO dataset is explained in Section 3.1.1 and a brief overview of the Slack dataset is presented in Section 3.1.2.
3.1.1 DISCO dataset

The overall process of the Discord data collection and conversation disentanglement is given in Figure 8. The Discord chat transcripts from our selected channels are first downloaded in JSON format by providing a date range. They are then cleaned to retain only helpful information such as timestamp, user name, and message content and converted into XML format. This was followed by anonymizing the usernames in XML to ensure the privacy of the users and eliminating the possibility of identifying the original Discord users. The disentanglement algorithm [22] was then leveraged to extract disentangled Discord conversations (in XML format). The final dataset includes an additional computed attribute, <conversation_id>, as part of each message utterance.

3.1.1.1 Data selection

We select Discord public server channels as the source in creating the dataset that can support interesting research opportunities and tool development. Our Discord chat data complements Slack data [22] to foster further research on studying distributed software development communities, communication among community and team members, informal documentation, etc. While Slack’s free plan supports only 10,000 of the most recent messages to be searched and viewed, Discord does not impose such limitations and preserves all the historical chat data. Hence, many software development communities have started to migrate their communication from Slack to Discord; while some communities continue to maintain both communication mediums [160,161]. Gitter is another instant messaging and chat platform designed for GitHub and GitLab users where the discussions are happened on specific projects. Since many open-source communities use Discord as their communication platform,
Table 2: Dataset of disentangled Discord conversations.

<table>
<thead>
<tr>
<th>Channel</th>
<th>#Conver.</th>
<th>#Utter.</th>
<th>#Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>python#python-general</td>
<td>19,155</td>
<td>1,254,362</td>
<td>300,919</td>
</tr>
<tr>
<td>gophers#golang</td>
<td>8,860</td>
<td>247,179</td>
<td>19,983</td>
</tr>
<tr>
<td>racket#general</td>
<td>538</td>
<td>4,975</td>
<td>917</td>
</tr>
<tr>
<td>clojurians#clojure</td>
<td>159</td>
<td>1,577</td>
<td>1,743</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>28,712</strong></td>
<td><strong>1,508,093</strong></td>
<td><strong>323,562</strong></td>
</tr>
</tbody>
</table>

it is important that the conversation data from 150 million active Discord users is collected and available to researchers.

For creating our dataset, we select Discord servers for four programming languages such as Python, Go (or GoLang), Racket, and Clojure which demonstrate a good daily activity and a substantial number of members (e.g., Python Discord server has a total of 300,919 members) compared to other public Discord programming servers. Anyone with a Discord user ID can join these servers as they are publicly visible and start asking general or technical help questions on these channels. We identified the following server channels that follow a Q&A format and offer general technical help including python#python-general, gophers#golang, racket#general, and clojurians#clojure for our data collection and conversation disentanglement process. To allow triangulation with previous datasets [22], we have selected similar channels. Metrics such as the number of conversations, utterances and users for each channel are reported in Table 2.

3.1.1.2 Data collection and preprocessing

Data from the Discord channels is exported as JSON files using an open-source application, Discord Chat Exporter [162], with a specific date range. The date range for three channels (Python, Clojure, Racket) is from Nov–2019 to Oct–2020, while for gophers#golang the date range is Nov–2019 to Sep–2020 due to our University’s Fair Dealing Policy in using public copyrighted data for research purposes [24].

The collected Discord chat transcripts in JSON format are then converted to XML files. Each message in the resulting XML has three tags such as a timestamp, the ID
of the user and the message text. All other information in the JSON files such as the user-related details, reactions on the messages, etc. is removed during JSON to XML conversion. The user IDs are then anonymized using the randomly selected person names to preserve the privacy of the channel users.

3.1.1.3 Conversation disentanglement

In chat servers, the message transcripts are formed by different conversations (both formal and informal) happening simultaneously. Figure 9 illustrates an example of a preprocessed Discord XML file. The presented XML snippet covers two separate conversations entangled with each other. The 2nd question was asked when the first conversation was in progress. The 1st conversation was then continued before a relevant reply to the 2nd question was given. This interlinked conversation flow makes it difficult to mine the chat data.

To enable mining of the chat transcripts for researchers and tool makers, we need to disentangle these conversations. The disentanglement techniques have been previously proposed for IRC [163], Gitter [23], and Slack [22]. One of the recent research on Slack data leveraged the well-known Elsner and Charniak disentanglement technique [77] with some modifications. The original Elsner and Charniak disentanglement technique used a supervised model that considers the time frame and features between the message pairs. It also considers the user similarity between the message pairs, cue words, similar word usage, and technical expressions while disentangling the chats. For Slack data, the Elsner and Charniak technique was modified on the feature computation between the message utterances [22].

The features were calculated 1) when the time frame of $\leq 1477$ (1.5$^{18}$) seconds was observed between the message utterances, or 2) when the utterance was within the last 5 messages from one another. New features that are specific to Slack including gratitude words (e.g., “thanks”, “this works”, “makes sense”) were also added to the modified algorithm. The modified classifier was then trained on 500 manually disentangled Slack conversations.

We have adopted Chatterjee et al.’s disentanglement technique [22] on Slack data for our Discord data since both Slack and Discord channels follow the same type of conversations in Q&A format. To check the accuracy of the disentanglement process,
the two first authors selected a random block of 500 Discord messages extracted from the Python channel, manually disentangled these messages into conversations, and calculated a micro-averaged F-score. Our average F-score was 0.79, similar to the one reported by Chatterjee et al. [22] for Slack disentanglement (i.e., F-score of 0.80) which is higher than the F-score of 0.66 reported by Elsner and Charniak. As the annotators can disagree with the disentanglement process, a micro-averaged F-score can be used as an appropriate metric to calculate the quality of disentaglement [77]. This result further supports our observation that Slack and Discord follow similar chat conversation patterns where multiple questions are asked and answered simultaneously. This observation also suggests that the same disentanglement algorithm used on Slack can be applied to Discord.
Table 3: Dataset of disentangled Slack conversations.

<table>
<thead>
<tr>
<th>Channel</th>
<th>#Conver.</th>
<th>#Utter.</th>
<th>#Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>pythondev#help</td>
<td>8,887</td>
<td>106,262</td>
<td>3295</td>
</tr>
<tr>
<td>clojurians#clojure</td>
<td>7,918</td>
<td>72,973</td>
<td>2,422</td>
</tr>
<tr>
<td>elmlang#beginners</td>
<td>13,169</td>
<td>168,689</td>
<td>3,695</td>
</tr>
<tr>
<td>elmlang#general</td>
<td>8,981</td>
<td>89,969</td>
<td>2,759</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>38,955</strong></td>
<td><strong>437,893</strong></td>
<td><strong>12,171</strong></td>
</tr>
</tbody>
</table>

3.1.2 Slack dataset

We have also included the Slack dataset published by Chatterjee et al. [22] that complemented our DISCO dataset, as part of our source. Slack’s three programming communities - python, clojure and elm were chosen that have an active presence where the admins are willing to provide API tokens to download. The channel chosen was of a Q&A format with a date range of July–2017 to Jun–2019 - pythondev#help, clojurians#clojure, elmlang#beginners, and elmlang#general. The chat transcripts were first downloaded as JSON and converted into XML with required tags before being subjected to a disentanglement process. The disentanglement technique by Elsner and Charniak [77] was modified for the task with a manually labelled training set of 500 Slack messages. This modified technique was subjected to a micro-averaged F-measure using a separate set of manually labelled messages and it resulted in 0.80. This is a strong improvement over the vanilla Elsner and Charniak approach’s micro-averaged F-measure of 0.66.

The data format is same as the DISCO dataset with each message having a conversation id <message conversation_id>, timestamp <ts>, anonymized participant names <user> for privacy, and the content <text> of the message. Few metrics about the dataset are reported in Table 3.

3.2 Modified ChatEO for Question Identification

ChatEO was developed by Chatterjee et al. [1] to automatically Extract Opinion Q&A from software developer Chats using Slack dataset. Our research focuses on
improving ChatEO’s question identification process using the dataset created in Section 3.1. Section 3.2.1 gives a brief explanation of the original ChatEO’s architecture while our modifications for the question identification task are presented in Section 3.2.5.

### 3.2.1 Original ChatEO architecture

One interesting aspect of the dataset collected is the prevalence of opinion-asking questions in chat than in any other developer communications [21]. Mining these opinions can pave the way for developing software tools and services such as automated virtual assistants, chatbots, Q&A thesaurus, and more. This knowledge motivated Chatterjee et al. to build a system to automatically identify the opinion-asking question and extract related answers called ChatEO [1] for Slack chat transcripts. The overall process of ChatEO [1] is depicted in Figure 10.

ChatEO takes chat data as a source and extracts the opinion Q&A pairs using three steps: 1. The interleaved chat transcripts are subjected to a disentanglement process to extract conversations from chat. 2. The conversations starting with opinion-asking questions are identified using textual heuristics. 3. Potential answer utterances within the conversations are identified using DL based approach. Section 3.2.2, Section 3.2.3, and Section 3.2.4 explain each of these processes, respectively.
3.2.2 ChatEO - conversation disentanglement

As chat messages with entwined conversations are the source for ChatEO, the messages need to be disentangled to allow individual conversation analysis. ChatEO uses the modified Elsner and Charniak disentanglement technique by Chatterjee et al. to disentangle the conversations. The modified algorithm showcases a micro-averaged F-measure of 0.80 that is higher than the original technique. The modifications are mentioned in Section 3.1.1.3 and the same is used in our research. After disentanglement, we obtain the output XML file where each utterances has a conversation ID, timestamp, and author information.

3.2.3 ChatEO - heuristics based question identification

The next step in ChatEO is an opinion-asking question identification task using a heuristics-based approach. We explain the original process used by ChatEO in this section, however, we have replaced it in our research with deep learning models. The motivation behind the original approach was the claim by Di Sorbo et al. that when discussing development topics, developers use recurrent linguistic patterns. The replication package, DECA by Di Sorbo et al. contains 5 linguistic patterns to identify opinion asking questions in developer emails. These patterns only identified 2 out of 400 as opinion questions in the manual analysis chat conversation dataset. Thus, Chatterjee et al. decided to perform a qualitative content analysis procedure to find additional linguistic patterns to identify opinion-asking questions in developer chats.

They manually analyzed the chat dataset to identify the parts of the utterances like part-of-speech tags and recurrent keywords that led them to categorize them as opinion-asking questions. The additional patterns identified by Chatterjee et al. exhibit any of the following characteristics:

- Expects subjective answers (i.e., opinions) about APIs, libraries, examples, resources, e.g., “Is this a bad style?”, “What do you think?”
- Asks for which path to take among several paths, e.g., “Should I use X instead of Y?”
Table 4: Example of a linguistic pattern for opinion-asking question and answers.

<table>
<thead>
<tr>
<th>Pattern code:</th>
<th>P_WHAT_RECADJ_TARGETNOUN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Question starting with What—Which and followed with a positive adjective and noun.</td>
</tr>
<tr>
<td>Rule:</td>
<td>`What</td>
</tr>
<tr>
<td>Definitions:</td>
<td><code>[rec positive adj] = ”good”,”better”,”best”,”right”,”optimal”,...</code></td>
</tr>
<tr>
<td></td>
<td><code>[rec target noun] = ”project”,”api”,”Visual Studio”,”Pycharm”,...</code></td>
</tr>
</tbody>
</table>

- Asks for an alternative solution (other than questioner’s current solution), e.g., “Is there a better way?”

We refer readers to Table 1 in Section 2.3 for an example of opinion-asking question. The linguistic pattern corresponding to this opinion-asking question is presented in Table 4. Chatterjee et al. added 10 linguistic patterns by extending Di Sorbo et al.’s work to identify opinion-asking questions.

### 3.2.4 ChatEO - DL based answer extraction

The final step in ChatEO is the answer extraction process for the opinion-asking questions identified in the previous step. There is no modification of this part of ChatEO in our research. Chatterjee et al. [1] used a modified version of the R−CNN designed by Zhou et al. [166] for a community question answering (CQA). They have used R−CNN as opposed to other answer extraction models [167, 169] as R−CNN take semantic links between successive answer candidates along with the semantic relevance between question and answer. As chat conversation has multiple exchanges happen in a fast and short way, finding out the context of the entire discussion is necessary to identify the answers to the questions asked. As the original R−CNN was created for CQA in the non-SE domain, Chatterjee et al. [1] customized it for the SE domain.

The modifications in the original R−CNN for ChatEO by Chatterjee et al. [1] are listed below.
• **Preprocessing:** The disentangled conversations are processed to facilitate the identification of semantics content in the chats by ChatEO. The steps include the replacement of URLs, user mentions, emojis and code with the specific tokens of the same name - 'URL', 'username', 'emoji', and 'code'. The text is then subjected to a manual phase expansion step and converted into lowercase.

• **Word embeddings for SE data:** As developer communication can differ from general English text in vocabulary and semantics, a customized GloVe embeddings was used after preprocessing. It was trained on Stack Overflow (as of June 2020). The final word embedding model has 123,995 words, where each word is represented by a 200-dimensional word vector. This custom word embedding model was applied to each word in each utterance of a conversation.

• **CNN to learn Q&A pair joint representation:** TextCNN [4] was used by ChatEO to learn the joint Q&A pair representation instead of two convolutional layers in the original R−CNN architecture as text are much shorter than CQA posts. Each utterance of the disentangled conversation subjected to customized GloVe word embeddings is taken as individual sentences and used as the input. Once the utterances are padded with zero vectors to maintain the same length, it is subjected to a convolutional layer with a fixed length kernel. The convolutional layers were paired with a max pooling layer that selects the most effective information with the highest value. The flattened output vectors for each kernel after max-pooling are concatenated as the final output.

• **Bi-LSTM:** To identify the potential answers in the conversation, we need to understand the flow and context of the conversation. This was achieved in ChatEO by passing the output of TextCNN as input to a Bi-directional LSTM. For answer extraction tasks in literature, researchers use a variation of LSTM [170][171] as it was capable of modelling semantic links between continuous text to perform answer sequence learning. Bi-LSTM is used in ChatEO instead of LSTM to improve the prediction of the answers in a conversation. Each conversation is processed as a sequence in two opposite directions (forward and backwards) to get two LSTM outputs, which will then be concatenated to form the final Bi-LSTM output.
3.2.5 Modifications in ChatEO - DL models for question identification

Our research focuses on the opinion-asking question identification part of ChatEO while the answer identification part remains the same. As explained in Section 3.2.3, ChatEO use a heuristic-based question identification process which consists of linguistic patterns. These rules are created using domain-specific knowledge of chat by analysing the chat transcripts. To move on from the extensive and expensive manual process of forming the heuristic rules, we decided to use the DL models that are proven efficient for NLP tasks [172, 173]. Section 3.2.6 provides information on different word embeddings that were used as a feature in our DL models, while Section 3.2.7 describes the DL models used in our opinion-asking question identification task.

3.2.6 Word embeddings as a feature for DL models

Feature engineering is an important step when using DL models as its domain knowledge can make the model interrupt the data well and produces better results. To understand a natural language like chat data, we need to understand the context of the sentences better for the model to produce good results. Word embeddings is a technique where individual words of a domain or language are represented as real-valued vectors in a lower dimensional space. By translating large sparse vectors into a lower-dimensional space, these embedding preserves the semantic relationships in text data and their context.

Even though chat data used in our research is from the SE domain, chat transcripts usually follow an informal general format. So, we have used both - general and software data trained word embeddings on the disentangled chat conversations dataset described in Section 3.1 before passing it to the different DL models. All the embedding variants used are either Word2Vec or GloVe and are pre-trained. We discuss both in detail and explain the variants used for our task. The five different word embeddings used in our modified ChatEO are as follows:

- Word2VeC - General
• Word2Vec - SO trained
• GloVe - General
• GloVe - SO trained
• GloVe - Twitter trained

3.2.6.1 Word2Vec embedding

Word2Vec was created by Mikolov et al. [7, 103] from Google, it processes the input text and outputs a vector representation for each word. The architecture of Word2Vec contains a feed-forward neural network with one hidden layer and is referred to as shallow neural network architecture. It relies on a hypothesis that the neighbouring words in a text have semantic similarities with each other. It uses the cosine similarity metric to measure semantic similarity. Cosine similarity is equal to \( \cos(\text{angle}) \), where the angle is measured between the vector representation of two words/documents. The generated vectors based on the common context words are located close to each other in the vector space. There are two approaches in Word2Vec - skip-gram tries to use the current word to predict its neighbours (i.e., its context), while Continuous Bag Of Words (CBOW) uses each of these contexts to predict the current word. CBOW is quick and finds better numerical representations for frequent words, while skip-gram can efficiently represent rare words. Figure 11 shows the architecture of both skip-gram and CBOW.

In the skip-gram model, the objective function is to maximize the likelihood of the prediction of contextual words given the centre word, \( w_t \). Formally, given a sequence of training words \( w_1, w_2, w_3, ..., w_T \), the objective of the skip-gram model is to maximize the average log probability:

\[
\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} \sum_{c \leq j \leq c, \ j \neq 0} \log p(w_{t+j}|w_t) \tag{1}
\]

where \( c \) is a hyperparameter defining the window of context words that is the function of the centre word \( w_t \). To obtain the output probability \( p(w_a|w_i) \), the model estimates a matrix \( O \in \mathbb{R}^{V \times d_w} \), which maps the embeddings \( r_{w_i} \) into a \( |V| \)-dimensional
vector $o_{w_i}$. Then, the probability of predicting the word $w_o$ given the word $w_i$ is called a softmax objective and is defined as:

$$p(w_o|w_i) = \frac{e^{O_w(w_o)}}{\sum_{w\in V} e^{O_w(w)}}$$

(2)

Whereas, the CBOW model predicts the center word $w_o$ given a representation of the surrounding words $w_c, ..., w_1, w_1, w_c$. Thus, the output vector $o_{w_c, ..., w_1, w_1, w_c}$ is obtained from the product of the matrix $O \in \mathbb{R}^{|V| \times d_w}$ with the sum of the embeddings of the context words $\sum_{-c \leq j \leq c, j \neq 0} r_{w_j}$.

The two variants of Word2Vec pre-trained embeddings used in our work are Word2Vec General and Word2Vec Stack Overflow trained. Word2Vec General was trained on a part of the Google News dataset that contains about 100 billion words. The model has 300—dimensional vectors for 3 million words and phrases and it can be downloaded in simple binary format [174]. Word2Vec Stack Overflow trained word vectors were created by Efstathiou et al. [175] to bridge the lack of domain-specific pre-trained models related to software engineering. It was trained on 15GB of textual data from Stack Overflow posts from August 2008 to December
2017. The model was trained with vector dimensions of 200 features, a lower dimension than the 300−feature state-of-the-art Google news vectors as the authors argued that a lower dimension should be sufficient for capturing the necessary features for the much smaller dataset with limited vocabulary. The resulting model consists of a vocabulary of 1,787,145 keywords.

### 3.2.6.2 GloVe embedding

GloVe embedding is short for “Global Vectors” for word representation and was created by Pennington et al. [108] of Stanford University in 2014. Unlike Word2vec, GloVe does not rely just on local statistics (local context information of words) but also incorporates global statistics (word co-occurrence) to obtain the word vectors. It relies on the idea that we can derive the semantic relationship between the words from the co-occurrence matrix. Given a dataset of $V$ words, $V \times V$ will the co-occurrence matrix $X$, where the $i^{th}$ row and $j^{th}$ column of $X$, $X_{ij}$ denotes how many times word $i$ has co-occurred with word $j$. Then this matrix to factorized to yield a lower-dimensional matrix, where each row now yields a vector representation for the corresponding word. In general, this is carried out by minimizing a reconstruction loss. The loss tries to find the lower-dimensional representations which can explain most of the variance in the high-dimensional data.

Formally, Pennington et al. [108] defined the probability that the word $j$ appears in the context of the word $i$ as:

$$P_{ij} = \frac{X_{ij}}{\sum_k X_{ik}}$$  \hspace{1cm} (3)

where $\sum_k X_{ik}$ is the number of times any word appears in the context of word $i$. This probability helps at quantifying the relevance of a word with regard to others. For instance, given three words $i$, $j$ and $k$, the quantity $Q$ acts as a discriminative feature.

$$Q = \frac{P_{ik}}{P_{jk}}$$  \hspace{1cm} (4)

Here, $Q \ll 1$ means that word $k$ is more related to $i$ than $j$, $Q \gg 1$ means that
word $k$ is less related to $i$ than $j$, $Q \simeq 1$ means that word $k$ is equally related to $i$ and $j$. The main idea behind GloVe is that the word probabilities are converted into word vectors $v^W_{wi}$, $v^W_{wj}$ and $v^C_{wk}$ in $R^D$ by approximating $Q$ using the vectors leveraging a function $F$ given as:

$$F(v^W_{wi}, v^W_{wj}, v^C_{wk}) = Q = \frac{P_{ik}}{P_{jk}},$$

(5)

The function $F$ has to satisfy two constraints such as

- $F$ has to treat word vectors linearly without ambiguity over dimensions, thus, it is written as $F((v^W_{wi} - v^W_{wj})^Tv^C_{wk})$. Since the dimensions are treated uniformly over the period using this property, it maintains the cosine similarity between the vectors.

- The symmetry between context word and target word is artificial and implies that $F$ must be able to exchange both roles.

Hence, the function $F$ is written as

$$F((v^W_{wi} - v^W_{wj})^Tv^C_{wk}) = \frac{F((v^W_{wi})^Tv^C_{wk})}{F((v^W_{wj})^Tv^C_{wk})}$$

(6)

with entails, $F = \exp$ and $P_{ik} = F((v^W_{wi})^Tv^C_{wk}) \iff (v^W_{wi})^Tv^C_{wk} = \log X_{ik} - \log X_i$.

Here, $\log X_i$ is a bias parameter $b^W_i$ to which, $b^C_k$ is added for symmetry. As most of the co-occurrence matrix has zero values, the log is not well defined. To correct this, a weight function $f$ is used in the loss function with the properties below to ensure that it does not overweight small or high values and smoothly cancel the logarithm.

- $f(0) = 0$ and $\lim_{x \to 0} f(x)\log^2(x)$ is finite,

- $f$ should increase,

- $f$ should be small for large values.

Thus, with $V$ as the size of the vocabulary, the loss of GloVe is defined as:
\[ J = \sum_{i,j=1}^{V} f(X_{ij})((v^W_{w_i})^T v^C_{w_j} + b^W_i + b^C_j - \log X_{ij})^2. \]  

The three variants of GloVe pre-trained embeddings used in our work are - **GloVe General**, **GloVe Stack Overflow** trained, and **GloVe Twitter** dataset trained. We have used a general **GloVe** that uses a common crawl with 42B tokens, 1.9M vocabulary, and 300– dimensional vectors. **GloVe Twitter** was a common crawl with 27B tokens, 1.2M vocabulary, and 200– dimensional vectors. We have used Twitter-trained GloVe as it follows the same informal conversation format as our chat transcripts. Another variant of pre-trained GloVe, **GloVe Stack Overflow**, was created by Chatterjee et al. [1] for the answer extraction process of the original ChatEO as described in Section 3.2.4. This was used to check if the domain-specific word embedding have any impact on the performance of the DL models.

### 3.2.7 DL models used in ChatEO

The core part of our research was to modify the opinion-asking question identification process of ChatEO. In the original architecture of ChatEO created by Chatterjee et al. [1], a heuristic-based approach was used to identify the opinion-asking questions. The obvious disadvantage of this rule-based approach is that it requires skilled experts in addition to manual crafting and enhancement of rules all the time. Moreover, the system can become complex that some rules can start contradicting each other. In our research, we have replaced the heuristic approach with DL based models. These models have proven to be better for NLP tasks since Collobert et al. [105] demonstrated that a simple deep learning framework outperforms most state-of-the-art approaches in several NLP tasks. The word embeddings discussed in the Section 3.2.6 will be the features used by the DL models as they help the models to understand the semantic structure better. The DL models used in our research are LSTM, Bi-LSTM, 1-layer CNN, and Multi-layer CNN; we explain each next.
3.2.7.1 LSTM

LSTM is a variant of Recurrent Neural Network (RNN) introduced by Schmidhuber et al. [117] that helps in learning the long-term dependencies that traditional RNN was not able to capture [117, 177]. As LSTM remembers information for long periods as default behaviour, it is explicitly designed to avoid the long-term dependency problem. We explain how LSTM works in this section. All RNN architectures usually have a chain of repeating modules of neural networks such as a simple Tanh layer. In LSTM, this repeating module has a slightly complicated structure of four neural networks specially interacting with each other. Figure 12 illustrates this structure.

The main concept behind LSTM is the cell state which functions as a conveyor belt [13]. It runs through the entire chain and it is easy to pass the information along it, unchanged. LSTM can add or remove information to the cell state using the structure, gates [14]. Gates which help in allowing the information optionally to the cell state are created by the combination Sigmoid neural net layer and a pointwise multiplication operation. The output range of the Sigmoid layer would be between zero and one - ‘nothing passes through’ and ‘everything passes through’, respectively. There are 3 types of gates in the LSTM to control the cell state - forget layer, input layer, and output layer.
Let us understand how LSTM works using the cell state and gates. The first gate of LSTM is the 'forget gate layer'\textsuperscript{15}, a Sigmoid layer that helps us to decide which information to keep in the cell state and which ones to remove. It looks at the hidden state from the previous module, and current input and outputs a number between 0 and 1 to decide whether the part of the old output is necessary (by giving the output closer to 1). The equation for the forget gate $f_t$, as shown in Figure\textsuperscript{15}, contains the timestep $t$, input $x_t$, previous hidden state $h_{t-1}$, weight matrix between forget gate and input gate $W_f$, and the connection bias $b_f$ at $t$. For example, consider a language model where the task is to predict the next word based on all the previous ones. Here, a cell state might store the subject’s gender so that correct pronouns can be used. When a new subject is encountered, we need to forget the gender of the old subject using the forget gate layer.

The next gate layer is the 'input gate layer'\textsuperscript{16} that helps us to decide the new information to be stored in the cell state. The Sigmoid layer decides on the values that need update, whereas the Tanh layer creates a new vector with all the candidates, $\tilde{C}_t$ that can be added to the cell state. These two are combined to update the state. The equations given in Figure\textsuperscript{16} contains the timestep $t$, the input gate $i_t$ at $t$, weight matrix of Sigmoid operator between input and output gate $W_i$, bias vector $b_i$ at $t$, value generated by Tanh $\tilde{C}_t$, weight matrix of Tanh operator between cell state information and network output $W_c$ and bias vector at $t$ w.r.t $W_c$. For instance, in the language model, we need to forget the gender of the old subject for us to update the gender of the new subject.

Now, we can update the old cell state $C_{t-1}$, into a new cell state $C_t$ using the previous step values. The old state is multiplied by $f_t$ to forget the things that were decided to be forgotten and added with $i_t$ and $\tilde{C}_t$ (the candidates along with how of the state values to be updated)\textsuperscript{17}. In the language model example, this is where we
would actually add the new information by forgetting the old subject’s gender.

The final gate is the ’output gate layer’ \([18]\) that helps us to decide the output that is the filtered cell state value. A Sigmoid layer is run first to decide which part of the cell state will be the output. Then the cell state will be subjected to a Tanh layer and multiplied with the Sigmoid output, to get the part of the cell state we decided as the output. The equation as shown in Figure \(18\) for the output gate \(O_t\) contains the timestep \(t\), weight matrix of output gate \(W_o\), bias vector \(b_o\) w.r.t \(W_o\) and the LSTM output \(h_t\). In the example of the language model, the output might be relevant to a verb (maybe singular or plural) as it just saw a subject so that we can understand what form the verb should take if that is the next word.

Finally, the new cell state \(C_t\) and new hidden state \(h_t\) are carried over to the next time step.

### 3.2.7.2 Bi-LSTM

The Bi-LSTM \([19]\) is a Bi-RNN \([118]\) variant introduced by Graves et al. \([178]\) to address the problem of vanishing gradient. Bi-LSTM combines the benefits of long-range memory and bidirectional processing and was proven to be useful in various domains \([179,180]\). It processes the sequence data in both forward and backward directions with two separate hidden layers and generates two independent sequences
of LSTM output vectors. Hence, the output of a BiLSTM at each time step is 
\[ y_t = \sigma(\{\vec{h}_{t}, \hat{h}_{t}\}) \], where \( \vec{h}_{t} \) and \( \hat{h}_{t} \) denote the outputs of two LSTM layers respectively. \( \sigma \) function is used to combine the two output sequences and can be a concatenating function, a summation function, an average function or a multiplication function.

For instance, consider the sentence “The boys went to the...” where the next word needs prediction. In the traditional LSTM, the next word is predicted using only the previous words. In Bi-LSTM, we can predict the next word using the information further down the road as well. The forward pass of Bi-LSTM will have the information “The boys went to the _” while the backward pass will have the information “_ and then they got out of the pool”. So by understanding the information in the future, Bi-LSTM is able to predict the target word more accurately than unidirectional LSTM.
3.2.7.3 CNN

CNN is short for Convolutional Neural Network developed by LeCun et al. [113] for a dataset of handwritten digits based on the idea by Fukushima et al. [181] - aggregating simpler features into more complicated features using complex artificial cells. The basic CNN consist of an array of convolutional layer and pooling layer which is connected to a fully connected layer to get the output. Collobert et al. [130] explored the use of CNN in NLP tasks and a sample CNN for text input is shown in Figure 20.

Convolution layer: The important layer in a CNN is the convolutional layer where the input is converted into a feature map with the help of a filter called kernel. A kernel is a grid of discrete values with initialized weights and is passed through the sentence matrix to get the feature maps. Two key hyperparameters that define the
Figure 21: CNN - kernel operation [10].

Convolution operation are the size and number of kernels. The former is typically $3 \times 3$, but sometimes $5 \times 5$ or $7 \times 7$. The dot product between the input and the kernel is determined, where their corresponding values are multiplied and then summed up to create a single scalar value, calculated concurrently. The whole process is then repeated until no further sliding is possible and explained by a sample in Figure 21. The figure graphically illustrates the primary calculations executed at each step. In this figure, the light green colour represents the $2 \times 2$ kernel, while the light blue colour represents a similar size area of the input. Both are multiplied by matrix dot product; the result after summing up the resulting product values (marked in light orange colour) represents an entry value to the output feature map.

The stride denoted for the selected step size overall vertical or horizontal locations used in the example is 1 and can be chosen according to the specific task. Padding is an operation applied to the input matrix to retain the in-plane dimension of the
Figure 22: CNN - kernel operation with zero padding [11].

input. Usually, zero padding where rows and columns of zeros are added on each side of the input to fit the centre of a kernel on the outermost element and retain the in-plane dimension. Figure 22 explains how this works with an example. Once the zero padding is completed, the feature map creation follows the same calculations as shown in Figure 21. The kernel size here is $3 \times 3$. Without padding, each successive feature map would get smaller after the convolution operation.

**Pooling layer:** The purpose of the pooling layer is to reduce progressively the spatial size of the feature map generated from the previous convolutional layer and to identify important features. There are different types of pooling methods like tree pooling, gated pooling, average pooling, min pooling, max pooling, global average pooling (GAP), and global max pooling. The common pooling operations are illustrated with an example in Figure 23. Let us consider the smaller matrix highlighted (with 4 values) in the input. Average Pooling takes the average of the 4 values and gives us 12 as a rounded-up result. Max Pooling takes the maximum value out of the 4 and results in 25. Global Average Pooling outputs the rounded-up value, 16, which is the average of all the values in the input.

**Non-linear Activation layer:** In CNN, a non-linear activation layer is applied after each convolution layer and the final fully connected layer. These layers give the CNN the ability to learn extra-complicated things. The activation function must also have the ability to differentiate, which is an extremely significant feature, as it allows error back-propagation to be used to train the network. The most common activation layers used are as follows:
Figure 23: CNN - common pooling operations [10].

- **Sigmoid**: It takes real numbers as input and outputs a value between 0 and 1. It can be represented mathematically by 
  \[ S(z) = \frac{1}{1 + e^{-z}}. \]

- **Tanh**: It is similar to Sigmoid but only outputs a value between -1 and 1. It is mathematically represented as
  \[ \text{Tanh}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}. \]

- **ReLU [182]**: ReLU is short for Rectified Linear Units and is most commonly used activation function for CNN. It avoids and rectifies vanishing gradient problem and is less computationally expensive than Tanh and Sigmoid. It is mathematically expressed as
  \[ \text{ReLU}(z) = \begin{cases} 
  1 & \text{if } z > 0 \\ 
  0 & \text{if } z \leq 0 
  \end{cases}. \]

- **LeakyReLU [183]**: LeakyReLU is a variant of ReLU. Instead of being 0 when \( z < 0 \), a leaky ReLU allows a small, non-zero, constant gradient \( \alpha \) (Normally, \( \alpha = 0.01 \)). It is expressed mathematically as
  \[ \text{LReLU}(z) = \begin{cases} 
  z & \text{if } z > 0 \\ 
  \alpha z & \text{if } z \leq 0 
  \end{cases}. \]

- **Parametric ReLU [184]**: Unlike Leaky ReLU were the slope for the negative inputs is fixed, parametric ReLUs takes the slope \( \alpha \) as a parameter that the neural network should figure out itself through gradient descent for a small cost. The mathematical equation is given by
  \[ f(z_i) = \begin{cases} 
  z_i, & \text{if } z_i > 0 \\ 
  \alpha_i z_i, & \text{otherwise} 
  \end{cases}. \]
  The subscript \( i \) indicates that we allow the nonlinear activation to vary on different channels.

**Fully connected layer**: It is a form of feed-forward neural network and is usually the
last layer of a CNN. Here, each neuron is connected to all the neurons of the previous layers and act as the CNN classifier. The input for the first fully connected layer is the output from the final pooling or convolutional layer and will be flattened, i.e., transformed one-dimensional (1D) array of numbers (or vector) as shown in Figure 24. The final fully connected layer typically has the same number of output nodes as the number of classes as it is connected to a Softmax or Sigmoid function.

The CNN variants in our research include both a single-layer CNN that has only one convolutional layer and a Multi-layer CNN with more than 1 convolutional layer. In our work, we have performed a hyper-parameter tuning on all the models using a scalable hyper-parameter optimization framework, KerasTuner [185]. The Hyperband algorithm [186] is employed from Kerastuner that uses adaptive resource allocation and early-stopping to quickly converge on a high-performing model using a sports championship style bracket. The models were run on a Linux server with 16 vCPU cores and 32 GB of RAM. Google Colab was also used to run the model in parallel when the word embeddings are smaller. All model combinations were run on random 70/30 train/test data split to avoid overfitting by ensuring that the trained model generalizes to unseen data.

3.3 Snorkel for Weak Supervision Labelling

The disentangled dataset explained in Section 3.1 is unlabelled. To train our DL models of the modified ChatEO, we have manually labelled 2,000 data points out of the original dataset. The manual labelling process is explained in Section 4. In practice, DL model performance is directly proportional to the volume of training data [187][188]. As the manual labelling process is expensive, we opted to use a weak
52

learner, Snorkel [13] to automatically label a much larger dataset with ’ ∼ 45k’ data points for DL model training. A weak learner uses noisy, limited or imprecise sources to provide supervision for labelling a large training data [189,190]. We explain how Snorkel works in labelling the training data in Section 3.3.1. With labelling a much larger dataset with Snorkel, it is highly possible to encounter class imbalance. To rectify this issue, we have resorted to class balancing techniques such as Near-Miss and SMOTE and they are explained in Section 3.3.2.

3.3.1 Snorkel Architecture

Snorkel was created on the concept of “Data Programming” to address the challenges of data labelling through “modelling multiple label sources without access to ground truth, and generating probabilistic training labels representing the lineage of the individual labels” [13]. It is a Python package which allows the rules to be written by domain experts that are used to automatically label the training samples. Snorkel is an effective labelling tool for the following reasons:

- Usage of different weak learning sources for labelling.
- The output of Snorkel is probabilistic labels that are used to train general classifier to look beyond the noisy labels.
- User can interact and supervise the system at any time in the process.

Snorkel architecture is illustrated in Figure 25 and consists of 3 stages - writing labelling functions, modelling accuracies and correlation and training a discriminative model. We explain how each stage works next.

Writing labelling functions: We can use different kinds of weak supervision such as patterns, heuristics, external knowledge bases, crowd-sourced labels, etc. in Snorkel. To enable flexible implementation from all sources, Snorkel allows users to write the Labelling Functions (LF) using custom functions (written in Python) and declarative operations. LFs usually is a Python function using depicts extract-transform-load scripts expressing basic heuristics or patterns. Snorkel also includes declarative operators for common weak supervision methods such as:
• Pattern-based: This technique try to capture the labelling patterns identified by subject matter experts using regular expression.

• Heuristics: Heuristics are rules of thump that can be identified by exploring the data. It can be used as LF when expressed as a function.

• Keyword Lookup: Easiest LF would keyword lookup when trying to assign labels based on certain keywords chosen by subject matter experts.

• Distant supervision: By aligning the data heuristics with an external data source, distant supervision will facilitate weak labelling.

• Weak classifiers: Classifiers with limited coverage, noisy, biased, and/or trained on a different dataset can be used as an LF.

• Labelling function generators: This combines multiple LF from various weak supervision sources as a single resource.

• Complex preprocessor: Some LFs might use the derived field in data rather than using the raw data. Preprocessors can be used with LFs to get the derived fields. For instance, Snorkel provides SpacyPreprocessor for NLP tasks.

**Modelling accuracies and correlation:** Once the LFs are created, Snorkel applies them to the unlabelled dataset to produce a matrix of label output. Snorkel has the ability to decide whether to apply a majority vote to determine the label, or model the accuracies in the generative model. The decision is made based on the label matrix density (mean number of non-abstention labels per data point) and an optimizer. As for the density parameter, the generative model performs best with
medium density, while the majority vote gives the best results with low and high density. However, this criterion was not sufficient, so another optimization rule was introduced that is based on “the ratio of positive to negative labels for each data point”. Snorkel’s generative model estimates the accuracies, correlation structures, and other statistical dependencies of these sources, without access to ground truth. The labels attached to data points are probabilistic labels that are supposed to be used to train a discriminative model.

**Training a discriminative model:** The ultimate goal of Snorkel is to train a model that looks beyond the labelling function information and generalizes. The probabilistic labels from the previous step are used to train the discriminative model by reducing the noise-aware variant of the loss function. The more unlabeled data we train with Snorkel, the better the predictive performance of the discriminatory model. The same happens when we increase the amount of hand-labelled data in training traditional models.

**Snorkel for our research:** We have used Snorkel to label a larger dataset to be used by the DL models. For the labelling functions, we have modified the heuristics created for the original ChatEO [1] based on the analysis of the larger dataset. We have also added a few additional heuristics based on the analysis. The modified/new rules of LFs are explained in Chapter 5.

### 3.3.2 Class balancing (after Snorkel)

Opinion-asking questions in chat transcripts are in abundance compared to other developer communication [1,21,22]. But they are still less in number (≈ 10%) compared to the overall volume of our larger dataset as there can be other kinds of details such as facts, error debugging methods, etc. in chat transcripts. This scenario classifies the opinion-asking questions in our dataset as a minority class. This uneven class distribution in the dataset can result in poor performance in DL models [191] as they favour the majority class which is not important in our research. To avoid this bias in learning, we have opted for SMOTE [155] and Near-Miss [192] class balancing techniques to balance our dataset. We present how both these techniques work next.
3.3.2.1 SMOTE

SMOTE is short for Synthetic Minority Over-sampling Technique created by Chawla et al. [155] where the minority class is over-sampled by creating “synthetic” examples. These samples are created by operating in feature space rather than data space and thereby are less application-oriented. The steps of the SMOTE algorithm are as follows:

- Take a random sample from the minority class.
- Identify the $k$-nearest neighbors of this sample.
- Identify the vector between the chosen sample and the one of its nearest neighbors.
- Multiple this vector with a random number between 0 and 1.
- Add this to new vector to the chosen sample to get a synthetic data point.

The method applied here [26] is moving the data point slightly in the direction of its neighbour. This way, the synthetic data created is not exactly a copy of the minority sample but also not too different from it.

3.3.2.2 Near-Miss

Near-Miss is an undersampling balancing technique, and it works by randomly eliminating the samples from the majority class by looking at the class distribution [192]. The basic idea is that the algorithm eliminates the data points of the majority class
if it is near to a minority sample. The steps followed by the Near-Miss algorithm are the following:

- Calculate the distance between all the majority samples with the points in minority samples.
- Select the data points in majority class with a shorter distance with the minority data point as \( n \) instances.
- If there are \( k \) instances in the minority class, return only \( k \cdot n \) instances of the majority class.

There are three versions in Near-Miss:

**Near-Miss Version 1:** As shown in Figure 27, it selects samples of the majority class for which average distances to the \( k \) closest instances of the minority class are the smallest.

**Near-Miss Version 2:** As illustrated in Figure 28, it selects samples of the majority class for which average distances to the \( k \) farthest instances of the minority class are the smallest.

**Near-Miss Version 3:** As presented in Figure 29, it works in two steps. Firstly, for each minority class instance, their \( M \) nearest neighbours are stored. Then finally, the majority class instances are selected for which the average distance to the \( N \) nearest neighbors is the largest.

### 3.4 Evaluation Metrics

Evaluating the deep learning models to compare and find the best performing one for each specific task is an important process in any research work. In our work, we employ the widely used evaluation metrics for machine learning - precision, recall, and F-measure.

- **Precision** is defined as the fraction of relevant retrieved items out of all retrieved items. In our research, we use precision to analyze whether the automatically identified instances are indeed opinion-asking questions by the ratio of
true positives over the sum of true and false positives. The equation is defined as:
\[
P = \frac{|\text{relevant items} \cap \text{retrieved items}|}{|\text{retrieved items}|} \tag{8}
\]

- **Recall** is also known as the true positive rate, the fraction of relevant retrieved items out of all the relevant items. To evaluate how often a technique fails to identify an opinion-asking question, we use recall, the ratio of true positives over the sum of true positives and false negatives. The equation is given by
\[
R = \frac{|\text{relevant items} \cap \text{retrieved items}|}{|\text{relevant items}|} \tag{9}
\]

- **F-measure** is the harmonic mean of both precision and recall and can be represented as:
\[
\frac{2 \cdot P \cdot R}{P + R} \tag{10}
\]
Chapter 4

Results

This section presents the results and answers for the research questions identified in Chapter 1 one by one along with their analysis.

4.1 Impact of Deep Learning Models in ChatEO

In our work, we have replaced ChatEO’s question identification heuristics with a deep learning-based approach. Before evaluating the DL model’s performance in ChatEO, we discuss the dataset metrics followed by the heuristic’s performance on the dataset as a baseline.

4.1.1 Discord and Slack datasets

As discussed in Chapter 3 we have published a new dataset on disentangled Discord chat conversations \(^{38}\) to complement the Slack dataset created by Chatterjee et al. \(^{22}\). These two datasets are used together in our research as the source. Metrics including the number of conversations, utterances and users of the overall dataset are presented in Table 5.

The comparison in the total data volume for each of the three metrics between Slack and Discord is illustrated in Figure 30. We can notice that the number of users and utterances in Discord for just one year is higher than the number of Slack users and utterances of two years.
Table 5: Slack and Discord dataset metrics.

<table>
<thead>
<tr>
<th>Channel</th>
<th>Duration</th>
<th>#Conversations</th>
<th>#Utterances</th>
<th>#Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>python (Discord)</td>
<td>Nov2019–Oct2020</td>
<td>19,155</td>
<td>1,254,362</td>
<td>300,919</td>
</tr>
<tr>
<td>gophers (Discord)</td>
<td>Nov2019–Sep2020</td>
<td>8,860</td>
<td>247,179</td>
<td>19,983</td>
</tr>
<tr>
<td>racket (Discord)</td>
<td>Nov2019–Oct2020</td>
<td>538</td>
<td>4,975</td>
<td>917</td>
</tr>
<tr>
<td>clojurians (Discord)</td>
<td>Nov2019–Oct2020</td>
<td>159</td>
<td>1,577</td>
<td>1,743</td>
</tr>
<tr>
<td>clojurians (Slack)</td>
<td>Jul2017–Jun2019</td>
<td>7,918</td>
<td>72,973</td>
<td>2,422</td>
</tr>
<tr>
<td>elmlang (Slack)</td>
<td>Jul2017–Jun2019</td>
<td>22,150</td>
<td>258,658</td>
<td>6,454</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>67,667</td>
<td>19,45,986</td>
<td>3,35,733</td>
</tr>
</tbody>
</table>

This confirms the trend that the software development community is slowly migrating toward Discord in recent years [160, 161]. Even though the volume of conversations in Discord is lower than for Slack, it is likely to match or overtake Slack in a few months. To check the validity of this claim, let us investigate the quarterly conversation trends for all the channels in Discord in Figure 31.

From this figure, we can observe that Discord conversation numbers are growing steadily for each of the channels supporting our claim. We also know that there is a 10,000 cap on messages in Slack which leads to the purging of conversations and in turn valuable information. These points along with the public availability of the Discord chat transcripts without admin permissions strengthen the choice to combine Discord with Slack as our data source.
4.1.2 Evaluation dataset

To train our DL models for opinion-asking question identification in this research as discussed in Section 3.2, we need a labelled dataset with two classes — *opinion asking questions* and *non-opinion asking questions*. As shown in Table 5, the resulting dataset after combining Discord and Slack chat transcripts is quite large. In the original paper [1], ChatEO’s heuristic-based question identification part uses a dataset of 400 conversations. As we would need a larger dataset to train our DL models, we randomly choose 2,000 conversations combining both Slack and Discord as our evaluation dataset.

Two annotators with experience in both Slack and Discord platforms have labelled the evaluation dataset. Following the guidelines for software engineering studies [165], two annotators first individually labelled a shared set of 400 conversations which represents 20% of 2,000 conversations. We computed Cohen’s Kappa inter-rater agreement between the two annotators, and found an agreement of 0.72, which is considered as *sufficient* (> 0.6) [193]. Then, the annotators split the remaining conversations and labelled them individually; the resulting labelled evaluation dataset consists of 2,000 conversation. The popular two channels from both Slack and Discord are selected to create the dataset - *pythondev#help, clojurians#clojure, python#python−general, gophers#golang*. We made sure to have a balanced dataset between the two classes since it is considered a good practice for evaluating DL model performance [154,191].
4.1.3 ChatEO’s heuristics model as baseline

Our study focuses on improving ChatEO’s question identification part [1] by using deep learning models. To compare the improvements, we ran the original ChatEO’s question identification script on our new labelled evaluation dataset of 2,000 conversations, a subset of the dataset explained in Section 3.1. This is considered as our baseline. The metrics used for evaluation are precision, recall, and F-measure as discussed in Section 3.4. Table 6 presents the after-run results.

As we see from Table 6, the F-measure of 0.62 for the combined evaluation data aligns with the original ChatEO results presented by Chatterjee et al. [1]. The values are comparable as we have not changed the model in the baseline. The precision (0.66) is a little lower than the original (0.87), stating that the number of false positives is higher. The recall is slightly higher (0.59) than the original model (0.49). Both these values mean that the heuristics-based model is misclassifying instances as opinions when they are not and vice versa. This can happen since heuristics are not extensive, and not every instance can fit into the predefined rules. One such example is ‘Hey guys where’s the best place typically to add a random Seed for a pkg?’. As heuristics patterns used in the model are based on linguistics rules and keywords, this instance was identified as an opinion given the keyword ‘best’ and the pattern starting with ‘where’ is described in the original paper. This is one of the reasons to replace the heuristic-based question identification part of ChatEO with DL models as DL models can learn the patterns on their own via training.

4.1.4 DL in ChatEO for opinion-asking question identification

The main part of our research is to improve the question identification part of ChatEO as explained in Section 3.2.5. The training source dataset used for the DL models is
Table 7: Opinion-asking question identification results: DL models.

<table>
<thead>
<tr>
<th>Word Embeddings</th>
<th>DL Models</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word2Vec_SO</td>
<td>LSTM</td>
<td>0.78</td>
<td>0.75</td>
<td>0.76</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td>0.80</td>
<td>0.79</td>
<td>0.80</td>
</tr>
<tr>
<td></td>
<td>1-CNN</td>
<td>0.81</td>
<td>0.72</td>
<td>0.77</td>
</tr>
<tr>
<td></td>
<td>Multi-CNN</td>
<td>0.86</td>
<td>0.75</td>
<td>0.80</td>
</tr>
<tr>
<td>Word2Vec_General</td>
<td>LSTM</td>
<td>0.62</td>
<td>0.75</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td>0.79</td>
<td>0.79</td>
<td>0.79</td>
</tr>
<tr>
<td></td>
<td>1-CNN</td>
<td>0.83</td>
<td>0.68</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>Multi-CNN</td>
<td>0.85</td>
<td>0.76</td>
<td>0.80</td>
</tr>
<tr>
<td>GloVe_General</td>
<td>LSTM</td>
<td>0.70</td>
<td>0.70</td>
<td>0.74</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td>0.79</td>
<td>0.84</td>
<td>0.81</td>
</tr>
<tr>
<td></td>
<td>1-CNN</td>
<td>0.74</td>
<td>0.82</td>
<td>0.78</td>
</tr>
<tr>
<td></td>
<td>Multi-CNN</td>
<td>0.80</td>
<td>0.82</td>
<td>0.81</td>
</tr>
<tr>
<td>GloVe.SO</td>
<td>LSTM</td>
<td>0.83</td>
<td>0.59</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td>0.86</td>
<td>0.68</td>
<td>0.76</td>
</tr>
<tr>
<td></td>
<td>1-CNN</td>
<td>0.85</td>
<td>0.75</td>
<td>0.80</td>
</tr>
<tr>
<td></td>
<td>Multi-CNN</td>
<td>0.87</td>
<td>0.81</td>
<td>0.84</td>
</tr>
<tr>
<td>GloVe_Twitter</td>
<td>LSTM</td>
<td>0.75</td>
<td>0.59</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td>0.73</td>
<td>0.76</td>
<td>0.74</td>
</tr>
<tr>
<td></td>
<td>1-CNN</td>
<td>0.73</td>
<td>0.81</td>
<td>0.77</td>
</tr>
<tr>
<td></td>
<td>Multi-CNN</td>
<td>0.83</td>
<td>0.85</td>
<td>0.84</td>
</tr>
</tbody>
</table>

The DL models used in our evaluation are LSTM, Bi-LSTM, 1-layer CNN, and Multi-layer CNN discussed in Section 3.2.7. The features used for the DL models are the word embedding results. We have used both Word2Vec and GloVe embeddings as discussed in Section 3.2.6. Thus, there are a total of 20 combination and DL models.
and embeddings that we ran in our evaluation study. The results are reported in Table 7.

Table 7 demonstrates that the results with DL models are better than the ones of the heuristics-based model. We note that the only features the DL models used are the word embeddings. From Table 7, multi-CNN performs best in all three metrics. This is because CNN can extract an area of features from global information and can consider the relationship among these features, unlike LSTM networks that consider the outputs only on the previous state. RNNs are usually good at predicting what comes next in a sequence, while CNNs can learn to classify a sentence or a paragraph as they process all the words in a text carefully and assign weights to each word in an unbiased manner. Another advantage of CNN over RNN is the faster execution time for training and testing. Multi-layer CNN also outperforms 1-CNN across all the embeddings. This is because, as the network grows deeper, the model seems to result in better generalization for a wide variety of tasks.

We could also see that GloVe SO and GloVe Twitter perform better than other embeddings. The advantage of GloVe is that, unlike Word2Vec, GloVe does not rely just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence) to obtain word vectors. This combined with the CNN that extracts global information as the features is the main reason for the good performance. We would like to note that GloVe Twitter performs better overall than even a software-related pre-trained embedding, i.e., GloVe SO. One reason could be because Twitter data resembles the chat transcripts more for its informal nature and short length than Stack Overflow data which contains more technical terms.

4.1.5 Quality analysis: heuristics vs. DL models

As we could see from Tables 6 and Table 7, DL models performed better than the heuristics-based baseline in all the three metrics. To interpret this result, we have performed a quality analysis of both approaches. To do so, we have manually identified all the instances from the evaluation dataset that were incorrectly classified by the heuristics but correctly classified by DL models in our research. These instances are then analysed to understand the trends and patterns that constitute the incorrect classification by the heuristics-based approach. They are discussed in this
section along with certain examples.

1. Heuristics follow the pre-defined linguistic rules without understanding the proper context. This leads to the misclassification of the instances in most of the cases analyzed. For instance,

- 'Does anyone know how to install a package from a repo branch other than main?' - this non-opinion question is classified as an opinion by the heuristics-based technique as it fits the predefined pattern starting with 'Does anyone know' without considering the context into the picture.
- 'Is it fine to leave a lot of variables leaving around?' - this opinion-asking question was incorrectly classified by the heuristics-based baseline as even though the pattern starting as 'is it' is defined, the positive adjective keyword 'fine' was not included.

2. When the word 'Way' is used, the heuristics-based approach identifies it only as an opinion-asking question. This is because heuristics patterns treat the keyword as mostly used to ask for opinions. But it can represent either opinion or not. For instance,

- 'Is there a way to auto-serialize a plain old data struct?' - is not an opinion-asking question but is incorrectly classified by the heuristics-based technique due to the presence of the keyword 'way'.
- 'Hey guys, the best way to convert 2d image to 3d using python is?' - is an opinion-asking question that was correctly identified by the heuristics-based tool. Even though it was identified correctly in this case, the heuristics still do not consider the context of the sentence.

3. Heuristics do not consider whether the spelling is different than what is expected in the rules.

- 'Hi, can anyone **recommended** me a tutorial for beginners?' - This would have been correctly identified by the heuristics as an opinion if the 'recommended' keyword was correctly spelled.

4. Short forms are not always captured by the heuristics unless and until specified explicitly.
- 'What’s the best ide for Linux?’ - This opinion-asking question was not captured by the heuristics because of the use of the short form.

5. Most of the coding questions are labelled as non-opinion by the heuristics even if they are not. This is because heuristics do not cover code within the question.

- ‘Is ‘if name == main’ a good pattern?’ - This opinion-asking question was not captured by the heuristics because of a code line inserted within the question.

6. Heuristics are not extensive as the dataset grows in size. This is because the patterns and keywords written by analyzing some data points cannot capture all the instances of the data.

- ‘Isn’t declaring a function inside a function the same as playing with classes?’ - A negative pattern question was not captured as it was not part of the pre-defined rules.

- ‘Why is Pycharm better? I’m confused about what makes IDE’s better rather than a VS Code editor.’ - This was not captured as an opinion as the pattern starting with ‘why’ was not part of the heuristics

- ‘Hi guys! General quick question. How do you guys have your project structure setup when you need multiple custom error classes?’ - This was not captured as an opinion as the pattern starting with ‘how’ was not part of the heuristics.

7. There are different ways to ask the same question as the chat transcripts are informal and do not always follow grammar rules followed by heuristics.

- ‘Experimenting with a collections cheat sheet diagram kind of thing. What all of you think?’

- ‘How is the best way to test a python Discord bot?’

DL models were able to classify all of these instances correctly which demonstrates a better performance showcased in Table 7. DL models can learn the patterns of identifying an opinion in questions by automatically learning the training data. The more the training data, the more the accurate learning [187,188]. This motivated us
to acquire a larger training dataset with more than 2,000 instances as described in Section 2.5.1.

**Answer to RQ1**: The findings of this thesis show that deep learning models perform better than the heuristics-based approach of ChatEO in identifying opinion-asking questions. It also shows that Multi-CNN performed the best overall due to its deep convolutional architecture when combined with GloVe-Twitter. We could understand the better performance of GloVe-Twitter word embedding over other software-engineering trained embeddings as Twitter texts are short and informal similar to chat transcripts. The quality analysis confirmed the better performance of DL models over the heuristics-based baseline in various scenarios.

### 4.2 Impact of the Dataset Size on DL Models

As discussed in Section 3.3 to increase the size of the labelled evaluation set, we have used a weak supervision learner, Snorkel [13]. Snorkel combines the labelling functions written using different weak supervision sources and models the correlation and accuracies between them before training a discriminative model to label the source data. Before labelling, a larger dataset is created as a subset of the combined Slack+Discord dataset created in Section 3.1. As the 2,000 instances dataset created in Section 4.1 was created from the channels - pythondev#help, clojurians#clojure, python#python – general, gophers#golang, we consider all the instances from these four channels to create our larger dataset. As discussed in Section 4.1.4, only the utterances that initiate a conversation and belong to a single speaker were considered while creating the dataset.

Snorkel labelling function to label this larger dataset was based on the improvement and addition to the heuristics defined by Chatterjee et al. [1]. This improvement and additions are made by analysing this much larger dataset randomly, as well as by using some of the results identified in Section 4.1.5. The changes are mentioned below.

- The original patterns are modified to add new keywords such as to cover the new data instances. Some examples of new keywords are ‘build’ (action verb)
and *feedback* (*noun*).

• Excluded certain keywords from the original patterns. For instance, we have excluded the keyword *fix* which is often associated with code debugging.

• Addition of new rules.
  
  – Different way to ask which path to take among several paths. Example: the pattern “Could I.. or..” was added.
  
  – A pattern starting with “How” followed by a positive adjective was added. Example: “How to best avoid longer code running time?” or “How effective would that method be?”
  
  – A pattern starting with “what do you think” is added. Example: “What do you think about static class methods?”
  
  – A pattern starting with “Will it be difficult” is added. Example: “Will it be difficult to convert my code from R to python?”
  
  – A pattern that includes ‘if’ followed by a positive adjective is added. Example: “I want to know if the used method is good?”
  
  – A pattern for negative sentences for opinion-asking is added. Example: “Why don’t/shouldn’t we do it in that manner?”

• Addition of new sub-patterns in the same rule of the heuristics. For instance, a new sub-pattern was added to the rule starting is ‘what/which’ to include the short forms like *what’s*.

• Removal of a rule from the original heuristics. For example, the rule on *Where to find X* was removed as it was indicating a URL or web location and not representing an opinion.

After running Snorkel on these enhanced heuristics rules, we have labelled all the 44,820 instances in our larger dataset. The labelled dataset has the class proportion as illustrated in Figure. 32. We can see that the number of positive classes (i.e., the opinion asking questions) is much smaller than the number of non-opinion questions.
4.2.1 Quality assessment of the Snorkel-generated labels

To assess the quality of the labels generated by Snorkel, we have conducted a qualitative study. We randomly picked 2,241 (5%) of the 44,820 instances and manually labelled them as opinion or non-opinion. Then, a micro-averaged F-Score was calculated to compare Snorkel labels and our manual labelling. The score came as 0.84 proving the good quality of the Snorkel automatic labelling [77].

We understand that balanced instances provide better results compared to unbalanced dataset [191]. To validate this, we ran the DL models with different embeddings on our larger unbalanced dataset that was discussed in Section 3.2.7 and Section 4.1.4. The results are reported in Table 8. The results demonstrate that on the larger dataset the DL models performed worse than when trained on the dataset of 2,000 instances (as reported in Table 7). This is because even though the dataset of 2,000 instances is smaller, it is balanced and hence, the better results. The highest values of the metrics for the larger dataset are only in the ranges of 60−70s compared to 80s on a much smaller dataset. This shows that class balance plays an important role when it comes to machine/deep learning models. Even though the metric values are lower, we could still see that GloVe embeddings and Multi-CNN performed better than other models/embeddings. This highlights the importance of global feature identification and correlation as discussed in Section 4.1.4.
Table 8: Opinion-asking question identification results: DL + larger dataset.

<table>
<thead>
<tr>
<th>Word Embeddings</th>
<th>DL Models</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word2Vec_SO</td>
<td>LSTM</td>
<td>0.62</td>
<td>0.58</td>
<td>0.54</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td>0.67</td>
<td>0.62</td>
<td>0.64</td>
</tr>
<tr>
<td></td>
<td>1-CNN</td>
<td>0.67</td>
<td>0.44</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>Multi-CNN</td>
<td>0.68</td>
<td>0.63</td>
<td>0.65</td>
</tr>
<tr>
<td>Word2Vec_General</td>
<td>LSTM</td>
<td>0.58</td>
<td>0.56</td>
<td>0.57</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td>0.66</td>
<td>0.60</td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td>1-CNN</td>
<td>0.59</td>
<td>0.60</td>
<td>0.59</td>
</tr>
<tr>
<td></td>
<td>Multi-CNN</td>
<td>0.67</td>
<td>0.61</td>
<td>0.64</td>
</tr>
<tr>
<td>GloVe_General</td>
<td>LSTM</td>
<td>0.67</td>
<td>0.52</td>
<td>0.59</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td>0.64</td>
<td>0.61</td>
<td>0.62</td>
</tr>
<tr>
<td></td>
<td>1-CNN</td>
<td>0.64</td>
<td>0.46</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>Multi-CNN</td>
<td>0.69</td>
<td>0.65</td>
<td>0.64</td>
</tr>
<tr>
<td>GloVe_SO</td>
<td>LSTM</td>
<td>0.61</td>
<td>0.62</td>
<td>0.61</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td>0.63</td>
<td>0.63</td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td>1-CNN</td>
<td>0.62</td>
<td>0.46</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>Multi-CNN</td>
<td>0.69</td>
<td>0.61</td>
<td>0.65</td>
</tr>
<tr>
<td>GloVe_Twitter</td>
<td>LSTM</td>
<td>0.65</td>
<td>0.38</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td>0.67</td>
<td>0.45</td>
<td>0.54</td>
</tr>
<tr>
<td></td>
<td>1-CNN</td>
<td>0.57</td>
<td>0.36</td>
<td>0.44</td>
</tr>
<tr>
<td></td>
<td>Multi-CNN</td>
<td><strong>0.70</strong></td>
<td>0.64</td>
<td><strong>0.67</strong></td>
</tr>
</tbody>
</table>

4.3 Impact of class balancing on DL models

Our larger evaluation dataset is highly imbalanced with just 10% of instances being opinion-asking questions. From the results of Section 4.1.4 and Section 4.2, we understand the importance of class balancing when working with deep learning models. To tackle the class imbalance problem of our dataset, we leverage the class balancing techniques such Near-Miss and SMOTE as explained in Section 3.3.2 and discuss the results in this section.
4.3.1 Near-Miss sampling results

Near-Miss [14] is an under-sampling technique developed to balance the imbalance dataset by eliminating the majority class. We have applied the Near-Miss version 1 to our larger imbalanced dataset after Snorkel labelling. The class proportion for 4,472 instances per class after applying Near-Miss is illustrated in Figure 33. This gives a larger balanced dataset than the initial 2,000 instance evaluation dataset.

This balanced dataset is then used to run the DL models with different embeddings that are discussed in Section 3.2.7 and Section 4.1.4. The results are reported in Table 9. As we can see, the performance of the DL models have increased considerably from the imbalanced larger dataset proving the efficiency of the balanced data. In recall, GloVe Twitter comes on top with 0.9. It also achieves the highest value in F-measure along with GloVe General. In precision, GloVe general performs best. In terms of DL models, Multi-CNN scores the highest across all the embeddings and metrics for its powerful convolutional architecture.

4.3.2 SMOTE sampling results

Near-Miss under-sampling offered better results overall but the resultant source dataset is still comparatively small than the full larger dataset. To increase the size of the dataset and keep it balanced, we have applied a SMOTE oversampling technique on the larger dataset. SMOTE [155] was developed as an oversampling technique that creates synthetic data points of the minority class to balance the dataset. The class proportions after applying SMOTE is presented in Figure 34 which is the largest balanced dataset with 39,650 instances each used in this study.
This SMOTE-based balanced dataset is then used to run the DL models with different embeddings that are discussed in Section 3.2.7 and Section 4.1.4. The results are provided in Table 10. We can see from the table that SMOTE performs the best overall in all our experiments. The recall score reaches 0.95 with GloVe General. F-measure is also at its highest at 0.95 with GloVe General and GloVe Twitter. In precision, GloVe Twitter scores the highest with 0.97. All of the highest results belong to Multi-CNN. These results present the evidence that large and balanced dataset yields the best performance of DL models.

<table>
<thead>
<tr>
<th>Word Embeddings</th>
<th>DL Models</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word2Vec_SO</td>
<td>LSTM</td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
</tr>
<tr>
<td>Word2Vec_SO</td>
<td>Bi-LSTM</td>
<td>0.84</td>
<td>0.87</td>
<td>0.85</td>
</tr>
<tr>
<td>Word2Vec_SO</td>
<td>1-CNN</td>
<td>0.85</td>
<td>0.84</td>
<td>0.84</td>
</tr>
<tr>
<td>Word2Vec_SO</td>
<td>Multi-CNN</td>
<td>0.82</td>
<td>0.89</td>
<td>0.85</td>
</tr>
<tr>
<td>Word2Vec General</td>
<td>LSTM</td>
<td>0.84</td>
<td>0.75</td>
<td>0.79</td>
</tr>
<tr>
<td>Word2Vec General</td>
<td>Bi-LSTM</td>
<td>0.85</td>
<td>0.88</td>
<td>0.86</td>
</tr>
<tr>
<td>Word2Vec General</td>
<td>1-CNN</td>
<td>0.82</td>
<td>0.83</td>
<td>0.85</td>
</tr>
<tr>
<td>Word2Vec General</td>
<td>Multi-CNN</td>
<td>0.85</td>
<td>0.86</td>
<td>0.84</td>
</tr>
<tr>
<td>GloVe_General</td>
<td>LSTM</td>
<td>0.79</td>
<td>0.79</td>
<td>0.79</td>
</tr>
<tr>
<td>GloVe_General</td>
<td>Bi-LSTM</td>
<td>0.85</td>
<td>0.89</td>
<td>0.87</td>
</tr>
<tr>
<td>GloVe_General</td>
<td>1-CNN</td>
<td>0.85</td>
<td>0.86</td>
<td>0.85</td>
</tr>
<tr>
<td>GloVe_General</td>
<td>Multi-CNN</td>
<td>0.86</td>
<td>0.87</td>
<td>0.86</td>
</tr>
<tr>
<td>GloVe_SO</td>
<td>LSTM</td>
<td>0.72</td>
<td>0.80</td>
<td>0.76</td>
</tr>
<tr>
<td>GloVe_SO</td>
<td>Bi-LSTM</td>
<td>0.82</td>
<td>0.82</td>
<td>0.82</td>
</tr>
<tr>
<td>GloVe_SO</td>
<td>1-CNN</td>
<td>0.81</td>
<td>0.85</td>
<td>0.83</td>
</tr>
<tr>
<td>GloVe_SO</td>
<td>Multi-CNN</td>
<td>0.83</td>
<td>0.88</td>
<td>0.85</td>
</tr>
<tr>
<td>GloVe_Twitter</td>
<td>LSTM</td>
<td>0.81</td>
<td>0.78</td>
<td>0.79</td>
</tr>
<tr>
<td>GloVe_Twitter</td>
<td>Bi-LSTM</td>
<td>0.83</td>
<td>0.78</td>
<td>0.80</td>
</tr>
<tr>
<td>GloVe_Twitter</td>
<td>1-CNN</td>
<td>0.80</td>
<td>0.79</td>
<td>0.79</td>
</tr>
<tr>
<td>GloVe_Twitter</td>
<td>Multi-CNN</td>
<td>0.81</td>
<td>0.90</td>
<td>0.87</td>
</tr>
</tbody>
</table>
Table 10: Opinion-asking question identification results: DL + SMOTE dataset.

<table>
<thead>
<tr>
<th>Word Embeddings</th>
<th>DL Models</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word2Vec_SO</td>
<td>LSTM</td>
<td>0.92</td>
<td>0.90</td>
<td>0.91</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td>0.93</td>
<td>0.94</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>1-CNN</td>
<td>0.91</td>
<td>0.92</td>
<td>0.91</td>
</tr>
<tr>
<td></td>
<td>Multi-CNN</td>
<td>0.93</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>Word2Vec_General</td>
<td>LSTM</td>
<td>0.93</td>
<td>0.91</td>
<td>0.92</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td>0.93</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>1-CNN</td>
<td>0.93</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>Multi-CNN</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>GloVe_General</td>
<td>LSTM</td>
<td>0.90</td>
<td>0.93</td>
<td>0.91</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td>0.93</td>
<td>0.94</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>1-CNN</td>
<td>0.92</td>
<td>0.94</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>Multi-CNN</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>GloVe_SO</td>
<td>LSTM</td>
<td>0.92</td>
<td>0.92</td>
<td>0.92</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td>0.93</td>
<td>0.92</td>
<td>0.92</td>
</tr>
<tr>
<td></td>
<td>1-CNN</td>
<td>0.93</td>
<td>0.92</td>
<td>0.92</td>
</tr>
<tr>
<td></td>
<td>Multi-CNN</td>
<td>0.96</td>
<td>0.92</td>
<td>0.94</td>
</tr>
<tr>
<td>GloVe_Twitter</td>
<td>LSTM</td>
<td>0.94</td>
<td>0.92</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>Bi-LSTM</td>
<td>0.96</td>
<td>0.92</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>1-CNN</td>
<td>0.96</td>
<td>0.90</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>Multi-CNN</td>
<td>0.97</td>
<td>0.93</td>
<td>0.95</td>
</tr>
</tbody>
</table>
**Answer to RQ2:** Our findings demonstrate that the size of the dataset has a greater impact on the deep learning models' performance. They also show that the balanced dataset is more important than its size. Regardless of the balancing technique used, Near-Miss or SMOTE, Multi-CNN with GloVe-Twitter and GloVe-General perform best on both balanced datasets. This further confirms the efficiency of these models in chat data training — Multi-CNN for its global feature identification method and the word embeddings for their similarity with chat transcripts. DL models perform the best overall in this study with all three metric values in 90s on the balanced dataset using the SMOTE technique.
Chapter 5

Discussion

This chapter presents the overall findings in Section 5.1, discusses the implications of this study in Section 5.2, and addresses potential threats to validity in Section 5.3.

5.1 Findings

From Chapter 3 and Chapter 4 (i.e., Methodology and Results, respectively), we understand that disentangled Discord dataset complements the Slack dataset by providing a large pool of Q&A instances. We also observed the better performance of deep learning models with different word embeddings 4.1.4 over the heuristics when replaced in the original ChatEO [1]. Finally, we witnessed that the DL models outperform themselves if provided with a larger and balanced dataset 4.2 through class balancing methods.

As the software community moving towards Discord and a lack of research in mining Discord chat discussions, we want to provision the wider use of Discord data in software engineering research. This has led us to publish a disentangled Discord dataset 38 that can benefit software engineering community in advancing and developing new tools and techniques for developer chat conversations. We included the published Discord dataset along with the Slack dataset while improving the ChatEO heuristics-based architecture in our work. One reason to include Discord data in our research is to balance out the limitations of Slack such as the message cap of 10,000 messages and admin-authorized transcript downloads. Another reason is the Q&A
format that Discord data follows same as Slack used in original ChatEO. Section 5 shows that Discord does have a large amount of Q&A data for even a year compared to the two-year data of Slack due to its no messages cap rule combined with the free download options. We have also seen from the manually labelled 2,000 instances dataset, the abundance of opinion Q&A when combined both Discord and Slack data. This validates our choice of mining Discord chat data in our research.

Opinion-mining is one of the important topics in the NLP research by providing resources to develop virtual chat assistants, chatbots, Q&A thesaurus, and more. Section 4.1.3 showed the performance of the existing heuristics-based model for the opinion-asking question identification of ChatEO with the metric values being in 60s. As discussed in Section 4.1.5, one of the main reasons is that heuristics are not extensive to cover all the instances with the dataset’s growth. To move on from the limitation of the heuristics, DL models with proven efficiency in various NLP tasks are used to enhance the performance of ChatEO’s question-answering system. Section 4.1.4 showed the performance improvement in all the three metrics such as precision, recall, and F-measure with DL models. Multi-CNN performs the best in all the three metrics, achieving 0.87, 0.85 and 0.84 in precision, recall and F-measure, respectively. This is because CNN can perform better in text classification tasks due to its use of global features in classifying the sentence. The model’s depth with 3 convolutional layers also plays a major part in its performance, as we know that the deeper the CNN model, the higher the understanding of the source dataset. In embeddings, GloVe outperforms Word2Vec due to its use of global statistics that complemented the Multi-CNN model and resulted in the best combination. The interesting observation here is the high or equal performance of GloVe pre-trained on Twitter data compared to GloVe pre-trained on Stack Overflow in recall and F-measure. Even though Twitter pre-trained GloVe is more general than software-related Stack Overflow trained GloVe, the comparatively better performance reflects the similarity between Twitter and our chat dataset. Both of them are short and informal in nature. Another reason might be that we only pass the questions of the chat conversations to the DL models and so, the availability of technical terms is also sparse than in the Stack Overflow embeddings.

Section 4.1.5 helped us to understand the better performance of DL models over
heuristics in depth through a manual quality analysis. The correctly classified instances by the DL model where the heuristics failed to classify properly demonstrated the following characteristics of the heuristics:

- Heuristics are not extensive and cannot cover all the data points unless we can codify them in our rules which is very expensive.
- Heuristics are rigid and very sensitive; they are prone to misclassifying instances when they slightly deviate from the predefined linguistic patterns or have minor spelling mistakes.
- Heuristics are not good at capturing the context that led to many misclassified instances.
- Heuristics are not good in classifying instances with code blocks that are common in developer chat conversations.

DL models are good at the above-mentioned points due to their neural net architecture and word embedding features. Word embedding features are the core of any DL model as they can project text data as low dimensional vectors that are easier to correlate and learn by the neural nets. They are also less time-consuming owing to the wide availability of pre-defined embeddings for different domains. They are the main reason for leveraging deep learning for NLP tasks as discussed in Section 2 from machine learning models with extensive feature engineering such as bigrams, trigrams, the parts of speech, counts of the polarities of sequences of semantically oriented words, the average semantic orientation score of the words in the sentence. Our experiments with DL models also proved the effectiveness of using them in NLP tasks.

To use the full potential of DL models in our research, we wanted to increase the dataset size as it is suggest for achieving high performance in DL models. But as the manual labelling of the data is expensive, we opted to go with automatic labelling. We have applied Snorkel, a weak learner, in labelling a larger dataset using improved heuristic rules as described in Section 4.2. A micro-averaged F-score was calculated by manually labelling 5% of the larger dataset to check the quality of the Snorkel-generated labels. The results after running the DL models showed us that
even though a larger dataset was used, the imbalanced nature of the dataset classes hinders the DL model’s performance.

As the results showed us the importance of class balancing, we opted to check two variants - SMOTE and Near-Miss as explained in Sections 3.3.2 with their results in Section 4.3. With a little larger data using Near-Miss, we can see the similarity between its results and the results of the 2,000 instance dataset 4.1.4. This proves that the deep learning models need a much larger dataset to show significant improvement. SMOTE with the largest balanced data in our research improved the efficiency of DL models remarkably 4.3. With the value of the metrics in 90s, SMOTE ultimately proves the potential of DL models in NLP given a balanced, labelled large dataset. GloVe (Twitter and General) along with Multi-CNN performs the best for both the class balancing techniques for their global nature in extracting and processing features.

5.2 Implications

Software development is facilitated by different communication mediums available to developers on the projects they are involved in. Many software communities and ecosystems have adopted Discord server as their chat platform. In our work, we collect and prepare a new Discord dataset 3.1 to foster future research on mining developer chat conversations. The key contributions of our work is the improvement of the existing technique of identifying opinion-asking questions in these chat conversations 3.2 and the development of a large labeled dataset using weak supervision 2.5.1. Thus, our research findings are useful for researchers, software developers, Discord chat users and moderators, and domain experts.

5.2.1 Implications for researchers

Now that we have a foundation for mining Discord data, it can be used for developing different applications. Discord data shows the recent trends within the software community provided most of the software communication moved to Discord in recent years, it can be used to find potential new research directions such as development of
chat summarization tools, chatbots, virtual assistants, etc. The dataset we created was from programming communities following a Q&A format as we wanted to mine opinion-asking questions. Researchers could explore other servers and channels other than programming based for their research agendas and create their own disentangled dataset. We have shared our code and implementation of the conversation disentanglement approach. They can also use our results as a baseline to evaluate different disentanglement techniques. Given the potential of modified ChatEO for Q&A identification, researchers can use it to create a detailed Q&A thesaurus. Snorkel employed in our study can help researchers to explore new automatic labelling methods that are more time-efficient and cost-effective.

5.2.2 Implications for software developers

Software developers can use our opinion-asking question answers to understand the software’s popularity and identify the area for improvements in their software. By applying topic modelling, they can understand the customer standpoints that can help with their development and feature additions. It can also help them to debug errors unknown to them during development by understanding the user concerns. Given the new Discord data creation process, software teams can create datasets that can help them to summarize conversations, gather insights, identify specific user contributions, document incremental changes, and populate FAQs to help new team members and more. ChatEO can serve as a base for developers to develop Q&A systems for different question types based on their software needs (for instance, to find facts). With the availability of a larger dataset using Snorkel, developers and tool makers can spend more time creating different ML or DL models for their tasks rather than spending it on labelling data.

5.2.3 Implications for Discord users and moderators

Chat conversations are the source of our research work. Most chat communication is around doubts, debugging, best practices, or new tools/trends. With our new disentangled dataset that groups conversations, Discord users can search for answers to the frequently asked questions instead of repeating them. It can save their time
and effort by avoiding scrolling through entwined conversations. Discord moderators can use the dataset to create FAQs on common questions that can benefit existing and new users. The need for new channels can be identified based on the frequency of discussion of certain topics. Moderators can build a repository with links to useful resources to learn particular tools/APIs/packages. They can build bots using the data to moderate discussions with redundant questions and guide users to their target channels.

5.2.4 Implications for domain experts

With our process of disentangling conversations, domain experts can generate informal documentation about different topics and concepts by creating their own datasets from different channels - design, data science, programming, and more. Experts can use our opinion-asking questions and answers to understand and update their domain knowledge as chat communication is a rich source of discussion of recent trends and tools. Some problems need stronger supervision for labelling by domain experts, for instance, specialized document classification (e.g., legal). The demonstrated efficiency of Snorkel in our work can allow domain experts to update or create their rules and use Snorkel as a pre-processing step for labelling.

5.3 Threats and Limitations

In this research, we identified several threats to the validity of our work, and we now describe them along with the measures we took to control them.

Data selection: We chose Discord programming servers and their popular channels to create our DISCO dataset. With more daily active users, we were able to get meaningful conversations with the disentanglement technique used. However, our study results may not be transferred to other chat platforms or developer communication. This uncertainty can be mitigated by scaling our approach to a wider developer communication set to understand its implications. Another threat could be the programming servers chosen within Discord as other channels with fewer daily active users could have affected our results. Our goal was to share a larger dataset;
but since the Discord chat data is copyrighted, we could only collect 10% of the data for research purposes [24]. While the currently shared Discord dataset serves as a starting point, we will be investigating other ways to expand this dataset in future.

**Manual annotations:** The smaller dataset in our study was annotated manually, this can be another threat as it can affect the validity of the labels. To mitigate this, we have used annotators experienced in both chat platforms and asked them to follow an annotation rule set to identify the opinions. We have also reported an inter-rater agreement coefficient, Cohen’s Kappa. It yielded a high value proving the sufficient agreement between the annotators.

**Automatic disentanglement:** The disentanglement process used in our research was automatic. The errors in this process can pose a threat to the research’s validity. It was mitigated by manually disentangling a smaller dataset and calculating a micro-averaged F-score. The high value of this metric validated the efficiency of the disentanglement algorithm used in our research. There could have been biases in our scripts used in pre-processing chat data before and after disentanglement. It was mitigated by performing unit testing.

**Automatic labelling:** Another threat could be the automatic labelling of the larger dataset using the weak learner Snorkel. The errors in this could affect the performance of DL models with or without balancing. To mitigate this, we manually labelled a smaller portion of data and calculated a micro-averaged F-score. This measure gave us a higher value demonstrating the good quality of the Snorkel-generated labels. Class balancing applied can also create a threat to our results. Near-Miss removed most of the majority class which could affect the learning of the majority class by the DL models. SMOTE created synthetic data points that can be difficult to interpret to understand its effect on the results.

**Training setup:** The account of randomness in DL model training could be another limitation. It can be mitigated by employing cross-validation techniques while training. Both class balancing techniques are used with a class ratio of 1:1 to create a fully balanced dataset to compare with the imbalanced larger dataset. Other class ratios can be tried to understand their impact on class balancing techniques.
Chapter 6

Conclusion

We conclude this thesis with a summary of our contributions in Section 6.1 and our suggestions for future research directions in Section 6.2.

6.1 Summary of Contributions

Owing to the recent popularity of the Discord platform as a medium for conducting chat communication within software development communities, our work enables Discord chat mining and uses it together with Slack data to improve an opinion-asking question answering system by replacing a heuristics-based approach with a deep learning architecture. Additionally, we have explored the conversation disentanglement process, word embeddings, hyper-parameter tuning, automatic labelling and class balancing techniques. Our contributions to the research community have been summarized and presented in this section.

A new Discord server dataset, DISCO was created using four software development communities that follow a Q&A format and offer general technical help (python, golang, racket, clojurains). The entwined chat transcripts were pre-processed and subjected to a modified Elsner and Charniak disentanglement technique to identify conversations. It is a collaborative effort where I have contributed to collecting original transcripts, prepossessing, manual labelling and applying the disentanglement technique. They were used as a source in our study along with a similar disentangled Slack dataset.
An existing Q&A system, ChatEO was modified to replace the heuristics-based opinion-asking question identification process with deep learning methods. Different hyper-parameters tuned deep learning models with different word embeddings were used to evaluate the efficiency of each of them in detecting opinion-asking questions. To train them, we have created a manually annotated labelled dataset of 2,000 conversations randomly selected from both Slack and Discord disentangled datasets. DL model results were compared with the original heuristics-based approach of ChatEO on different metrics (precision, recall, and F-measure) as well as using a qualitative study. Both results show the better performance of the DL model compared to the heuristics for opinion-asking question identification. Multi-CNN with GloVe-Twitter word embedding achieved the best performance. This is due to the identification and correlation of the global features along with the informal nature of chat transcripts similar to Twitter data.

As DL models need more labelled data and manual labelling is very expensive, we have used an automatic weak learner, Snorkel to create a larger dataset of 44,820 instances. We obtained average results in DL models on this dataset as it was not balanced. We have employed two class balancing techniques - NearMiss with 44,72 instances in each class and SMOTE with 39,650 instances in each class. Multi-CNN with GloVe-Twitter and GloVe-General produced the best performance in both cases. This further proves the efficiency of these techniques for chat conversation mining. DL models with SMOTE produced the best metrics scores in our study in the range of 90s.

In summary, 1) We have created three datasets in our study - a disentangled Discord dataset, a manually annotated small dataset for opinion-asking questions, and an automatically labelled larger dataset for opinion-asking. 2) We evaluated several DL models with different word embeddings to improve the performance of ChatEO’s opinion-asking question identification process - using the smaller dataset, using the larger unbalanced dataset, and using two balanced larger datasets.
6.2 Future Work

Our study presents different future research opportunities. The immediate next step in our research will be to enhance the answer extraction process of ChatEO. Currently, Bi-LSTM is used to extract the answers. We can substitute Bi-LSTM with an attention-based LSTM. Another potential enhancement would be to combine both question identification and answer extraction process instead of having independent neural nets. The word embeddings can be investigated further by substituting with context-based and character-based embeddings such as ELMO and Character-BERT, respectively.

Cross-validation can be used on the model combination to help it generalize to independent datasets in future. Another research direction would be to employ different class ratios in class balancing techniques to understand their impact on the results. The explainability of deep learning models would be a non-domain-specific future research direction that could pave a new era for deep learning models. We can also use different statistical analysis testing as additional evaluation metrics to ensure the effectiveness of the best models.

One research direction is to create FAQs, Q&A thesaurus and knowledge graphs from the opinion-asking question answers extracted by ChatEO. By expanding the Q&A corpus, it can act as a tool support for recommendation system and help in understanding developer behavior. Also, researchers will have an opportunity to work on Conversational Search Assistants (CSA), chatbots and more. As the current modified ChatEO only focuses on the opinion-asking question answers, it can be expanded to include other questions types such as fact seeking questions.

Another research direction would be to extend the source dataset created with more diverse Discord servers or with other developer communication medium. With the help of Snorkel used in our research, this larger datasets can be labelled in shorter time. Explanability and impact of synthetic data points on DL model’s performance can become another interesting research topic.
List of References


[71] “Url to access slack.” [https://slack.com/](https://slack.com/)

[72] “Url to access discord.” [https://discord.com/](https://discord.com/)

[73] “Url to access gitter.” [https://gitter.im/](https://gitter.im/)

[74] “Url to access github.” [https://github.com/](https://github.com/)

[75] “Url to access python server of discord.” [https://discord.com/channels/267624335836053506/267624335836053506](https://discord.com/channels/267624335836053506/267624335836053506)


