LOW DATA DIALOGUE ACT CLASSIFICATION FOR VIRTUAL AGENTS DURING DEBUGGING

A Thesis

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science
in
Computer Science and Engineering

by

Andrew Edward Wood

Collin McMillan, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
September 2019

LOW DATA DIALOGUE ACT CLASSIFICATION
FOR VIRTUAL AGENTS DURING DEBUGGING

Abstract

by

Andrew Edward Wood

A "dialogue act" is a written or spoken action during a conversation. Dialogue acts are usually only a few words long, and are divided by researchers into a relatively small set (often less than 10) of dialogue act types, such as eliciting information, expressing an opinion, or making a greeting. Research interest into automatic classification of dialogue acts has grown recently due to the proliferation of Virtual Agents (VA) e.g. Siri, Cortana, Alexa. But unfortunately, the gains made into VA development in one domain are generally not applicable to other domains, since the composition of dialogue acts differs in different conversations. In this thesis, I target the problem of dialogue act classification for a VA assistant to software engineering repairing bugs in a low data setting. A problem in the SE domain is that very little sample data exists. Therefore, I present a transfer-learning approach to learn on a much larger dataset for general business conversations, and apply the knowledge to a manually created corpus of debugging conversations collected from 30 professional developers in a "Wizard of Oz" experiment and manually annotated with a predetermined dialogue act set. In experiments, we observe between 8% and 20% improvements over two key baselines. Additionally, I present a separate dialogue act classifier on the manually collected dataset that uses a manually discovered SE specific dialogue act set which achieves on average 69% precision and 50% recall over 5-fold cross validation.

CONTENTS

FIGURES

TABLES

ACKNOWLEDGMENTS

I would like to thank my family and my fiancée Karen for their loving support and patience during the construction of this thesis. I also thank my advisor, Dr. Collin McMillan as well as my committee members: Dr. David Chiang and Dr. Jane Cleland-Huang, for their tireless work, advice, and guidance, without which this thesis would not be possible.

I would also like to thank the University of Notre Dame, my labmates Zachary Eberhart, Alex LeClair, Paige Rodeghero, Ameer Armaly, and Siyuan Jiang, and the 30 professional software developers who participated in this research study. This work is supported in part by the NSF CCF-1452959, CCF-1717607, and CNS-1510329 grants. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

CHAPTER 1

INTRODUCTION

A "dialogue act" (DA) is a written or spoken action taken during a conversation. For example, an utterance "turn left at the next light" is an instruction, versus an information elicitation such as "in which direction should I turn?" Dialogue acts are important components of conversation analysis and modeling, and have been studied for decades: first in sociology [91, 7] and later in computational linguistics [88]. DAs are "the minimal unit of linguistic communication" [91] and are key to both automated comprehension and generation of natural language dialogue.

Dialogue act classification is the automated placement of utterances into the appropriate DA type. It is a well-studied problem, often treated as a subcategory of text classification, that has largely followed the history of other areas of applied machine learning: early attempts involved manually-curated feature sets based on word usage [3], later enhanced by knowledge of relationships between DA types such as common conversation flows [9]. Today, state-of-the-art performance on large academic datasets is achieved by one of several proposed neural network-based architectures [23, 50, 56, 47].

Interest in dialogue act classification has ballooned due to the proliferation of automated virtual agents (VAs) such as Siri, Cortana, and Alexa. For VAs to converse with human users, VAs must analyze utterances in a similar way as humans. And while human conversation can seem effortless at times, in fact there are several key steps that we do without even being aware [102, 32, 46, 67]: we detect when dialogue acts occur, we comprehend the dialogue act as being a particular type of act (e.g., an information request, a command, a clarification), and craft an appropriate response. Therefore, one of the first

tasks a VA must perform is classification of a human conversation participant's utterance into a dialogue act; the VA must know whether it is e.g. being given an instruction or being asked for information before it can craft a reasonable reply. And for many VAs, large and context-appropriate datasets exist to train strong classifiers [94]. For example, a designer of a VA for appointment scheduling could draw on a dataset with many thousands of utterances [18].

But this happy situation is not present in many specialty applications where datasets are rare and expensive. As Gangadharaiah *et al.* [36] pointed out at NAACL'18, "methods that achieve state of the art performance on synthetic datasets perform poorly in real world dialog tasks." Likewise, Kang *et al.* [49] emphasize that methods trained on datasets in one domain tend not to generalize well to problems in other domains, since the number and composition of dialogue act types differ dramatically.

In this thesis, I target the problem of dialogue act classification for automatic virtual agents in the domain of *software engineering*. Specifically, we envision constructing a VA to assist software engineers during debugging: the imagined situation is that a programmer receives a bug report, and has questions about the codebase in which the bug occurs. A VA is desirable in this situation because debugging is often assigned to junior programmers who are learning a new codebase [11] and may have more questions than senior colleagues can handle [87].

Datasets in this domain are extremely expensive to create. Currently, there are no publicly available datasets that target the problem of debugging in software engineering. One key accepted strategy for collecting examples of conversations between human users and a VA is called a "Wizard Of Oz" (WOZ) experiment [28, 79]. In a WOZ experiment, human participants interact with a machine that the participants believe to be automated. In reality, the machine is controlled by human experimenters. The participants are asked to use the machine for a particular purpose (in our case, to debug programs). This deception is necessary to accurately reflect real-world use where human users assume they are talk-

ing to a machine, where language differs than if users assume they are interacting with a human [28, 79]. By recording transcripts of these conversations and manually annotating them with DAs, datasets appropriate for training automated dialogue act classification systems can be constructed. However, such datasets typically contain only several thousand utterances due to several bottlenecks that limit the size of the dataset: in software engineering, the human participants of WOZ experiments are professional software developers and must be paid competitively with respect to expected developer salaries, and manual transcript annotation is a time-consuming process.

Therefore, in this thesis, I collect the first-of-its-kind software engineering debugging dataset using WOZ experiments called the *Madeline SE debugging corpus*, manually annotate the Madeline corpus with software engineering specific dialogue acts using an open coding procedure, and provide a baseline from which further dialogue act classifiers can measure against. We recruited 30 professional programmers to fix bugs for two hours each, while providing an interface to a WOZ simulated virtual agent with which the participants interacted. Across the 30 two-hour conversations, we made 2459 annotations and discovered 26 software engineering debugging-specific dialogue act types. The baseline dialogue act classifier achieved an average 69% precision and 50% recall across 5-fold cross validation experiments.

Additionally, I present a transfer learning architecture to "learn what we can" from an available, large dataset and apply it to the specialized SE VA problem. My approach works by re-annotating the Madeline dataset with the dialogue act set of the larger dataset, and combines two encoders: a "global" encoder trained with the larger corpus of generic conversations, and a "local" encoder trained with the limited Madeline corpus.

I evaluate my transfer learning approach against two baselines: one baseline is a recent applicable approach from related literature trained only on the limited Madeline corpus, and another baseline is the same algorithm trained with data from a large corpus of generic conversations. I further select a subset of the dialogue act types as a target set, since not

all types are of equal value for a VA to recognize. My experimental results show that our approach outperforms the baselines by 8% and 20% on these target dialogue act types.

By releasing the Madeline corpus and transfer learning approach, we contribute one of the very few WOZ corpora which are extremely rare in software engineering [93]. We release all data, including anonymized conversations, annotations, and all dialogue act classifiers via online appendices to promote reproducability and assist future research in software engineering virtual agents.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter summarizes the history of research on classification of dialogue acts, Wizard of Oz studies, and interactive natural language systems for software engineering.

2.1 Problem, Significance, and Scope

At a high level, the problem I target in this thesis is that models of developer conversations are not described in the literature. Certainly, strong efforts in the area of program comprehension have made inroads into our (the software engineering community's) understanding of the types of information that programmers need and how programmers make sense of software problems. However, the "nuts and bolts" of actual conversations among programmers are still not well-understood.

A key component of those nuts and bolts are "dialogue acts" (as defined in the previous section), and our goal is to automatically detect these dialogue acts in conversations. But detection of dialogue acts is useful beyond pure academic interest: advancements in programmer tool support depend on improved detection of programmer intent. Numerous software engineering tools depend on natural language interfaces, such as code search engines, navigation tools, traceability tools, and our target context of **automated virtual assistant** technology. The situation we envision is that a programmer asks an automated virtual assistant a question in lieu of a fellow human programmer, and the virtual assistant is expected to provide an answer to that question. A fundamental part of answering these questions is to detect the types of statements, comments, etc., that programmers make when asking and clarifying their questions.

Throughout this thesis, I refer to a 2011 book by Rieser and Lemon [79] as both motivation for and rationale behind my work. The book provides an excellent summary of the design decisions required for building dialog systems and reflects the significant momentum in years of research on virtual agents – one key theme is that using Wizard of Oz studies to inform data-driven dialog system construction is a highly effective strategy. They point out that while it is possible to design a virtual assistant using manually-crafted assumptions about user behavior, the existence of annotated, simulated dialog (via a WoZ study) provides an immense boost to the flexibility and effectiveness of virtual agent design. One benefit is from the increased knowledge scientists gain from studying the dialog, while another benefit is from the ability to use supervised and reinforcement learning algorithms to "teach the computer" correct behavior, even with relatively sparse data.

In this thesis, I contribute the dataset, our manual annotation of the dataset, our analysis of those annotations, and our transfer-learning models to the community as a foundation for building better software engineering virtual agents. This contribution alone is significant, considering that a recent survey by Serban *et al.* [93] found only four publicly-available WoZ datasets (more are held privately) suitable for building dialog systems – and none related to Software Engineering. However, I take a further step towards a working virtual agent by building a classifier to automatically label the dataset; in essence, this is a detector for dialogue act type using supervised learning (as chapter 7 of [79] highlights, supervised learning is often the first technique tried for dialogue act type detection, prior to resorting to more complex approaches).

Note that in our manual annotation process, we annotated the entire conversation (both "Madeline's" and the study participants' side). However, during the dialogue act type detection, we only predict the type of dialogue acts from the participants' side of the conversation. This is because during the manual annotation process, we study not only the participants, but the wizards' actions also: this is for the purpose of laying a groundwork for conversation flow analysis in future work, in addition to the academic interest presented

in this thesis. But, during dialogue act detection, the realistic scenario is that a virtual assistant would only need to detect user DAs, and never need to classify its own conversation, since it would already know the dialogue act types it generated itself.

The long-term vision of this thesis is an automated virtual agent for software engineers during debugging, as mentioned in the previous section. Following the process recommended by Rieser and Lemon, we have collected a debugging conversation corpus from Wizard of Oz experiments, and have attempted to build effective dialogue act classifiers based on the published state-of-the-art. However, in pilot studies we found results similar to Gangadharaiah *et al.* [36] and Kang *et al.* [49]: the latest technology often does not achieve usable results in practice.

In my view, the reason for relatively low SotA performance boils down to two points targeted in this thesis: First, the size of the available data is quite limited in my application, when compared to the large synthetic datasets used in many papers on dialogue act classification. The neural algorithms used in recent papers are notorious for requiring tens of thousands or more examples for training for good results in text classification. The WoZ dataset I use is not even one tenth as large.

Second, the composition of the dialogue acts in conversations is quite different in the collected dataset, meaning that the accuracy levels reported for large synthetic datasets do not apply. A majority of papers on dialogue act classification report an overall accuracy for all classes. For example, Chen *et al.* [23] report a remarkable 91.7% accuracy, versus 90.9% for a baseline on a standard dataset. However, the performance can vary considerably for different dialogue act types. In my situation, I care a lot more about acts denoting information requests (given that we seek to answer questions) than I do about greetings or opinions. Also, it is important to distinguish differences such as a request for information and a request for an assessment. But DA types associated with these are among the worst performers in DA classification. The result is that my "in practice" usage of off-the-shelf DA classification is much lower than the high reported (90%+) overall accuracy suggests.

TABLE 2.1
DA CLASSIFICATION HISTORY AND METHOD PROPERTIES OVER THE LAST 20 YEARS.

	W	M	H	N	P
Andernach [3]	x	x			
Reithinger and Klesen [76]	x		x		
Stolcke et al. [99]	x	x	x		
Serafin et al. [92]	x				
Grau et al. [40]	x	x			
Webb et al. [106]	x	x			
Ang et al. [4]	x	x			
Surendran and Levow [100]	x		x		
Geertzen et al. [37]	x	x			
Zimmermann [109]	x	x			
Boyer et al. [16]			x		
Tavafi et al. [101]	x		x		
Blunsom et al. [14]	x	x		x	
Milajevs and Purver [62]	x		x	x	
Khanpour et al. [50]	x			x	
Ji et al. [47]	x		x	x	
Lee and Dernoncourt [56]	x			x	
Liu et al. [60]	x		x	x	x
Kumar et al. [54]	x		x	x	x
Chen et al. [23]	x		x	x	x

Selection of closely-related projects targeting Dialogue Act Classification over the last twenty years. Column W indicates whether words in the dialogue were used by the classifier. M indicates manually-crafted features other than words (but which may be based on words, e.g. length). A mark in the H column means that the history of previous dialogue acts in a conversation is used to predict the current act type. N indicates whether the approach is based on neural nets. P indicates whether the approach is post-hoc, that is whether it predicts all act types for a whole conversation, rather than one act at a time (i.e. ongoing conversations).

2.2 Dialogue Act Classification

Table 2.1 summarizes the key related work on Dialogue Act Classification in chronological order. Related work can be broadly categorized along the five dimensions in the table. Observations include: 1) Nearly all related work uses the words in a dialogue act as features for classification in on way or another. The way they are used varies considerably, with some projects relying on a bag-of-words representations and others using n-grams

or sequence-based representations (e.g. as provided by a recurrent neural network). 2) A shift from manually-crafted features to neural net-based approaches is clearly visible starting around the year 2013. This shift reflects the trend across many areas of NLP and AI research, as a recent survey by Chen *et al.* points out [22]. And, 3) a majority of studies have found that history of previous dialogue act types in a conversation improves classification performance.

The neural net-based approaches can be further divided as either post-hoc or online. A post-hoc approach is one that labels every utterance in an entire conversation that has already been completed, in contrast to an online approach in which labeling only requires a "current" dialogue act and previous ones in the same conversation. Some neural net-based architectures are designed such that the structure of a whole dialogue is input – Kumar *et al.* [54], for instance, have recurrent "sentence level" layers that output to another recurrent "conversation level" layer that ultimately produces predictions for every utterance in a conversation. This structure is useful in post-hoc analysis of conversations, but does not match the need for predictions in VAs, which must classify dialogue acts in real time.

2.3 Studies of Program Comprehension

This thesis could be broadly classified as a study in program comprehension – how programmers comprehend and communicate about software development and behavior. Typically questions asked by program comprehension literature relate to the mental and physical processes that developers follow [44, 55]. Examples of mental processes include targeting how code is connected [63, 53, 96, 97]. Physical processes include taking of notes [2] and patterns of movements of the eyes [83, 95]. Notably, Roehm *et al.* [86] point out that programmers "try to avoid" program comprehension, and look for short cuts whenever possible. This finding is in line with several others that suggest that tool support for comprehension should provide information incrementally and at as high a level as possible, and avoid too many low-level details [98, 33, 59]. Our vision in this thesis is

to build a foundation for software engineering virtual assistants, to provide information in the order and at the time requested by programmers during a dialog.

2.4 Software Engineering Virtual Agents

Virtual Agents to assist software engineers during development have been envisioned for decades [103, 15, 81, 82], but the recent proliferation of virtual agents for general tasks (e.g. Siri, Cortana, Alexa) has reignited research towards that vision [82]. Key related work includes WhyLine by Ko and Myers [51] which attempts to explain program behavior, TiQi by Pruski *et al.* [72] a dialogue system for database queries, and a natural language dialogue system by Escobar-Avila *et al.* [30] to accompany video tutorials.

We are still far away from this dream. Nevertheless, advancements are being made in that direction. Recently, Bradley *et al.* [17] built Devy, a virtual agent to help automate programmer tasks. Devy differs from our work in that we seek to understand the structure of programmers' conversations, to build a system to help programmers learn and recall information, rather than automate tasks. Ko and Myers [51] created Whyline, which answers questions about program output. A majority of current efforts focus on understanding unstructured software engineering data; for a more complete survey we direct readers to Arnaoudova *et al.* [6]. But what holds back progress at the moment is an incomplete understanding of how programmers communicate – it is not possible to build a tool that participates in this communication without understanding the nature of that communication. This understanding can only be completed with conversation analysis and modeling.

2.5 Conversation Analysis and Modeling

Conversation analysis and modeling is the task of extracting meaning from human written or verbal communication. It usually involves creating a representation of a type of conversation (e.g., restaurant recommendations, or technical support calls [79]), and then

using that representation to predict the flow of the conversation. A "flow" of a conversation is how people tend to put information in conversations, for example one conversation participant asking "does that make sense?" if the other participant is silent after receiving new information. Conversations are typically broken up by *turns* [102, 32, 46, 67, 90]. A turn begins every time a speaker begins speaking and can encompass multiple sentences. Conversation analysis and modeling is what allows automated virtual assistants to create human-like conversations.

Conversation modeling has its roots in sociology [102, 32, 46, 67, 90] and psychology [41], where researchers studied the factors behind conversation flow and form. These often employ qualitative analysis methods to isolate human factors such as social rank or fatigue. After a significant investment in the 1990s, quantitative analysis procedures have been developed to model and predict the types of information that human conversations include, in order to create interactive dialog systems. Work in this area has flourished, with representative work including: [42, 105, 77, 65, 64, 29, 31]. For example, work by Lemon [58, 79] models restaurant recommendation conversations as a Markov Decision Process, in which each turn is one of six possible states.

A typical strategy in conversation modeling for discovering dialogue acts is *user simulation*, in which participants in a study are told that they are interacting with a dialog system, which is actually a human acting like a dialog system via a chat program [1, 89]. The simulation results in a transcript of a conversation between a human participant and an idealized virtual assistant (simulated by the researcher). The transcript is an extremely valuable source of information on how the human participant expects to interact with a machine and how the machine should respond. While rare in Software Engineering, these studies are not unheard of: Goodrum *et al.* [39] perform a WoZ study to discover what requirements knowledge programmers need, related conceptually to requirements-gathering WoZ studies proposed earlier [107].

CHAPTER 3

THE MADELINE CORPUS: EXPERIMENTS WITH A SIMULATED VIRTUAL AGENT

This chapter discusses how the software engineering debugging dataset was collected and annotated, as well as dialogue act classification baselines constructed. This chapter begins with sections describing the experimental setup, methodology, and annotation scheme, with subsequent sections covering dialogue act classification. This work was published in the 2018 ACM Joint Euopean Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) [108].

3.1 User Simulations

In this section, we describe our user simulation study. In general, a user simulation is an imitation of a conversation between a human and a machine – instead of a real machine, a researcher stands in for the machine without the human being aware of it [28]. In this, our user simulation is the interaction between our participants and an imitated virtual assistant. Participants believed the assistant could automatically assist programmers with tasks. They were informed their participation in this study was helping to improve the assistant for use by programmers. However, there was no actual virtual assistant responding to the questions asked by the participants; we manually answered every question the participants had.

3.1.1 Methodology

We based our methodology on previous studies of bug repair in software engineering [48, 38, 52] and previous "Wizard of Oz" studies in sociology [28]. We asked the

programmers to remotely participate in the study using a provided Ubuntu 64-bit virtual machine and the Microsoft Skype application on their local machine. We instructed the participants to fix bugs from pre-installed open source Java projects contained within an Eclipse IDE [35] workspace on the provided virtual machine. We instructed the participants to fix as many bugs as they could within a pre-defined two-hour time frame. During that time, we gave the participants one bug at a time, one bug per project. We asked the participants to avoid using the Internet to search for solutions or answers to any questions that they might have, and to instead direct their questions to an automated virtual assistant named "Madeline" through the Skype platform. Note that this involved two key design decisions informed by Rieser and Lemon's guide on WoZ studies for dialog systems (chapter 6 of [79]): First, we used the written text chat only, no voice, to limit the scope of the study to developer Q/A conversations instead of introducing the possibility of voice transcription errors (it is necessary to deliberately add noise to WoZ studies involving voice to simulate actual noise, and we felt this would add too many variables considering the already complicated nature of debugging). Second, we restricted access to internet resources. While this may seem to create an unrealistic situation (since programmers frequently use Stackoverflow, etc.), it was necessary in order to learn how programmers might use a virtual agent, due to a bias in which people might not try a new technology simply because it is unfamiliar, and to avoid biases introduced by external tools. These restrictions are often a "necessary evil" in WoZ experiments – for example, 94% of papers surveyed by Riek [78] placed substantive restrictions on participant behavior and resources.

During each study, two to three of the authors collaborated at all times to respond to the participants. At least one of the authors had previously fixed the bugs given to the participants. This allowed for quick and intelligent responses to the participants, giving the illusion that Madeline produced responses automatically. This deception, typical of the "Wizard of Oz" strategy [27] was necessary to ensure the authenticity of the responses. The participants were explicitly told that they were communicating with an automated system

supervised by humans (Madeline). The participants were told to interact with Madeline through Skype conversations, and also to share their screens for quality assurance purposes. In reality, screen sharing provided the means to prepare responses in real time and was critical for imitating a fully autonomous system. Following Rieser and Lemon's WoZ process for dialog systems (again, chapter 6 of [79]), we did not restrict wizards to a script or set of predefined dialogue act types, since a goal of our study was to understand what the programmers needed rather than test a predefined script.

3.1.2 Participants

We recruited 30 professional programmers to participate in our study. These programmers were recruited through email and an online freelance website called Upwork [104]. The programmers work at various companies such as IBM, GolfNow, and Hyland Software, while some work as freelancers full time. Note that the programmers recruited are not students, but professionals working in industry. Each programmer recruited had familiarity with Java before participating in the study. Overall, the participants had an average of 5.5 years of experience with Java. The maximum number of years of Java experience was 12 and the minimum was one.

3.1.3 Threats to Validity

As with most studies, this project has a few threats to validity. First, since each experiment was two hours long (not including any technical problems), it is possible that the participants experienced fatigue. This is compounded with any fatigue that they already experienced from their normal work schedule. This was mitigated by using a large pool of participants. Another threat came from technical problems with screen sharing. The only issue with this, however, was a possible reduction in response speed, but we saw no noticeable reductions in any of the studies. Either through technical problems or participants forgetting to save them, a few screen shares were unable to be stored. However, these stored recording were not actually used in analysis. Finally, another threat to validity was

our lack of control over whether participants actually refrained from searching for answers over the Internet rather than asking our simulated virtual assistant. Participants could have used another device to search the web. We did not notice any extended lapses in questions or work time from any participants, though, so we believe most participants followed our searching instructions correctly.

3.1.4 Data Collection

We provided each participant with an Ubuntu 64-bit virtual machine. We asked the participants to download the virtual machine ahead of the study. Inside the virtual machine, we provided Eclipse with all the buggy projects. We also provided a screen recording program called SimpleScreenRecorder [8]. We asked each participant to start the screen recording at the beginning of the study and leave the software running until the study had completed. The participants then saved and sent the screen recording file to us. We then collected the Skype transcript created by the study and removed all identifying information from the conversations. Some participants also sent back the project files of the fixed bugs, but these files were not used in our analysis.

3.1.5 Bugs

The 20 Java bugs come from 17 different open source projects. The project domains include a Pacman game, a calender application, and a PDF file merger. We also selected bugs from commonly used Java libraries such as OpenCSV [68] and Apache Commons IO [34]. We chose the bugs based on four criteria:

1. The bugs had to be simple enough that they could be solved in a few hours, but complicated enough to take at least 20 minutes to solve.
2. We had to be able to understand the bugs well enough to give meaningful answers to questions during the simulation.
3. The user had to be able to reproduce the bug easily.
4. The bugs had to be solvable without obscure domain knowledge.

Project Name: 2048
Bug Report: The game board becomes unresponsive.

```
public GamePane(int size, BasePane basePane)
{
    this.size = size;
    this.basePane = basePane;
    setScore(0);
    this.tileSize = tileSizes[size];
    this.moveTime = 100 * 4 / size;

    setPrefSize(size * tileSize, size * tileSize);
    setLayoutX(175 - (size * tileSize) / 2);
    setLayoutY(175 - (size * tileSize) / 2);
    setStyle("-fx-background-color: #FFFFFF;");
    addTile();

    ... [Irrelevant code cut for thesis space limitations]

    Thread focusField = new Thread(new Runnable()
    {
    @Override
    public void run()
    {
        while(!Thread.currentThread().isInterrupted()) {
            if(!isFocused()) {
                try { Thread.sleep(100); }
                catch (InterruptedException e) {
                    e.printStackTrace();
                }
                requestFocus();
            }}}});
    focusField.setDaemon(true);
    focusField.start();
}
```

Figure 3.1: A description of a bug in the "2048" project with source code. Participants received full copies of the source code, however parts have been omitted for space limitations in this figure.

All of the bugs were previously solved, and we had the actual solutions on hand throughout each study. We also discussed other solutions to the bugs before the user simulations. This is because some of the bugs had multiple solutions. The bugs were presented individually and randomized for each study. An example of a bug given to the participants is as follows (a subset of the source code from this example can be seen in Figure 3.1):

The bug in the source code above occurs when a user tries to make a move with the arrow keys. The source of the bug is the result of an incorrect fusion of a third party library used for graphics (JavaFX) and the structural design of the project. The project contains multiple panes which house buttons performing different types of actions. For the sake of simplicity, consider there to be only two panes; one that displays the board and is controlled by the arrow keys (the "game pane"), and another that allows users to save and load games (the "file pane"). Both of these panes are vying for focus, but for the game to be played, the "game pane" must always have focus. To ensure this, the project's implementation spawns a daemon thread that almost constantly requests focus for the "game pane." The bug comes from the fact that JavaFX only allows for one thread, called the "event thread," to make changes to the UI. When creating the daemon thread, the developer uses the "Thread" type to request focus, which JavaFX interprets as modifying the UI. This causes an exception to be raised, and for the game to become unresponsive to the arrow keys.

One solution to this bug is to use JavaFX safe data types to perform the action of the daemon thread. During studies, participants were only provided with the buggy projects and the bug description. We (pretending to be Madeline), while aware of solutions, would in no form "give" a solution to the participants, but would only react to questions asked. Participants were incentivized to search the source project for the files containing bugs, as questions designed to tease solutions out of Madeline were met with vague and unhelpful responses (i.e. "I am unsure"). A complete list of bugs can be found at our online appendix (see Section 5.3).

3.1.6 Experiences and Lessons Learned

In this section, we discuss our experiences and lessons learned while conducting the user simulation study. We do this to provide some guidance to software engineering researchers who might do studies similar to ours in the future. One of the biggest lessons we learned was to confirm that the virtual machine we provided worked on the participant's machine before the study started. In roughly half of the studies, we found ourselves fix-

ing problems on the participants' machines and spending, on average, an extra 20 minutes fixing the issues. This was problematic, as the studies took up more time than originally anticipated, which threw off our original study schedule. We also learned that additional information should be advertised (beyond the scope of the study) to allow for smooth experiments, such as experience with virtual machines or experience with the Eclipse IDE.

Another lesson learned was how to effectively schedule remote studies. Many participants were unable to participate in the study until they returned home from their jobs in the evening. Some had families and wanted to participate in the study even later, once their children were in bed. Many of our participants were in different time zones, there were days where we would schedule studies at 8 am, 1 pm, and 10 pm in our time zone. We learned, over time, to hire participants overseas where their evening was our work day.

3.2 Annotations

In this section, we describe our process for annotating the dialogue acts from the data collected during the user simulation studies (see Section 3.1.4). Essentially, our goal is to 1) determine what the dialogue acts are and 2) to determine what parts of the conversations are associated with those dialogue act types. We also discuss our research questions, the rationale for asking them, and provide annotation examples.

3.2.1 Research Questions

The research objective of this section is to determine how programmers would use a virtual assistant to assist them in fixing a source code bug. We seek to see what types of questions programmers would ask a virtual assistant and if those types of questions are consistent across multiple programmers.

RQ_1 Do different programmers ask the virtual assistant similar questions for the same bugs?

RQ_2 What types of questions did programmers ask during bug repair?

RQ_3 What type of questions did programmers most frequently ask?

The rationale behind RQ_1 is that if programmers ask for help, and if they ask similar questions for the same bug, it is possible to create a dialogue act classification system given training data. We group the questions to create labels for the training data in RQ_2. Finally, we investigate the most common uses of a potential virtual assistant in RQ_3 to advise future virtual assistant implementations.

3.2.2 Methodology

We used a manual open coding qualitative analysis process [12] adapted from the social sciences to create labels for the conversations we collected (though for the purposes of this thesis, we follow Rastkar *et al.* [74] in referring to "coding" as "annotating" to prevent conceptual conflicts between sociological coding and computer coding). Qualitative annotation is becoming more common in software engineering literature [85, 66, 73, 26, 43], and it is important to recognize that while standards and principles exist, the nature of qualitative analysis is that each annotation procedure is slightly different based on the needs and circumstances of the study. In this thesis, we followed the recommendations of Rieser and Lemon in a 2011 book on creating dialog systems from WoZ data [79], with one exception noted below.

A metaphor for open coding is unsupervised learning, in that the human annotators do not begin with a set of labels: our goal is to discover those labels from the data, and then assign them to the turns in our data. Practically speaking, we did this in three rounds. The first round of annotation consisted of "label creation" where we created labels as we saw fit and did not have a pre-determined list to choose from. The second round consisted of "label pruning" where we decided what labels could be safely removed or merged. The second round became necessary the more progress was made in the first round, and was due to the complexity of compressing sometimes vague and complex english text down into its major concepts. The result of label pruning was a set of well defined and disjoint descriptions of english text describing our examples. The third and final stage of annotating involved re-examining the annotations but instead searching for spelling errors or other

small mistakes. This round had the effect of ensuring labels were consistent and resolving labels that represented the same concept but used different terminology (i.e. synonyms), or were spelled incorrectly.

During any annotation process, and especially an open process in which we do not begin with labels, the bias of the human annotator becomes a major concern. The degree of bias is known as the "reliability" of the data, and it is an extremely controversial research topic. One possibility is to follow the lead of Carletta [19] in calculating Kappa agreement from multiple annotators, and only accepting agreement above a certain threshold; if agreement cannot be achieved, the argument goes, then more annotators are necessary. While this is a common procedure, it is not universally accepted. As Craggs and McGee Wood point out, "one must decide for oneself, based on the intended use of [an annotation] scheme, whether the observed level of agreement is sufficient" [24]. Likewise, they "suggest that if a coding scheme is to be used to generate data from which a system will learn to perform similar coding, then we should be 'unwilling to rely on imperfect data'."

At the same time, it is not an option to merely add more and more annotators until agreement is achieved. There has long been a recognized split between expert and naive annotators [19, 69]. It is not proper to allow naive annotators to have majority rule over the experts. To be an expert annotator in our study, a person would need to have 1) knowledge of the bugs solved in our study so they can understand the conversations, and 2) not been a participant in the study. Only the first and second authors were both qualified and available (manual annotation is weeks of effort).

Rieser and Lemon faced a similar situation, and solved it by having a discussion between two annotators for all disagreements, followed by independent decision-making and calculation of Kappa (page 110 of [79]). We differ from this procedure in that we consider our situation to be more "unwilling to rely on imperfect data" due to the fact that our research questions in Section 3.2.1 and our prediction training in Section 3.4 could be affected by errors. Therefore, for this thesis, we had two experts annotate all data and

solve every disagreement through discussion as disagreements occurred, followed by mutual decision-making, resulting in one set of annotations. While this mutual process makes it impossible to calculate a reliability metric, we felt it was more important to maximize correctness of the annotations.

3.3 Annotations Results

In this section, we present the results from our annotation process. We also provide annotation examples following the results. We note that the programmers asked on average 12.8 questions throughout the two hour user simulation study. A select few did not ask more than three, however, these participants were outliers. The highest number of questions asked during a user simulation study was 54 and the lowest number of questions asked during a study was 3.

3.3.1 RQ_1: Programmers Asking Similar Questions

We found that programmers asked similar questions to one another. Of all the questions asked by the programmers, the ones that were consistent across the majority of participants included confirmationQuestions, clarificationQuestions, and apiQuestions. Of these three types of questions, clarificationQuestion was asked the most by all programmers. It was asked a total of 204 times, which comprised 53.1% of all questions asked by programmers. There were various types of clarification questions asked. Some of the clarification questions included questions about what the bug report said, what questions Madeline could and could not answer, and clarifying answers from Madeline. The participants also asked clarification questions to confirm their understanding of the task that they were to complete.

3.3.2 RQ_2: Types of Questions Being Asked

We found that programmers asked a variety of questions that ranged from system type questions to API and documentation types. An example of an API question is:

```
"What methods are in eventyhandler(?)"
```

We also found many programmers asked implementation questions:

```
"What are valid values for the int direction in
 PacPlayer.java?"
```

After finishing the annotation process, we were able to narrow down the question annotation types into 10 categories. The categories are: syntax, parameter, documentation, API, clarification, implementation, bug report, confirmation, clarification, and system questions. Figure A.1 lists the number of occurrences for each of the dialogue act types. In Section 3.3.4 we go into detail with a short example of an annotated conversation. We also provide all of the annotations on our online appendix (see Section 5.3).

3.3.3 RQ_3: Most Frequent Questions Being Asked

We found programmers asked a few questions significantly more than others. In Figure A.1, the dialogue act type "statement" has the most occurrences. We would like to point out that there was another, more popular type of question; the "setup" dialogue act. Since this dialogue act type is not relevant to the study itself, this dialogue act type was removed from our corpus. "clarificationQuestion" has the highest occurrence out of any question type. This label appeared 204 times throughout all 30 transcripts. Many of the participants asked clarification questions about the bugs and about the responses Madeline gave. Madeline asked clarification questions as well when we needed more information from a participant to answer a question. Sometimes, the participants would ask questions that needed more detail in context so that Madeline could answer the question. The second highest occurrence annotation label for a question type was "APIquestion." This label occurred 94 times in the transcripts. This makes sense as programmers were not allowed to use the internet during the bug repair task, were unfamiliar with the given source code, and were forced to rely upon Madeline to explore the api space of each program.

3.3.4 Annotation Examples

We annotated over two thousand dialogue acts during the annotation process. To further explain the previous sections, we provide an example of one of the annotations. Throughout the data, participants asked API questions, documentation, and implementation questions. Below is a section of a developer conversation. This section of the conversation includes implementation questions and clarification questions. At the end of each dialogue act, there is the annotation label for that dialogue act. The annotation is in bold text and is in brackets. The dialogue acts begin with "P" or "M" denoting the speaker as a "participant" or "Madeline - Virtual Assistant" respectively.

P: So the bug is that the PacPlayer does not face right when the key is released, but it is supposed to?
[clarificationQuestion]

M: Yes. He also disappears. **[clarificationAnswer]**

P: Does he disappear because the alive bool is set to false at the wrong time **[implementationQuestion]**

M: I am unsure **[unsureAnswer]**

Figure 3.2: An annotated exchange between a participant and Madeline VA observed and segmented during our studies.

Throughout the annotation process, we found similar results to the previous example. However, we found programmers asked varying amounts of questions throughout the bug repair task. This was evident once deep into the annotation process. It appeared that the more senior a participant was, the less the participant asked for help from the virtual assistant. There are three interpretations we derive from these observations. First, the programmers possibly did not want to ask for help and instead wanted to solve the bug without help. Second, it is possible that the programmers did not feel comfortable asking questions. Finally, the programmers may have assumed that there was no automated virtual assistant and, therefore, did not ask questions.

We found that programmers often made a statement before asking a question. It appeared the participants were explaining their thought process before asking a question. This occurred about 20% of the time in the user simulation studies. An example of this is below in figure 3.3.

> participant: first I tried "sudo apt-get install default-jre"
>
> participant: it told me it depends on default-jre-headless and openjdk-7-jre
>
> participant: is it possible to set a command line argument for start up of the program?

Figure 3.3: Programmers often offered extra information outside their question through statements.

Here, the participant makes multiple statements before asking Madeline a question. We did not ask participants to "think aloud" during this study. However, we observed this phenomenon throughout the user simulations and annotation process.

3.4 Predicting Dialogue Act Type

Our approach for predicting the dialogue act type is, essentially, a text classifier based on Logistic Regression. Recall the use case that we envision in Section 2.1: a virtual assistant receives a message, and needs to classify that message into one of several categories, so that it can respond appropriately. Our idea is to train a prediction model, then use that prediction model to classify incoming messages.

3.4.1 Labeled Training Data

Supervised machine learning algorithms depend on labeled training data. We use the labels from Section 3.3.2. In that section, we manually annotated every turn in every conversation as belonging to one of the dialogue act types we identified. In this section, we use that data (however, only turns from the participants' side of the conversation, not "Madeline's", to match the use case of the virtual agent classifying incoming messages) to train

a classifier to annotate the turns automatically. Note that this is a multi-label classification problem, because an "example" consists of a turn annotated with a list of all the dialogue act types to which that turn belongs. Each dialogue act turn type is a label, so each turn may belong to many labels.

3.4.2 Attributes

We use two types of attributes. First, we treat the problem as text classification, where each word is an attribute. We calculate the attributes as a binary "bag of words". For each example, the set of attributes includes either a one or zero for each word, depending on if that word occurs in the text of the turn or not. Recent industry-track papers [5, 25] came to the conclusion that to maximize potential industrial impact, researchers should prioritize simplicity, and only move to more complex approaches when absolutely necessary. We stuck to binary bag of words for this reason. We also did not do stop word removal or stemming. We defer word count normalization (e.g. TF/IDF), NLP-based solutions, advanced preprocessing techniques, etc., to our future work. As we will explain in Section 3.6.1, the simple approach already achieves reasonable performance.

Second, we used three shallow features identified by related literature [84, 74, 66]. While this related literature actually identifies over twenty shallow features that complement or replace text classification, the majority are not applicable in our context. For example, many rely on "post-hoc" information such as computing entropy over an entire conversation, which is impossible in our real-time context. The three features we used are: `slen`, the number of words in the message normalized over all previous messages, `wc`, the number of words not normalized, and `ppau`, the number of seconds between the message and the previous message. To account for the variability that `ppau`, we converted this continuous feature into a nominal one by "binning" each value into one of 5 bins whose partition values we determined empirically by looking at the distribution of our `ppau` values.

3.4.3 SMOTE

We use SMOTE [21] to overcome the problem of unbalanced data. Some of the user dialogue acts we identified only have a few examples (e.g. we only found eight examples for the `parameterQuestion` type). That presents a problem because the learning process will inevitably classify no turns in that type, and still seem to achieve very high accuracy. SMOTE works by synthesizing examples in small classes from the known examples in those classes. The result is that the small classes are filled with synthesized examples until the data are balanced. SMOTE has been widely used to resolve situations of unbalanced data generally as well as conversational analysis [84]. In pilot studies, we compared SMOTE to duplicative oversampling and observed slight performance improvements using SMOTE. We used SMOTE only on the training data, to avoid biasing the testing set.

3.4.4 Prediction Models

We trained a multi-label prediction model using the *binary relevance* [75] procedure. The procedure is to create one binary classifier for every class. We used the Logistic Regression (LR) algorithm [45] to create each classifier. We also tested Naive Bayes (NB) and Support Vector Machines in pilot studies – LR had superior performance to NB, and the difference between LR and SVM was so slight as to not be worth the much longer training time for SVM (eight hours versus four minutes). Note that while we built a multi-label prediction model, we calculated SMOTE using a multi-class structure. That is, we ran SMOTE once for each category, then trained each classifier, then combined the classifiers with the binary relevance procedure. In theory it is possible to run SMOTE in a multi-label configuration, by executing SMOTE on every combination of labels. However, this would necessitate n^n runs of SMOTE (for n categories), which would be far more expensive.

We also performed parameter tuning for LR across twelve parameters and NB across four parameters. Parameter tuning has been recommended generally when working with SE data [13]. We direct readers to our online appendix for complete details (Section 5.3).

3.4.5 Implementation Details

We used the toolkit `scikit-learn` [71, 70] to implement our classifiers and SMOTE (`imblearn.over_sampling.SMOTE`) [57].We implemented the shallow attribute calculators ourselves, using related work as a guide [84]. The hardware was an HP Z640 workstation with an E1630v3 CPU and 64GB of memory. For total clarity, we make all implementation scripts and datasets available via our online appendix (see Section 5.3).

3.5 Evaluation of Predictions

This section describes our evaluation of the prediction models we create. Essentially, we use a 5-fold cross validation procedure to test the quality of the predictions, as well as explore where the predictions are most accurate.

3.5.1 Research Questions

Our research objective is to determine what level of performance we can expect from the prediction models, as well as to understand which dialogue acts are "easiest" to detect.

RQ_4 What is the performance of our prediction models, overall in the multi-label configuration, according to the metrics described in Section 3.5.3?

RQ_5 For which dialogue acts do the prediction models have the highest performance?

RQ_6 Which attributes are the most informative?

The rationale behind RQ_4 lies in the application we intend in Section 2.1: the performance of a virtual assistant will be limited by its ability to detect what type of dialogue act to which an incoming message belongs. While we do not expect perfect performance, we need to at least have an understanding of how much inaccuracy may stem from the detection process. The rationale behind RQ_5 is similar. Some dialogue acts are bound to be easier to detect than others. It is helpful to know which dialogue acts about which we may be confident, or others where we are less sure. In practice, it may be necessary to return a message to the user indicating that the virtual assistant is unsure what the user intends, and

ask the user to clarify. RQ_6 is useful because the presence of some attributes may indicate high confidence, while others may indicate low confidence.

3.5.2 Methodology

In general, we follow a 5-fold cross validation study design. In a standard n-fold design for evaluating classifiers, $1/n$ examples are set aside as a testing set, while the remaining $(n-1)/n$ examples are used for training. The evaluation is conducted n times, once for each nth selection of the examples as a testing set. Then, the evaluation metrics are calculated for each "fold" and averaged. We chose 5 as a value for n because it ensured that our testing set would not be too small (as it might have been with a 10-fold design), while still maintaining multiple folds that could be averaged.

The selection of a testing set is a non-trivial exercise in a multi-label dataset, in contrast to a single-label one. In a single-label dataset, it is usually sufficient to randomly selected $1/n$ of the examples for the testing set. But in our multi-label dataset, we need to ensure that the testing set represents the same distribution of labels as the overall dataset. With only five folds, it is conceivable that a random selection would give too much weight to one label, and this overweighted selection would not be "averaged out" over a large number of folds. Therefore, we sampled each label in proportion to the number of examples in that label, and confirmed that the distribution of the labels over the testing set was as close as possible to the distribution of labels over the entire dataset.

After we separated the testing and training data, we ran SMOTE on the training data only. If we had executed SMOTE on the entire dataset, then divided the data into testing/training groups, we would have contaminated the testing set with information from the training set. SMOTE synthesizes examples based on the examples it is given. If we had run SMOTE on the entire dataset, we would have created synthesized examples based on real examples that ended up in testing set. Therefore, we only ran SMOTE on the training set. This did increase the execution cost of our experiment slightly, since we needed to execute SMOTE five times (once for each fold, after we separated testing and training data).

Note also that this methodology is conservative – it only uses real examples for the testing set. We use the results from this conservative approach to answer RQ_4 and RQ_5, to avoid bias. We also use these results to calculate other metrics to answer RQ_6.

3.5.3 Metrics

We report the metrics precision, recall, F-measure, and support to answer RQ_4 and RQ_5 These are standard metrics for evaluating classifiers and have been covered extensively elsewhere [10, 20]; for space we do not discuss their details here. We calculate these metrics for each dialogue act type (i.e., each label) for RQ_2, and combine the results for each dialogue act type to answer RQ_4. We combine the precision and recall values for each dialogue act type with a weighted average, where the weights are based on the support for each dialogue act type. The reason is so that the combined values reflect the size of each label. A simple average, without the weights, would be biased by labels that only have a few examples in the testing set.

For RQ_6, we calculate F-score [80] for the attributes. F-score (distinguished from F-measure, the harmonic mean of precision and recall) is typically used for feature selection, to indicate which features are the most informative.

3.5.4 Threats to Validity

Like all experiments, our study carries threats to validity. The main sources of threats include: the participants in the user simulations, the bugs we asked the users to repair, and the influences of the technology used by the participants (e.g., the IDE) on the questions they asked. Also, it is possible that there are errors in our manual annotation process, or in our selection of categories. While we try to mitigate these risks by following accepted data collection and annotation procedures, and by including a relatively large number of participants (30) and different bugs, the threat remains that changes in these variables could affect the performance of our classifiers. As an additional guard against these risks, we release all data via an online appendix for community scrutiny (see Section 5.3).

3.6 Prediction Eval. Results

This section discusses our answers to RQ_4-RQ_6, including our supporting data and rationale.

TABLE 3.1
PERFORMANCE METRICS CALCULATED FOR EACH DA TYPE.

	precision	recall	f-measure	support
apiAnswer	0.93	0.76	0.83	24.6
apiQuestion	0.81	0.66	0.71	17.2
clarifAnswer	0.13	0.07	0.09	6.0
clarifQuestion	0.59	0.41	0.48	32.6
confirmation	0.88	0.8	0.83	27.0
docAnswer	0.25	0.2	0.22	3.2
implQuestion	0.52	0.21	0.28	10.6
implStatement	0.0	0.0	0.0	3.0
introduction	0.76	0.6	0.63	4.0
stmnt	0.69	0.4	0.51	49.8
systemQuestion	0.37	0.22	0.27	4.8
avg / total	0.69	0.5	0.57	16.62

Note that some types have been abbreviated. Recall that the averages are a weighted average based on the support for each dialogue type, see Section 3.5.3.

3.6.1 RQ_4: Overall Performance

The weighted average precision of from our classifiers was 69%, while the weighted average recall was 50%, as reported in Table 3.1. Thus for an arbitrary incoming message, we can expect this classifier to correctly identify the dialogue act type of that message 69% of the time, while identifying 50% of the dialogue acts types to which the message belongs. If the classifier claims that a message of a particular type, we can estimate that that claim will be correct roughly 2/3rds of the time. We acknowledge that we cannot evaluate whether these improve over an earlier approach, given that we are not aware of an earlier technique for identifying dialogue acts on our data. Nevertheless, we find these results to be an encouraging starting point for building a virtual assistant, in light of the somewhat bare bones bag of words (see Section 3.4) text classification strategy we used. A promising area of future work, in our view, is to adapt more advanced classification techniques.

3.6.2 RQ_5: Dialogue Act Type Variations

The performance of our classifiers varied considerably across different dialogue act types. At the high end, precision was over 90% and recall over 75%. At the low end, precision and recall dipped below around 10%. This observation is important because it means that for some dialogue act types, a virtual assistant can be highly confident that the prediction is correct. As a practical matter, a virtual assistant may request the user to repeat a message in different words, or ask for other followup information if the classifier is not able to place the message into a dialogue act type with sufficient confidence. This observation is also important from an academic viewpoint, because it means that programmers use different types of language to make different types of messages. In some cases, programmers consistently use the same language (ideal for classifier predictions). In other cases, programmers use much more different language – it makes the prediction process more challenging, but also raises academic questions about what is different about the language, which is an area of future work. We begin to explore this in RQ_6.

3.6.3 RQ_6: Attribute Effects

Table A.1 shows the top-10 most informative features for each dialogue act type. We make two observations from this data: First, the shallow features are far more useful for some dialogue act types than others. For example, confirmation actions are likely to be short messages, so the word count metric (`wc_sf`) is informative in this case. This observation is useful because shallow features are easy to compute, so areas where they are informative can be predicted with reasonable accuracy at low cost. Second, many of the words are general enough that they are likely to be generalizable beyond the set of bugs we chose, even though others are specific to particular domains. For example, the dialogue act `implementationStatement` is informed by words like "function" and "signature", which are likely to be true across many programming conversations. But the most informative feature for that action is "jcomponent", which is a word specific to Java and perhaps the domain of programs we study. It is not likely to appear in every domain.

Therefore, one possible mediation is to use placeholder features that count the number of e.g. domain-specific programming words used in a message. Also, we note again that we used the binary bag-of-words model, which separates the words from their contexts. An area of future work is in NLP-based recognition such as phrases or n-grams.

CHAPTER 4

THE BANTER PROJECT: LOW DATA TRANSFER LEARNING NEURAL MODELS

This chapter, discusses the transfer learning approach used to boost dialogue act classification performance on the Madeline dataset. This chapter begins with sections describing the experimental setup, methodology, and model architecture, with subsequent sections covering model performance and analysis.

4.1 Model Design

This section describes our model design. The key insight to our model is to use what we call a "dual encoder" design in which one encoder is trained using a large dataset of generic business conversations, while another is trained only on the small number of conversations available in the target domain. Our target domain is software engineering debugging virtual assistants as covered above. The large generic dataset we learn from is the AMI business conversation set provided by McCowan *et al.* [61].

4.1.1 Overview

An overview of our model is in Figure 4.1. At a high level, it resembles many attentional encoder/decoder NMT systems, though many practical details differ. First, during training, we provide a "global" encoder (area 1) with every word sequence for every dialogue act in the large, generic business dataset. We train only that encoder based on the label for that dialogue act (area 2). We save the weights for the encoder to disk. We then reload the weights for the global encoder and set it as not trainable, and then train the local encoder (area 3) on the much smaller, SE-specific dataset.

Note that we send the same word sequence to both the global and local encoder while training the local encoder. The global encoder is not trainable at this point, and therefore creates a representation based on the generic dataset alone. The local encoder's initial state is equal to the final global encoder state (area 4). Then we attend each state in the local encoder to states in the global encoder (area 5). The intended effect is to identify states where the global and local encoders are similar (i.e., where global knowledge is relevant to the local situation) while attenuating other states (i.e., less applicable global knowledge).

We create a context vector by concatenating the attended global states to the local states (area 6). We then use one fully-connected layer per concatenated state, to help decide how to combine the global and local state (i.e. which should be used for classification, area 7). Then we flatten the output of these layers into a single vector and prepare for the final classification (area 8).

During inference, we provide the word sequence to both the global and local encoder, discard the output at area 2, and use only the output at area 8 as the prediction.

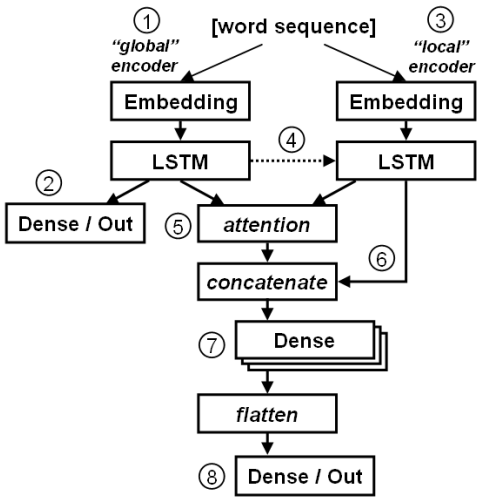

Figure 4.1: Overview of our model. See text in Section 4.1.1 for discussion of labeled areas.

4.1.2 Model Details

Given a corpus of utterances $C = \{\mathbf{u}_j\}_{j=0}^{k}$ composed of k utterances, where each utterance is a sequence of words $\mathbf{u}_j = \{w_{jn}\}_{n=0}^{l}$ that has a maximum utterance size l, we can construct a vocabulary V_C for corpus C such that $|V_C| = m$ and contains the top m most frequent words in C. We define our DA classification problem $\mathbf{f}$ as predicting a single DA p_i for d DA types given an entire utterance $\mathbf{u}_i$. We construct the solution by defining the decision function as computing the probability distribution over all potential d DA types given an utterance $\mathbf{u}_j$:

$$\mathbf{f} = \{Pr(p_i|\mathbf{u}_j)\}_{i=0}^{d} \tag{4.1}$$

Specifically, our base models compute this decision function using an embedding function $\mathbf{e}$ to map each word into a continuous vector space $\mathbf{e} : w_i \in V_C \rightarrow \mathbb{R}^{1\times n}$, a Long-Term-Short-Memory (LSTM) Recurrent Neural Network (RNN), and an output softmax layer. Formally, we can divide this decision function into two discrete parts: the **encoder** containing the embedding function and RNN encoder with q encoding dimensions, and the **predictor** containing of just the softmax layer. Given learnable encoder parameters $\mathbf{\Phi}_{enc}$ and learnable predictor parameters $\mathbf{\Phi}_{pred}$, the base decision function can be expressed as (for an utterance $\mathbf{u}_j$):

$$\mathbf{h}_{enc,t}, \mathbf{c}_{enc,t} = f_{enc}(\mathbf{h}_{enc,t-1}, \mathbf{c}_{enc,t-1}, \mathbf{e}(\mathbf{u}_{jt}), \mathbf{\Phi}_{enc}) \tag{4.2}$$

$$\mathbf{f} = softmax(\mathbf{h}_{enc,|\mathbf{u}_j|}, \mathbf{\Phi}_{pred}) \tag{4.3}$$

where $softmax(\mathbf{v})_k = \frac{e^{\mathbf{y}_k}}{\sum_j e^{\mathbf{y}_j}}$ returns a probability distribution, and $\mathbf{h}_{enc,t}$ and $\mathbf{c}_{enc,t}$ are the LSTM encoding and context of the sequence through time t.

Now, assume there are two corpi, a "general" corpus C_1 and a "local" corpus C_2, which are different corpi, have separate vocabularies V_{C_1} and V_{C_2} that share common words $V_{overlap}$, and are different sizes. Specifically, the general corpus is significantly larger than

the local corpus, or $|C_2| \ll |C_1|$. We define four decision functions designed to minimize cross entropy loss over the local corpus.

Our first decision function is training the architecture described above solely on the local corpus, which can directly be expressed by equations (2) and (3), and serves as our baseline.

Our second decision function is trained on the local corpus immediately after training on the general corpus. This can be expressed as conditioning the function for the local corpus $\mathbf{f}_2$ with the function learned for the general corpus $\mathbf{f}_1$. Since these architectures share all learnable parameters, we can express $\mathbf{f}_2$ as:

$$\mathbf{h}_{2,enc,t}, \mathbf{c}_{2,enc,t} = f_{2,enc}(\mathbf{h}_{2,enc,t-1}, \mathbf{c}_{2,enc,t-1}, \mathbf{e}(\mathbf{u}_{jt}), \mathbf{\Phi}_{2,enc} | \mathbf{\Phi}_{1,enc}) \tag{4.4}$$

$$\mathbf{f}_2 = softmax(\mathbf{h}_{2,enc,|\mathbf{u}_j|}, \mathbf{\Phi}_{2,pred} | \mathbf{\Phi}_{1,pred}) \tag{4.5}$$

Our third decision function is defined over the local corpus $\mathbf{f}_2$ conditioned upon the encoding function of for the general corpus $\mathbf{h}_{1,enc,t}$. It contains two separate LSTM RNNs that do not share trainable parameters. In fact, the general encoder cannot be trained when the local encoder is trained on the local corpus. The initial state of the local encoder is the output state of the general encoder that encoded the same input sequence:

$$\mathbf{h}_{1,enc,t}, \mathbf{c}_{1,enc,t} = f_{1,enc}(\mathbf{h}_{1,enc,t-1}, \mathbf{c}_{1,enc,t-1}, \mathbf{e}_1(\mathbf{u}_{jt}), \mathbf{\Phi}_{1,enc}) \tag{4.6}$$

$$\mathbf{h}_{2,enc,t}, \mathbf{c}_{2,enc,t} = f_{2,enc}(\mathbf{h}_{2,enc,t-1}, \mathbf{c}_{2,enc,t-1}, \mathbf{e}_2(\mathbf{u}_{jt}), \mathbf{\Phi}_{2,enc} | \mathbf{h}_{1,enc,|\mathbf{u}_j|}) \tag{4.7}$$

$$\mathbf{f}_2 = softmax(\mathbf{h}_{2,enc,|\mathbf{u}_j|}, \mathbf{\Phi}_{2,pred}) \tag{4.8}$$

The decision function first encodes the entire utterance $\mathbf{u}_j$ using the encoder for the general corpus $f_{1,enc}$, and then encodes $\mathbf{u}_j$ again using the local encoder conditioned on the global encoder by using the final states of the global encoder as the initial states of the local encoder.

Finally, our last decision function includes extra parameters that allows the local encoder to attend to the general encoder states. The local and general encoder relationship is the same as described in the previous decision function, however an attention mechanism is introduced between the encoder and predictor sections to allow an overall decision to be made after examining the history of encodings produced by both encoders:

$$\mathbf{h}_{1,enc,t}, \mathbf{c}_{1,enc,t} = f_{1,enc}(\mathbf{h}_{1,enc,t-1}, \mathbf{c}_{1,enc,t-1}, \mathbf{e}_1(\mathbf{u}_{jt}), \mathbf{\Phi}_{1,enc}) \tag{4.9}$$

$$\mathbf{h}_{2,enc,t}, \mathbf{c}_{2,enc,t} = f_{2,enc}(\mathbf{h}_{2,enc,t-1}, \mathbf{c}_{2,enc,t-1}, \mathbf{e}_2(\mathbf{u}_{jt}), \mathbf{\Phi}_{2,enc}|\mathbf{h}_{1,enc,|\mathbf{u}_j|}) \tag{4.10}$$

$$\mathbf{w}_{attn,t} = softmax(\{\mathbf{h}_{2,enc,t} \cdot \mathbf{h}_{1,enc,s}\}_{s=0}^{|\mathbf{u}_j|}) \tag{4.11}$$

$$\mathbf{c}_{attn,t} = \mathbf{w}_{attn,t} \cdot \{\mathbf{h}_{1,enc,s}\}_{s=0}^{|\mathbf{u}_j|} \tag{4.12}$$

$$\mathbf{a}_{attn,t} = tanh([\mathbf{c}_{attn,t}; \mathbf{h}_{2,enc,t}], \mathbf{\Phi}_{attn}) \tag{4.13}$$

$$\mathbf{f}_2 = softmax([\mathbf{a}_{attn,1}; \ldots; \mathbf{a}_{attn,|\mathbf{u}_j|}], \mathbf{\Phi}_{2,pred}) \tag{4.14}$$

This decision function will collect all general and local encoder states for a sequence, perform a simple attention mechanism between the two sequences of states, and use the flattened output of the attention mechanism as the input to the prediction softmax layer.

We direct readers to our online appendix (Section 5.3), file `dualencattendlstmwe.py`, for an implementation in Keras.

4.2 Data Preparation

We used two sources of data for this thesis: 1) the AMI business meeting corpus [61], and 2) the Madeline corpus described in Chapter 3. The AMI corpus has been well-organized by its creators, and includes word sequences labeled with dialogue act types for all conversations. The dialogue act types are "generic" in that they are applicable to a wide variety of conversations.

1. Backchannel
2. Stall
3. Fragment
4. Inform
5. Elicit-Inform
6. Suggest
7. Offer
8. Elicit-Offer-Or-Suggestion
9. Assess
10. Comment-About-Understanding
11. Elicit-Assessment
12. Elicit-Comment-About-Underst.
13. Be-Positive
14. Be-Negative

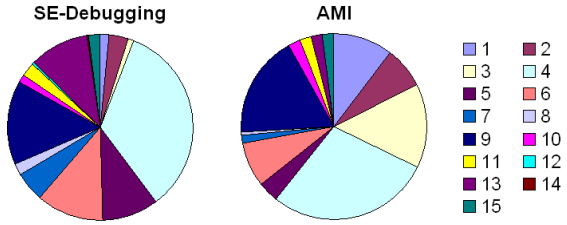

Figure 4.2: Dialogue act types from AMI and the composition of our two datasets. Note that in our experiment we collapse the three smallest classes into the 15^{th} "other" class due to their tiny size in the SE-Debugging data.

The AMI corpus was both *segmented* and *annotated* by its authors. The authors provide a thorough guide, but in brief, the segmentation process involved human evaluators reading transcripts of the conversations and selecting a continuous sequence of words by one speaker, and then annotating that sequence with one dialogue act type.

But the Madeline corpus was segmented using simple block segmentation in which each turn in the conversation was considered one segment, and each turn received one or more dialogue act types. Plus, the set of dialogue act types were specific to the SE domain, with types such as "API Question" and "Implementation Answer." The segmentation and annotation processes must be identical for our transfer learning approach. So, we hired two human evaluators to segment and annotate all 30 conversation transcripts in the Madeline corpus using the same procedures published for AMI (including the same instruction documents). Both annotators did have at least two years programming experience.

We observe that the composition of dialogue acts in the datasets is quite different, as depicted in Figure 4.2. Some dialogue acts are in roughly the same proportion, such as #4 Inform and #9 Assess, but others such as #5 Elicit-Inform are far more common in the SE domain-specific dataset. This observation is important because, in practice for a

virtual agent, some dialogue act types are much more important to recognize than others (see Section 2.1 above). In particular, we identify the following five dialogue act types are the most significant ones for a SE/debugging VA assistant to classify correctly:

5. Elicit-Inform
8. Elicit-Offer-Or-Suggestion
10. Comment-About-Understanding
11. Elicit-Assessment
12. Elicit-Comment-About-Understanding

These are the most important for our use case for two reasons: First, these cover the questions asked by programmers of a virtual agent. We do not care if a system can classify text generated by the agent itself, because the agent will already know what its own dialogue act type is. Also, mistakes made in classifying commentary as e.g. Be-Positive are easily recoverable by the VA in a real conversation, while mistakes in classifying questions can degrade trust in the system to answer those questions properly. Furthermore, a second reason is that these dialogue act types closely overlap with the key dialogue act types uncovered in chapter 3 (see section 3.3) related to the questions programmers ask (such as Elicit-Inform and API Question).

4.3 Experiment

This section describes our experiments, plus relevant metrics and baselines.

4.3.1 Research Questions

Our research objective is to assess the performance of our model in comparison to competitive baselines. To that end, we ask the following Research Questions (RQs):

RQ_1 What is the baseline performance training and testing with AMI data?

RQ_2 What is the baseline performance training with AMI data and testing with the SE debugging dataset?

RQ_3 What is the baseline performance training and testing with the SE debugging dataset?

RQ_4 What is the our model's performance training with AMI and the SE data, and testing with the SE dataset?

The rationale behind RQ_1 is to establish a "reasonable expectation" for performance given state-of-the-art tools and a large dataset. A key goal of our model is to transfer knowledge learned on a large generic dataset to a small, domain-specific dataset, and one simple transfer approach is to just train on the large dataset. Another straightforward approach is to train only on the limited domain-specific data we have available. Our model will need to improve over these approaches to justify the added complexity of our model, so we ask RQ_2 and RQ_3 to establish baseline performance and RQ_4 to compare our model to that performance.

4.3.2 Methodology

Our methodology to answer RQ_1 is to randomly split the conversations from AMI into training/validation/test sets (80%, 10%, 10%), then place all dialogue acts from training conversations into a training set, etc. That way, dialogue acts from the same conversations would not end up in both the training and test sets. We then train each baseline using the training set (max epoch 1000, early stopping patience 50) and report results from the test.

For RQ_2 and RQ_4, we combine the training and test sets of AMI and use both together to train the baselines and global encoder portion of our model. This combination maximizes the amount of available data, without compromising evaluation results because the AMI test set is only used to establish expected performance ceilings in RQ_1. We still hold the validation set aside for model selection (again, we use max epoch 1000, early stopping patience 50).

For the SE debugging dataset, we were especially interested in evaluating the performance on very small dataset sizes, so we created ten random subsets containing different numbers of conversations: 3, 6, 9, 12, 15, 18, 21, 24, 27, 30. There are 30 conversations in the Madeline dataset; the subsets we create range from 10 to 100% of the data.

Then we further create ten random training/val/test splits of each subset, resulting in a total of 100 datasets based on the full 30 conversations. The splits contained 70% for training, 10% validation, 20% testing. However, we enforced a minimum of one conversation

in validation and testing, so for very small numbers of conversations, there may be fewer than 70% in the training set.

Finally, for RQ_2, we evaluate the baseline by training on the AMI dataset and testing on each of the 100 test sets from the random splits. For RQ_3, we evaluate the baseline by training/testing on the training and test sets from the 100 splits. And for RQ_4, we evaluate our model training/testing on the 100 splits. We report the average of the ten random splits for at each of the ten sizes of conversation (e.g., average results for all ten splits of 21 conversations, 24 conversations, and so on).

4.3.3 Metrics

We use three standard metrics for evaluation: precision, recall, and F1-score. We weight based on class size using the default `sklearn` settings.

4.3.4 Baselines

We compare our model against two main baselines. The first is the model from Chapter 3. This model uses a bag of words representation augmented by three shallow features computed from the text and a logisic regression decision function (see Section 3.4 for more details). Interestingly, we found that a binary bag of words representation produced superior models than a tf-idf bag of words representation, so specifically we compare our model against a binary bag of words approach. The second baseline we use is based on related work by Khanpour *et al.* [50]: a word embedding and LSTM with the word sequence of the dialogue act as input. Note that we provide the baseline and our approach with the segmented word sequence, since we focus on classification instead of segmentation in this thesis. One configuration of the baseline is to train with the AMI corpus and test with the SE debugging corpus (RQ_2), which represents a standard attempt at transfer learning. Another configuration is to train and test on only the SE debugging corpus (RQ_3). We provide an implementation of this baseline in file `lstmwe.py` in our online appendix (Section 5.3). We keep parameters of the model e.g. word embedding dimensions and LSTM output size

identical to the global encoder in our approach (Section 4.1), to minimize the number of experimental conditions. Our focus is on the transfer learning strategy, and if we change model parameters, we would not know if experimental differences were due to those parameters or the learning strategy.

As discussed in Section 2.2, there are many published approaches for DA classification. Several of the newest are not directly comparable because they are "post hoc" in that they are intended to classify an entire conversation at once, rather than each dialogue act as it occurs like a virtual agent would need to do. A consensus of the others is that a combination of word embedding space and recurrent or convolutional layers leads to good performance, leading to our choice of Khanpour's approach as a strong baseline to evaluate transfer learning strategies for our specific problem area.

One specific case we would like to highlight is the model developed by Ji *et al.* [47]. This model starts with a similar approach to Khanpour *et al.* where they encode an input utterance using word embeddings and an LSTM. They then compute a distribution over the dialogue act types (which is modeled as a latent variable z_t in their paper) given the input sequence $Pr(z_t|\mathbf{u}_t)$. However, Ji *et al.* then improve this distribution by using the distribution and encoding of $\mathbf{u}_t$ to generate the probability of observing the *next utterance* and finally estimate $Pr(z_t|\mathbf{u}_t, \mathbf{u}_{t+1})$ given the two observed utterances.

This model is interesting for us because it falls on the boundary condition for what constitutes a post hoc vs an online approach, as their model uses *only* the next utterance and not the entire conversation as input. Ultimatley, we decided that a conservative definition of what constitues an "online" approach is most appropriate, and that Ji *et al.*'s model does not meet our definition; requiring access to the next utterance (i.e. future information) does not help a VA answer the current utterance, despite the potential performance boost that using the next utterance could give. Therefore, we excluded Ji *et al.*'s model from our experiments.

In pilot studies, we verified a conclusion by Milajevs and Purver [62] that history data does not improve (actually, it reduced) performance when combined with text data, except in post hoc approaches where the context of the whole conversation is available. Therefore, we do not use history data in our baseline.

4.3.5 Threats to Validity

As with any study, our experiment carries threats to validity. These threats include errors in extracting and parsing the word sequences, random factors such as a random dataset split in which the testing set turns out to be "easier" than average, experimental parameters that we chose such as the LSTM output size, and a potential lack of generality of the datasets. While we have taken steps to minimize these threats such as averaging results of ten splits instead of relying on one, it is possible that large changes in any of these threats could lead to different conclusions.

4.4 Experimental Results

In this section we answer our Research Questions as well as provide our rationale and supporting data.

4.4.1 RQ_1: Baseline Expectations

The baseline performance when trained and tested on the AMI corpus is a 72.59% precision, 54.88% recall, and 58.90% F1 score, with most errors related to the "other" category, as visible in Figure 4.5. We view this as a ceiling on performance expectations for the SE dataset. Also, we note that these results were very similar for all dialogue act types and the five target types, unlike in the SE-debugging dataset. We also note that we expect the logistic regression model from Chapter 3 to either outperform all neural models when corpus sizes are extremely small, but for this gap in performance to shrink, vanish, and eventually invert (neural models perform better) as the corpus size increases.

4.4.2 RQ_2: Baseline Transfer Learning

The baseline transfer learning approach's performance is visible as the purple line in Figures 4.3 and 4.4. In terms of all dialogue act types, the baseline transfer learning approach obtains quite lower performance for all dataset sizes six conversations and above. However, a majority of the errors occur when classifying dialogue act types of lower value to us, such as backchannel and stall.

4.4.3 RQ_3: Baseline Local Training

The baseline direct training approach is broadly similar to our approach for all dialogue act types, but does not exceed the baseline transfer learning for the five target types. Of note is that precision on the five target types (Figure 4.4c) begins to exceed our model for the two largest dataset sizes, which may imply diminishing returns for transfer learning at greater dataset sizes. Suprisingly, the baseline direct training approach is a comparable and slightly superior solution to the logistic regression model regarding overall performance, but struggles significantly with recall (and therefore f1 score) in comparison regarding the five target dialogue act types.

4.4.4 RQ_4: Our Model Performance

The strongest area of performance for our model is in recall on the five target dialogue act types (Figure 4.4b), when recall exceeds the baselines for all dataset sizes six conversations or greater and approaches comparable performance to the logistic regression model.

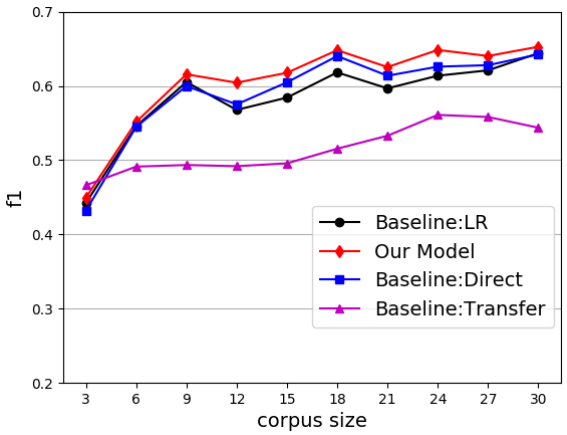

Figure 4.3. F1 scores for all dialogue act types.

A typical pattern is to trade precision for recall, but in our case increased recall does not come with a penalty for precision, leading to higher F1 scores. In fact, our model is has overall comparable performance to the logistic regression baseline despite having several orders of magnitude more learnable parameters. As shown in Figure 4.5, the approach does struggle with the type elicit-comment-about-understanding, probably due to the low incidence of that type. However, a majority of the mistakes for the target DA types are categorized as one of the other four target types, as opposed to the lesser-important ones e.g. backchannel.

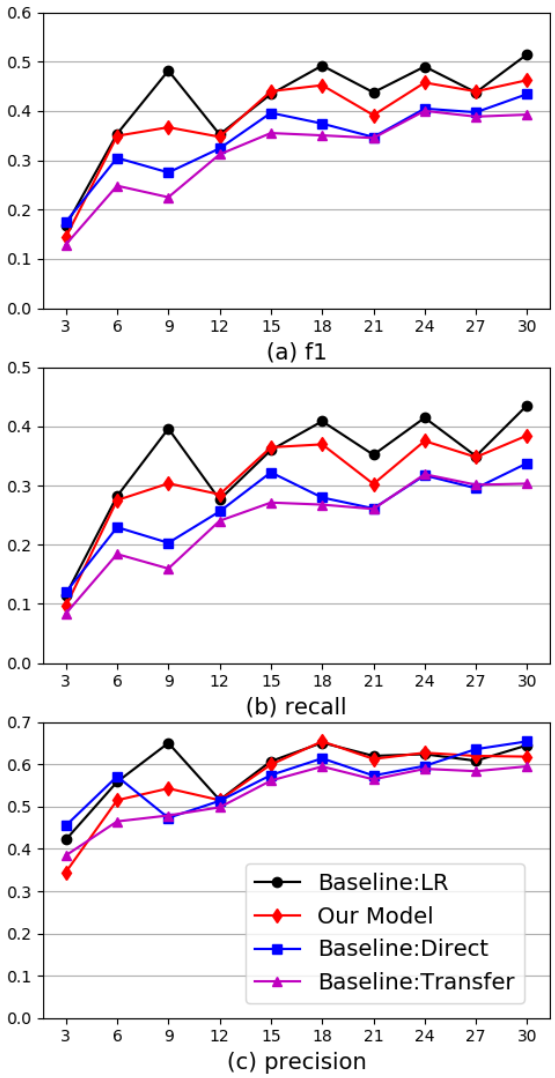

Figure 4.4. Performance metrics for the five most-important dialogue act types discussed in Section 4.2. X-axis is the number of SE debugging conversations.

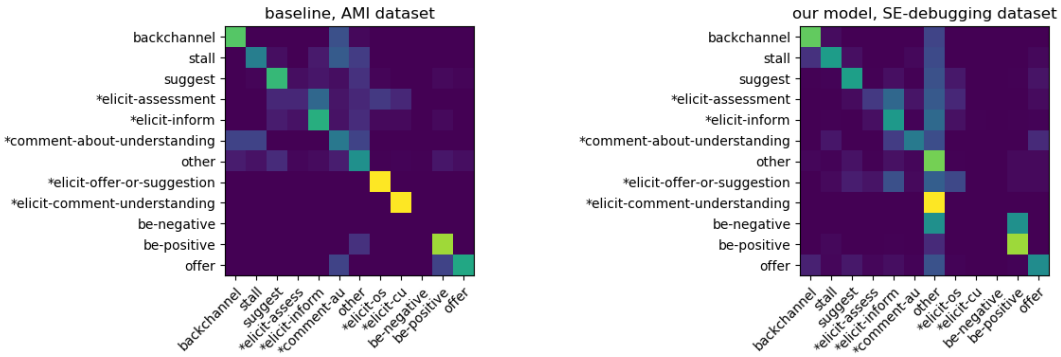

Figure 4.5. Confusion matrix for RQ_1 (top) and RQ_4 (bottom). Note that we have collapsed three tiny classes to "other" (see Table **??**) throughout our experiment.

An important caveat is that the F1 scores for all approaches are about 30% lower for the five target dialogue act types than for all types ($\sim$40% versus $\sim$60%). Nearly all papers on dialogue act classification report accuracy over all dialogue act types, but some types are more important than others in some domains, and some types are much "easier" to detect than others.

CHAPTER 5

CONCLUSION AND FUTURE WORK

This chapter provides a brief summary of the work in this thesis, as well as concluding remarks regarding the contribution this thesis makes to the field of Software Engineering. Additionally, this chapter suggests future directions for this work, links for downloading all resources in this thesis; including the Madeline corpus and all code, and instructions for reproducing all experiments and results.

5.1 Conclusion

We have collected a first-of-its-kind corpus of conversations between professional developers and a simulated virtual agent, and presented a technique for transfer learning in dialogue act classification. First, we simulated a virtual agent named "Madeline" and hired 30 professional software developers from around the globe to fix bugs in a pre-selected set popular Java projects. We then required the participants to interact with Madeline for two hours by typing in the Skype platform, and to use Madeline in liu of typical resources such as the internet. We then manually annotated the transcripts of these conversations using an open coding process to manually discover SE specific DA types using a turn-based segmentation. We then analyzed our collected corpus to answer important questions regarding what kind of behavior virtual agent designers should expect of professional developers when debugging. Finally, we established baseline performance of simple text-classifiers on this corpus using a 5-fold cross validation experiment.

Second, we moved to more advanced models with superior improvement obeserved in the DA classification literature. However, our collected corpus is too small to apply state-

of-the-art models directly, so we developed a transfer-learning approach to "learn what we can" from a large, unrelated corpus, and apply that knowledge to our collected corpus. To minimize the amount of variables in our experiment, we first re-annotated our collected corpus by hiring professional developers who used the same guidelines and DA labels as the large, unrelated corpus. Then, we evaluated our approach against reasonable baselines by performing 10-fold cross validation experiments across 10 different "subsampled corpora" by undersampling our collected corpus. From these 100 experiments, we were able to observe a clear improvement of our technique over our baselines not only on overall performance, but especially on DA types critically important to our context of answering debugging questions.

This thesis makes three contributions to software engineering literature. First, we contribute 30 software engineering conversations with professional developers. Second, we created a system of classification for developer dialogue acts. We manually detect and classify relevant dialogue acts in order to contribute to the understanding of developer question/answer conversations. We also provide this annotation classification system on our online appendix for future researchers to use. Third, we lay the foundation for a virtual assistant by building an automatic dialogue act classification system, and present a transfer learning technique for boosting performance in the common low-data setting.

5.2 Future Work

Dialogue act classification is one of the first tasks that any virtual agent must perform when processing an utterance from a human. Unfortunately, datasets with which to train dialogue act classification systems are rare, and, especially in the case of software engineering, extremely small with respect to the number of parameters in state-of-the-art models. In the immediate future, there are several avenues of research that, based off of this work, would benefit dialogue act classification for virtual agents.

First, this thesis examines transfer learning between two corpi: the AMI corpus, and the self-collected Madeline corpus. These corpi differ from more than the topic of conversation: the AMI corpus was recorded via three people acting our meeting scenarios while the Madeline corpus was recorded from real developers interacting with a simulated virtual agent named Madeline one-on-one. Future research could discover "rules" between the makeup of the local and general corpi in a transfer learning setting. For instance, a better choice of global corpus might be to try and hold all corpi hyperparameters as close as possible to being identical: if the local corpus was recorded between two participants, then best transfer learning results may be obtained when the general corpus also was recorded between two participants, etc.

Second, this thesis assumes static segmentation, i.e. segmentation performed by humans beforehand and immutable at the model level. However, this situation is not realistic for deployed virtual agents; they must also segment incoming utterances as well as classify them. This problem is highly difficult, as segmentation affects DA classification, and DA classification affects segmentation. One promising avenue of research is to therefore consider models that jointly segment *and* perform dialogue classification at the same time, and consider information pertaining to one task in the decision of the other (i.e. consider the "best" DAs given to an utterance while segmenting utterances, and to consider the "best" segmentation while making DA classification decisions).

Finally, another promising area of future research is to actively model expected user behavior from the model side by taking into account model responses. For instance, if the virtual agent knows it is going to ask a *question* in the current response, then it is reasonable to shape a prior DA distribution based on that knowledge, and expect a higher chance of classifying a subsequent utterance from the user (after the virtual agent's response) to be a DA type representing *answers*.

5.3 Reproducibility

We have made our raw data, annotations, models, and source code available via two online appendices. The online appendix for the Madeline experiments is located at `https://tinyurl.com/yadfpojd`, while the online appendix for the Banter experiments is located at `https://tinyurl.com/y83v6v39`.

APPENDIX A

ADDITIONAL DATA

TABLE A.1
THE TOP 10 MOST-INFORMATIVE FEATURES FOR EACH DIALOGUE ACT TYPE, CALCULATED BY F-SCORE.

	10	9	8	7	6	5	4	3	2	1
apiAnswer	node	if	onfinished	keyframes	constructor	values	time	timeline	keyvalue	keyframe
apiQuestion	size	method	have	how	pane	class	object	an	does	what
clarifAnswer	compilation	configurations	word	trigger	supply	green	appear	box	clicking	bottom
clarifQuestion	need	the	or	other	wc_sf	you	this	fix	prime bug	
confirmation	of	yes	is	to	thanks	slen_sf	the	thank	wc_sf	ok
documentAnswer	byte	marks	later	reading	bytes	joptionpane	external	input	audio	stream
implementQuestion	face	why	mark	eratosthenes	occurs	gets	arraycopy	reason	button	clicked
implementStatement	signature	widget	funtion	hidden	drawing	waitfor	throwing	paint	timeout	jcomponent
introduction	supervised	programmers	today	hello	human	am	start	hi	study	ready
statement	seems	what	slen_sf	looks	fixed	but	works	was	think	it
systemQuestion	password	there	permitted	lang	running	way	programs	kill	eclipse	how

Most features are words, but features with the suffix `_sf` are shallow features (see Section 3.4.2). See Section 3.6.3 for a deeper discussion of this table.

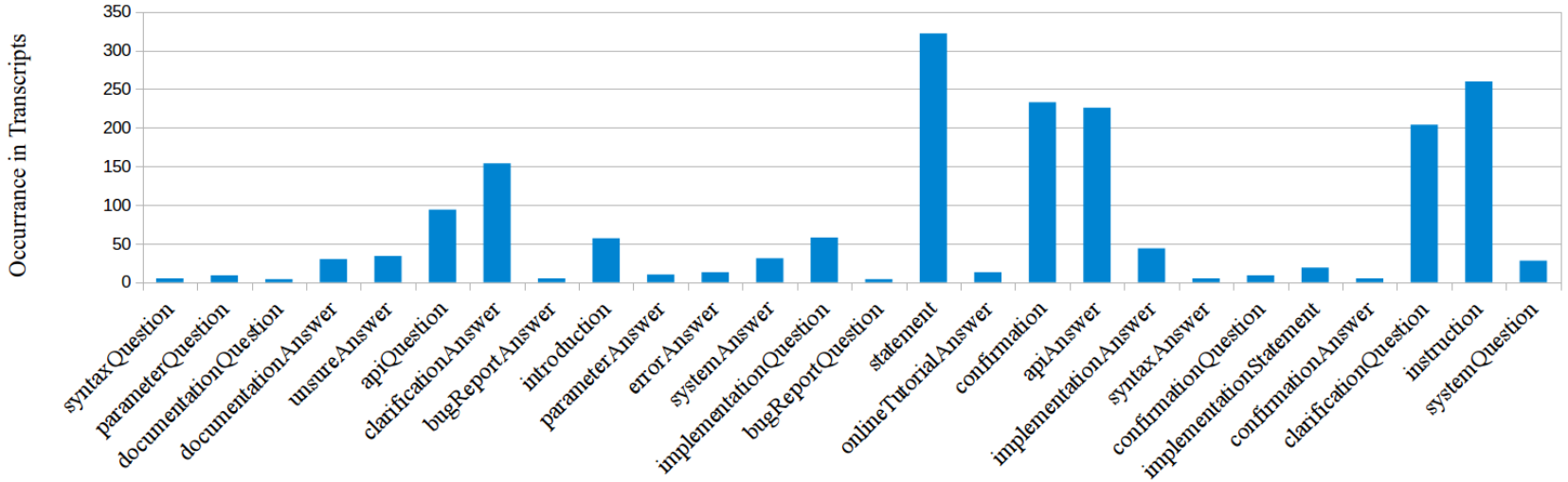

Figure A.1: The annotation lables of all 30 transcripts and occurances for each label. Note that each turn can have multiple labels (dialogue act type).

BIBLIOGRAPHY

1. H. Ai, J. R. Tetreault, and D. J. Litman. Comparing user simulation models for dialog strategy learning. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 1–4. Association for Computational Linguistics, 2007.

2. E. M. Altmann. Near-term memory in programming: a simulation-based analysis. *International Journal of Human-Computer Studies*, 54(2):189 – 210, 2001. ISSN 1071-5819. doi: http://dx.doi.org/10.1006/ijhc.2000.0407. URL `http://www.sciencedirect.com/science/article/pii/S1071581900904075`.

3. T. Andernach. A machine learning approach to the classification of dialogue utterances. In *Proceedings of the Second International Conference on New Methods in Language Processing*, pages 98–109. ACM, 1996.

4. J. Ang, Y. Liu, and E. Shriberg. Automatic dialog act segmentation and classification in multiparty meetings. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, volume 1, pages I–1061. IEEE, 2005.

5. A. Armaly, J. Klaczynski, and C. McMillan. A case study of automated feature location techniques for industrial cost estimation. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 553–562, Oct 2016. doi: 10.1109/ICSME.2016.76.

6. V. Arnaoudova, S. Haiduc, A. Marcus, and G. Antoniol. The use of text retrieval and natural language processing in software engineering. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 949–950, May 2015. doi: 10.1109/ICSE.2015.301.

7. K. Bach and R. Harnish. Linguistic communication and speech acts. 1979.

8. M. Baert. Simplescreenrecorder. `http://www.maartenbaert.be/simplescreenrecorder/`, 2018. Accessed: 2018-03-02.

9. S. Bangalore, G. Di Fabbrizio, and A. Stent. Learning the structure of task-driven human–human dialogs. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(7):1249–1259, 2008.

10. G. E. Batista, R. C. Prati, and M. C. Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM Sigkdd Explorations Newsletter*, 6(1):20–29, 2004.

11. A. Begel and B. Simon. Struggles of new college graduates in their first software development job. In *ACM SIGCSE Bulletin*, volume 40, pages 226–230. ACM, 2008.

12. B. L. Berg. *Methods for the social sciences*. Pearson Education Inc, United States of America, 2004.

13. D. Binkley, D. Heinz, D. Lawrie, and J. Overfelt. Understanding lda in source code analysis. In *Proceedings of the 22Nd International Conference on Program Comprehension*, ICPC 2014, pages 26–36, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2879-1. doi: 10.1145/2597008.2597150. URL `http://doi.acm.org/10.1145/2597008.2597150`.

14. P. Blunsom, N. Kalchbrenner, and N. Kalchbrenner. Recurrent convolutional neural networks for discourse compositionality. In *Proceedings of the 2013 Workshop on Continuous Vector Space Models and their Compositionality*. Proceedings of the 2013 Workshop on Continuous Vector Space Models and their Compositionality, 2013.

15. B. Boehm. A view of 20th and 21st century software engineering. In *Proceedings of the 28th international conference on Software engineering*, pages 12–29. ACM, 2006.

16. K. E. Boyer, E. Y. Ha, R. Phillips, M. D. Wallis, M. A. Vouk, and J. C. Lester. Dialogue act modeling in a complex task-oriented domain. In *Proceedings of the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 297–305. Association for Computational Linguistics, 2010.

17. N. Bradley, T. Fritz, and R. Holmes. Context-aware conversational developer assistants. In *International Conference on Software Engineering*, page 12. ACM, 2018.

18. S. Burger, K. Weilhammer, F. Schiel, and H. G. Tillmann. Verbmobil data collection and annotation. In *Verbmobil: Foundations of speech-to-speech translation*, pages 537–549. Springer, 2000.

19. J. Carletta. Assessing agreement on classification tasks: The kappa statistic. *Comput. Linguist.*, 22(2):249–254, June 1996. ISSN 0891-2017. URL `http://dl.acm.org/citation.cfm?id=230386.230390`.

20. R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 161–168, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: 10.1145/1143844.1143865. URL `http://doi.acm.org/10.1145/1143844.1143865`.

21. N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16: 321–357, 2002.

22. H. Chen, X. Liu, D. Yin, and J. Tang. A survey on dialogue systems: Recent advances and new frontiers. *ACM SIGKDD Explorations Newsletter*, 19(2):25–35, 2017.

23. Z. Chen, R. Yang, Z. Zhao, D. Cai, and X. He. Dialogue act recognition via crf-attentive structured network. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 225–234. ACM, 2018.

24. R. Craggs and M. M. Wood. Evaluating discourse and dialogue coding schemes. *Comput. Linguist.*, 31(3):289–296, Sept. 2005. ISSN 0891-2017. doi: 10.1162/089120105774321109. URL http://dx.doi.org/10.1162/089120105774321109.

25. B. Cruz, B. Jayaraman, A. Dwarakanath, and C. McMillan. Detecting vague words & phrases in requirements documents in a multilingual environment. In *Requirements Engineering Conference (RE), 2017 25th IEEE International*, September 2017.

26. L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286. ACM, 2012.

27. N. Dahlbäck, A. Jönsson, and L. Ahrenberg. Wizard of oz studies—why and how. *Knowledge-based systems*, 6(4):258–266, 1993.

28. N. Dahlbäck, A. Jönsson, and L. Ahrenberg. Wizard of oz studies — why and how. *Knowledge-Based Systems*, 6(4):258 – 266, 1993. ISSN 0950-7051. doi: http://dx.doi.org/10.1016/0950-7051(93)90017-N. URL http://www.sciencedirect.com/science/article/pii/095070519390017N. Special Issue: Intelligent User Interfaces.

29. S. D'mello and A. Graesser. Autotutor and affective autotutor: Learning by talking with cognitively and emotionally intelligent computers that talk back. *ACM Trans. Interact. Intell. Syst.*, 2(4):23:1–23:39, Jan. 2013. ISSN 2160-6455. doi: 10.1145/2395123.2395128. URL http://doi.acm.org/10.1145/2395123.2395128.

30. J. Escobar-Avila, E. Parra, and S. Haiduc. Text retrieval-based tagging of software engineering video tutorials. In *Proceedings of the 39th International Conference on Software Engineering Companion*, ICSE-C '17, pages 341–343, Piscataway, NJ, USA, 2017. IEEE Press. ISBN 978-1-5386-1589-8. doi: 10.1109/ICSE-C.2017.121. URL https://doi.org/10.1109/ICSE-C.2017.121.

31. K. Forbes-Riley, M. Rotaru, and D. J. Litman. The relative impact of student affect on performance models in a spoken dialogue tutoring system. *User Modeling and User-Adapted Interaction*, 18(1-2):11–43, Feb. 2008. ISSN 0924-1868. doi: 10.1007/s11257-007-9038-5. URL http://dx.doi.org/10.1007/s11257-007-9038-5.

32. C. E. Ford, B. A. Fox, and S. A. Thompson. *The language of turn and sequence*. Oxford University Press on Demand, 2002.

33. A. Forward and T. C. Lethbridge. The relevance of software documentation, tools and technologies: a survey. In *Proceedings of the 2002 ACM symposium on Document engineering*, DocEng '02, pages 26–33, New York, NY, USA, 2002. ACM. ISBN 1-58113-594-7. doi: 10.1145/585058.585065. URL http://doi.acm.org/10.1145/585058.585065.

34. A. Foundation. Apache commons io. https://commons.apache.org/proper/commons-io/, 2018. Accessed: 2018-03-02.

35. E. Foundation. Eclipse. https://eclipse.org/ide/, 2018. Accessed: 2018-03-02.

36. R. Gangadharaiah, B. Narayanaswamy, and C. Elkan. What we need to learn if we want to do and not just talk. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, volume 3, pages 25–32, 2018.

37. J. Geertzen, V. Petukhova, and H. Bunt. A multidimensional approach to utterance segmentation and dialogue act classification. In *Proceedings of the 8th SIGdial Workshop on Discourse and Dialogue, Antwerp*, pages 140–149, 2007.

38. M. Gethers, B. Dit, H. Kagdi, and D. Poshyvanyk. Integrated impact analysis for managing software changes. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 430–440. IEEE, 2012.

39. M. Goodrum, J. Cleland-Huang, R. Lutz, J. Cheng, and R. Metoyer. What requirements knowledge do developers need to manage change in safety-critical systems? In *Requirements Engineering Conference (RE), 2017 IEEE 25th International*, pages 90–99. IEEE, 2017.

40. S. Grau, E. Sanchis, M. J. Castro, and D. Vilar. Dialogue act classification using a bayesian approach. In *9th Conference Speech and Computer*, 2004.

41. S. Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42 (1-3):335–346, 1990.

42. L. Hirschman. Evaluating spoken language interaction: Experiences from the darpa spoken language program 1988–1995. *To appear. See http://www. research. att. com/~ walker/eval/hirschman-survey. ps*, 1998.

43. R. Hoda, J. Noble, and S. Marshall. Using grounded theory to study the human aspects of software engineering. In *Human Aspects of Software Engineering*, page 5. ACM, 2010.

44. R. Holmes and R. J. Walker. Systematizing pragmatic software reuse. *ACM Trans. Softw. Eng. Methodol.*, 21(4):20:1–20:44, Feb. 2013. ISSN 1049-331X. doi: 10.1145/2377656.2377657. URL http://doi.acm.org/10.1145/2377656.2377657.

45. D. W. Hosmer Jr and S. Lemeshow. *Applied logistic regression*. John Wiley & Sons, 2004.

46. I. Hutchby and R. Wooffitt. *Conversation analysis*. Polity, 2008.

47. Y. Ji, G. Haffari, and J. Eisenstein. A latent variable recurrent neural network for discourse relation language models. *NAACL-HLT*, 2016.

48. S. Jiang, C. McMillan, and R. Santelices. Do programmers do change impact analysis in debugging? *Empirical Software Engineering*, 22(2):631–669, Apr 2017. ISSN 1573-7616. doi: 10.1007/s10664-016-9441-9. URL https://doi.org/10.1007/s10664-016-9441-9.

49. Y. Kang, Y. Zhang, J. K. Kummerfeld, L. Tang, and J. Mars. Data collection for dialogue system: A startup perspective. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, volume 3, pages 33–40, 2018.

50. H. Khanpour, N. Guntakandla, and R. Nielsen. Dialogue act classification in domain-independent conversations using a deep recurrent neural network. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2012–2021, 2016.

51. A. J. Ko and B. A. Myers. Extracting and answering why and why not questions about java program output. *ACM Trans. Softw. Eng. Methodol.*, 20(2):4:1–4:36, Sept. 2010. ISSN 1049-331X. doi: 10.1145/1824760.1824761. URL http://doi.acm.org/10.1145/1824760.1824761.

52. A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 344–353. IEEE, 2007.

53. J.-P. Krämer, J. Kurz, T. Karrer, and J. Borchers. Blaze. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 1457–1458, Piscataway, NJ, USA, 2012. IEEE Press. ISBN 978-1-4673-1067-3. URL http://dl.acm.org/citation.cfm?id=2337223.2337451.

54. H. Kumar, A. Agarwal, R. Dasgupta, S. Joshi, and A. Kumar. Dialogue act sequence labeling using hierarchical encoder with crf. *AAAI*, 2018.

55. T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 492–501, New York, NY, USA, 2006. ACM. ISBN 1-59593-375-1. doi: 10.1145/1134285.1134355. URL http://doi.acm.org/10.1145/1134285.1134355.

56. J. Y. Lee and F. Dernoncourt. Sequential short-text classification with recurrent and convolutional neural networks. *NAACL-HLT*, 2016.

57. G. Lemaître, F. Nogueira, and C. K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017. URL http://jmlr.org/papers/v18/16-365.

58. O. Lemon. Learning what to say and how to say it: Joint optimisation of spoken dialogue management and natural language generation. *Computer Speech & Language*, 25(2):210–221, 2011.

59. T. Lethbridge, J. Singer, and A. Forward. How software engineers use documentation: the state of the practice. *Software, IEEE*, 20(6):35–39, 2003. ISSN 0740-7459. doi: 10.1109/MS.2003.1241364.

60. Y. Liu, K. Han, Z. Tan, and Y. Lei. Using context information for dialog act classification in dnn framework. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2170–2178, 2017.

61. I. McCowan, J. Carletta, W. Kraaij, S. Ashby, S. Bourban, M. Flynn, M. Guillemot, T. Hain, J. Kadlec, V. Karaiskos, et al. The ami meeting corpus. In *Proceedings of the 5th International Conference on Methods and Techniques in Behavioral Research*, volume 88, 2005.

62. D. Milajevs and M. Purver. Investigating the contribution of distributional semantic information for dialogue act classification. In *Proceedings of the 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)*, pages 40–47, 2014.

63. S. Mirghasemi, J. J. Barton, and C. Petitpierre. Querypoint: moving backwards on wrong values in the buggy execution. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ESEC/FSE '11, pages 436–439, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0443-6. doi: 10.1145/2025113.2025184. URL http://doi.acm.org/10.1145/2025113.2025184.

64. V. O. Mittal and J. D. Moore. Dynamic generation of follow up question menus: Facilitating interactive natural language dialogues. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 90–97, New

York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-84705-1. doi: 10.1145/223904.223916. URL http://dx.doi.org/10.1145/223904.223916.

65. J. D. Moore. *Participating in Explanatory Dialogues: Interpreting and Responding to Questions in Context*. MIT Press, Cambridge, MA, USA, 1994. ISBN 0-262-13301-6.

66. G. Murray and G. Carenini. Summarizing spoken and written conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 773–782. Association for Computational Linguistics, 2008.

67. T. Nishida. *Conversational Informatics: An Engineering Approach*. Wiley, 2007.

68. OpenCSV. Opencsv. http://opencsv.sourceforge.net/, 2017. Accessed: 2017-08-20.

69. R. J. Passonneau and D. J. Litman. Intention-based segmentation: Human reliability and correlation with linguistic cues. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 148–155. Association for Computational Linguistics, 1993.

70. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

71. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

72. P. Pruski, S. Lohar, W. Goss, A. Rasin, and J. Cleland-Huang. Tiqi: answering unstructured natural language trace queries. *Requirements Engineering*, 20(3):215–232, Sep 2015. ISSN 1432-010X. doi: 10.1007/s00766-015-0224-4. URL http://dx.doi.org/10.1007/s00766-015-0224-4.

73. S. Rastkar, G. C. Murphy, and G. Murray. Summarizing software artifacts: a case study of bug reports. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 505–514. ACM, 2010.

74. S. Rastkar, G. C. Murphy, and G. Murray. Automatic summarization of bug reports. *IEEE Transactions on Software Engineering*, 40(4):366–380, 2014.

75. J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333–359, 2011.

76. N. Reithinger and M. Klesen. Dialogue act classification using language models. In *Fifth European Conference on Speech Communication and Technology*, 1997.

77. N. Reithinger and E. Maier. Utilizing statistical dialogue act processing in verbmobil. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*, ACL '95, pages 116–121, Stroudsburg, PA, USA, 1995. Association for Computational Linguistics. doi: 10.3115/981658.981674. URL http://dx.doi.org/10.3115/981658.981674.

78. L. D. Riek. Wizard of oz studies in hri: a systematic review and new reporting guidelines. *Journal of Human-Robot Interaction*, 1(1), 2012.

79. V. Rieser and O. Lemon. *Reinforcement learning for adaptive dialogue systems: a data-driven methodology for dialogue management and natural language generation*. Springer Science & Business Media, 2011.

80. C. J. V. Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979. ISBN 0408709294.

81. M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann. *Recommendation systems in software engineering*. Springer, 2014.

82. M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, M. Linares-Vásquez, et al. On-demand developer documentation. In *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*. IEEE, 2017.

83. P. Rodeghero, C. McMillan, P. W. McBurney, N. Bosch, and S. D'Mello. Improving automated source code summarization via an eye-tracking study of programmers. In *Proceedings of the 36th international conference on Software engineering*, ICSE '14, 2014. To appear.

84. P. Rodeghero, S. Jiang, A. Armaly, and C. McMillan. Detecting user story information in developer-client conversations to generate extractive summaries. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, pages 49–59, Piscataway, NJ, USA, 2017. IEEE Press. ISBN 978-1-5386-3868-2. doi: 10.1109/ICSE.2017.13. URL https://doi.org/10.1109/ICSE.2017.13.

85. P. Rodeghero, S. Jiang, A. Armaly, and C. McMillan. Detecting user story information in developer-client conversations to generate extractive summaries. In *Proceedings of the 39th International Conference on Software Engineering*, pages 49–59. IEEE Press, 2017.

86. T. Roehm, R. Tiarks, R. Koschke, and W. Maalej. How do professional developers comprehend software? In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 255–265, Piscataway, NJ, USA, 2012. IEEE Press. ISBN 978-1-4673-1067-3. URL http://dl.acm.org/citation.cfm?id=2337223.2337254.

87. T. Roehm, R. Tiarks, R. Koschke, and W. Maalej. How do professional developers comprehend software? In *Proceedings of the 34th International Conference on Software Engineering*, pages 255–265. IEEE Press, 2012.

88. R. Sarikaya, P. A. Crook, A. Marin, M. Jeong, J.-P. Robichaud, A. Celikyilmaz, Y.-B. Kim, A. Rochette, O. Z. Khan, X. Liu, et al. An overview of end-to-end language understanding and dialog management for personal digital assistants. In *Spoken Language Technology Workshop (SLT), 2016 IEEE*, pages 391–397. IEEE, 2016.

89. J. Schatzmann, B. Thomson, K. Weilhammer, H. Ye, and S. Young. Agenda-based user simulation for bootstrapping a pomdp dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 149–152. Association for Computational Linguistics, 2007.

90. E. A. Schegloff. *Sequence Organization in Interaction: A Primer in Conversation Analysis I*. Cambridge University Press, 2007.

91. J. Searle. *What is a speech act?* na, 1965.

92. R. Serafin, B. Di Eugenio, and M. Glass. Latent semantic analysis for dialogue act classification. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: companion volume of the Proceedings of HLT-NAACL 2003–short papers-Volume 2*, pages 94–96. Association for Computational Linguistics, 2003.

93. I. V. Serban, R. Lowe, P. Henderson, L. Charlin, and J. Pineau. A survey of available corpora for building data-driven dialogue systems. *arXiv preprint arXiv:1512.05742*, 2015.

94. I. V. Serban, R. Lowe, P. Henderson, L. Charlin, and J. Pineau. A survey of available corpora for building data-driven dialogue systems: The journal version. *Dialogue & Discourse*, 9(1):1–49, 2018.

95. B. Sharif, M. Falcone, and J. I. Maletic. An eye-tracking study on the role of scan time in finding source code defects. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12, pages 381–384, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1221-9. doi: 10.1145/2168556.2168642. URL `http://doi.acm.org/10.1145/2168556.2168642`.

96. J. Sillito, G. C. Murphy, and K. De Volder. Asking and answering questions during a programming change task. *IEEE Trans. Softw. Eng.*, 34(4):434–451, July 2008. ISSN 0098-5589. doi: 10.1109/TSE.2008.26. URL `http://dx.doi.org/10.1109/TSE.2008.26`.

97. S. E. Sim, C. L. A. Clarke, and R. C. Holt. Archetypal source code searches: A survey of software developers and maintainers. In *Proceedings of the 6th International Workshop on Program Comprehension*, IWPC '98, pages 180–, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8560-3. URL `http://dl.acm.org/citation.cfm?id=580914.858229`.

98. J. Starke, C. Luce, and J. Sillito. Searching and skimming: An exploratory study. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 157–166, 2009. doi: 10.1109/ICSM.2009.5306335.

99. A. Stolcke, K. Ries, N. Coccaro, E. Shriberg, R. Bates, D. Jurafsky, P. Taylor, R. Martin, C. Van Ess-Dykema, and M. Meteer. Dialogue act modeling for automatic tagging and recognition of conversational speech. *Computational linguistics*, 26(3): 339–373, 2000.

100. D. Surendran and G.-A. Levow. Dialog act tagging with support vector machines and hidden markov models. In *Ninth International Conference on Spoken Language Processing*, 2006.

101. M. Tavafi, Y. Mehdad, S. Joty, G. Carenini, and R. Ng. Dialogue act recognition in synchronous and asynchronous conversations. In *Proceedings of the SIGDIAL 2013 Conference*, pages 117–121, 2013.

102. P. Ten Have. *Doing conversation analysis*. Sage, 2007.

103. E. Tyugu. *Knowledge-Based Programming (Turing Institute Press Knowledge Engineering Tutorial Series)*. Addison-Wesley Longman Publishing Co., Inc., 1988.

104. UpWork. Upwork. `https://www.upwork.com/`, 2018. Accessed: 2018-03-02.

105. M. A. Walker, R. Passonneau, and J. E. Boland. Quantitative and qualitative evaluation of darpa communicator spoken dialogue systems. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, ACL '01, pages 515–522, Stroudsburg, PA, USA, 2001. Association for Computational Linguistics. doi: 10.3115/1073012.1073078. URL `http://dx.doi.org/10.3115/1073012.1073078`.

106. N. Webb, M. Hepple, and Y. Wilks. Dialogue act classification based on intra-utterance features. In *Proceedings of the AAAI Workshop on Spoken Language Understanding*, volume 4, page 5. Citeseer, 2005.

107. K. F. White and W. G. Lutters. Behind the curtain: Lessons learned from a wizard of oz field experiment. *SIGGROUP Bull.*, 24(3):129–135, Dec. 2003. ISSN 2372-7403. doi: 10.1145/1052829.1052854. URL `http://doi.acm.org/10.1145/1052829.1052854`.

108. A. Wood, P. Rodeghero, A. Armaly, and C. McMillan. Detecting speech act types in developer question/answer conversations during bug repair. *Foundations of Software Engineering (FSE)*, 2018.

109. M. Zimmermann. Joint segmentation and classification of dialog acts using conditional random fields. In *Tenth Annual Conference of the International Speech Communication Association*, 2009.

This document was prepared & typeset with pdfLaTeX*, and formatted with* NDdiss2$_\varepsilon$ *classfile (v3.2017.2[2017/05/09])*