



MONASH University

Intelligent Question and Answering in Software Q&A Community

Zhipeng Gao

Doctor of Philosophy

A thesis submitted for the degree of *Doctor of Philosophy* at

Monash University in 2021

Faculty of Information Technology

Copyright notice

© Zhipeng Gao (2021) Except as provided in the Copyright Act 1968, this thesis may not be reproduced in any form without the written permission of the author.

I certify that I have made all reasonable efforts to secure copyright permissions for third-party content included in this thesis and have not knowingly added copyright content to my work without the owner's permission.

Abstract

Software question and answering (SQA) platforms (e.g., Stack Overflow, Ask Ubuntu and Super User, etc) have been heavily used by developers to seek help to solve their daily problems. Typically, when a developer encounters a technical problem, he or she formulates the problem as a query and use the search engine to find possible solutions in SQA sites such as Stack Overflow. Interacting with SQA, i.e., asking questions and searching answers, have been an essential part of developers' daily work for problem solving.

Despite the great success of SQA and active user participation, the “*answer hungry*” problem is still one major existing problem with these SQA platforms, i.e., a large number of questions remain unanswered and/or unresolved. The “*answer hungry*” problem is probably caused by the following several reasons: (1) the questions posted by developers are of poor quality (e.g., ambiguous, inaccurate or uninformative), which may discourage the potential experts. (2) the SQA search engine are hardly recommend the appropriate answers to the unanswered/unresolved questions. (3) even though the appropriate answer is returned by the search engine, developers may fail to filter out irrelevant results and tend to get lost in the massive amount of information. Three key research questions are raised regarding the above three reasons, i.e., “*how to improve the quality of questions in SQA community?*”, “*how to recommend appropriate answers to technical questions?*”, and “*how to search the best code solutions in SQA for a programming task?*”.

This Ph.D. work focuses on this open issue, which aims to alleviate the “*answer hungry*” problem in SQA sites. To be more specific, we present three different approaches to solve the aforementioned three research questions respectively, we present this dissertation to explore question and answering in software Q&A community by contributing to the following three building blocks:

- **Code2Que.** A significant number of questions in SQA sites are of low-quality and not attractive to other potential experts. Sometimes it is hard for developers, especially the novice users, to describe the problem they meet due to their lack of knowledge or terminology behind the problems. We propose a sequence-to-sequence learning approach, **Code2Que**, which can help developers in writing higher quality questions for a given code snippet. We evaluated our approach on Stack Overflow datasets over a variety of programming languages, and the experimental results show that our approach significantly outperforms existing baselines and can improve the question titles in terms of *Cleanness*, *Fitness* and *Willingness* measures.

- **DeepAns**. One of the major reasons for the “*answer hungry*” problem in SQA community is the search engines are hardly to recommend relevant answers for the unanswered and/or unresolved questions. However, with the rapid growth of the SQA community, a tremendous number of historical question answer (QA) pairs, as time goes on, have been archived in the SQA databases. Developers therefore have large chances to directly get the answers by searching from the repositories, rather than the time-consuming waiting. To solve this we propose a novel neural network based approach, named **DeepAns**, to identify the most relevant answer among a set of answer candidates. Our experimental results show that our approach significantly outperforms several state-of-the-art baselines in both automatic evaluation and human evaluation.
- **Que2Code**. Following our previous work, sometimes even if the search engine successfully returns a target post for a given user question, there is often too much noisy and redundant information associated with the returned posts, and the code solution can be easily buried in all this overwhelming amount of information and developers may get lost with the irrelevant information. There is a need by developers to only receive the most suitable and useful code solutions to their current programming tasks. To address this, we present a query-driven code search tool, named **Que2Code**, that identifies the best code solutions for a user query from Stack Overflow posts. Both the automatic and human evaluation results demonstrate the promising performance of our approach over a set of state-of-the-art baselines.

Publications during enrolment

Publications included in this thesis

- 1) **Zhipeng Gao**, Xin Xia, John Grundy, David Lo, and Yuan-Fang Li. 2020. Generating Question Titles for Stack Overflow from Mined Code Snippets. *ACM Transactions on Software Engineering and Methodology*, 29, 4, Article 26 (October 2020), 37 pages. (Chapter 3)
- 2) **Zhipeng Gao**, Xin Xia, John Grundy, David Lo, and Yuan-Fang Li, 2021, Code2Que: A Tool for Improving Question Titles from Mined Code Snippets in Stack Overflow. *The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2021*, Demo Track. (Chapter 3)
- 3) **Zhipeng Gao**, Xin Xia, David Lo, and John Grundy. 2021. Technical Q&A Site Answer Recommendation via Question Boosting. *ACM Transactions on Software Engineering and Methodology*, 30, 1, Article 11 (January 2021), 34 pages. (Chapter 4)

Submitted manuscripts included in this thesis

- 4) **Zhipeng Gao**, Xin Xia, David Lo, John Grundy, Xindong Zhang and Zhenchang Xing. 2020. I Know What You Are Searching For: Code Snippet Recommendation from Stack Overflow Posts. *ACM Transactions on Software Engineering and Methodology* (Under Major Revision). (Chapter 5)

Other publications during candidature

- 5) **Zhipeng Gao**, Lingxiao Jiang, Xin Xia, David Lo and John Grundy, Checking Smart Contracts with Structural Code Embedding. 2020, *IEEE Transactions on Software Engineering*, doi: 10.1109/TSE.2020.2971482.
- 6) **Zhipeng Gao**, Xin Xia, Vinoj Jayasundara, David Lo, Lingxiao Jiang and John Grundy. SMARTEMBED: A Tool for Clone and Bug Detection in Smart Contracts through Structural Code Embedding. *Proceedings - 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019*.
- 7) **Zhipeng Gao**. When Deep Learning Meets Smart Contracts. *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020*.
- 8) **Zhipeng Gao**, Xin Xia, David Lo, John Grundy and Thomas Zimmermann. 2021, *Automating the Removal of Obsolete TODO Comments*. The ACM Joint European

Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2021.

- 9) Xing Hu, **Zhipeng Gao**, Xin Xia, David Lo and Thomas Zimmermann. 2021, Automating User Notice Generation for Smart Contract Functions, *The IEEE/ACM International Conference on Automated Software Engineering*. ASE 2021.

Thesis including published works declaration

In accordance with Monash University Doctorate Regulation 17.2 Doctor of Philosophy and Research Master's regulations the following declarations are made:

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

This thesis includes 2 original papers published in top software engineering journals and 1 submitted publications. The core theme of the thesis is intelligent question and answering in software Q&A community. The ideas, development and writing up of all the papers in the thesis were the principal responsibility of myself, the student, working within the Faculty of Information Technology under the supervision of Professor John Grundy and Dr. Xin Xia.

The inclusion of co-authors reflects the fact that the work came from active collaboration between researchers and acknowledges input into team-based research.

In the case of Chapter 3, 4, and 5 my contribution to the work involved the following:

Thesis Chapter	Publication Title	Status (published, in press, accepted or returned for revision, submitted)	Nature and % of student contribution	Co-author name(s) Nature and % of Co-author's contribution*	Co-author(s), Monash student Y/N*
3	Generating Question Titles for Stack Overflow from Mined Code Snippets	<i>Published</i>	70%. Concept, conducting the experiments and writing the manuscript	<ol style="list-style-type: none"> Xin Xia, Concept and input into manuscript, 10%. John Grundy, Supervised study and input into manuscript. 10%. David Lo, Concept and input into manuscript, 5% Yuan-Fang Li, Concept and input into manuscript, 5%. 	<ol style="list-style-type: none"> No No No No

4	Technical Q&A Site Answer Recommendation via Question Boosting	<i>Published</i>	70%. Concept, conducting the experiments and writing the manuscript	<ol style="list-style-type: none"> 1. Xin Xia, Concept and input into manuscript, 10%. 2. David Lo, Concept and input into manuscript, 10%. 3. John Grundy, Supervised study and input into manuscript. 10%. 	<ol style="list-style-type: none"> 1. No 2. No 3. No
5	I Know What You Are Searching For: Code Snippet Recommendation from Stack Overflow Posts	<i>Submitted</i>	65%. Concept, conducting the experiments and writing the manuscript	<ol style="list-style-type: none"> 1. Xin Xia, Concept and input into manuscript, 10%. 2. David Lo, Concept and input into manuscript, 10%. 3. John Grundy, Supervised study and input into manuscript, 10%. 4. Xindong Zhang, Concept and input manuscript, 3%. 5. Zhenchang Xing, Concept and input into manuscript, 2%. 	<ol style="list-style-type: none"> 1. No 2. No 3. No 4. No 5. No

I have renumbered sections of submitted or published papers in order to generate a consistent presentation within the thesis.

Student name: Zhipeng Gao

Student signature: 

Date: 11/19/2021

I hereby certify that the above declaration correctly reflects the nature and extent of the student's and co-authors' contributions to this work. In instances where I am not the responsible author I have consulted with the responsible author to agree on the respective contributions of the authors.

Main Supervisor name: John Grundy

Main Supervisor signature: 

Date: 11/19/2021

Acknowledgements

First, I would like to express my sincere gratitude and appreciation to my supervisors: Prof. John Grundy, A/Prof. Aldeida Aleti and Dr. Xin Xia for their patient and thoughtful guidance during my Ph.D. study. Dr. Xin Xia and Prof. John Grundy gave me this great opportunity and encouraged me to come to Monash to pursue my doctoral degree. My supervisors supported me with great kindness throughout my Ph.D. journey. Without their endless help, I would never know how to solve real problem in software engineering, Prof. John Grundy and A/Prof. Aldeida have taught me how to perform research, write technical report and conduct attractive presentations, Dr. Xin Xia gave me a lot of guidance on idea formulation and implementation. The weekly meetings with them inspired me a lot for my Ph.D. study. Their dedicated guidance has made my Ph.D. journey a fruitful and fascinating experience.

Secondly, I am equally grateful to all my co-authors, including Prof. David Lo, A/Prof. Lingxiao Jiang, A/Prof. Zhenchang Xing, Prof. Thomas Zimmermann. They have provided their valuable discussions and suggestions for our collaborated works. Their detailed comments have made our work more solid and significantly improved the quality of our work.

Furthermore, I would extend my thanks to all my milestone panel members, including Dr. Li Li, Dr. Chunyang Chen and A/Prof. Ron Steinfeld, they gave me valuable feedback for my three milestones during my Ph.D. study. It is my great honor to have them as my panel members, and I appreciate very much for their efforts to examine my milestones and evaluate my Ph.D. work. I would also like to thank all the academic staff and administrative staff of the Faculty of Information Technology. They helped me a lot especially when I perform research overseas.

Finally, I would like to express my gratitude to my parents and grandparents, who gave me unconditional support and encouragement, they let me know whenever I encounter any troubles, they will always stay by my side. I would like to thank my girlfriend Miaomiao, who brings the most considerable insights and happiness to my life, her endless support makes me more than I can be.

Contents

1	Introduction	1
1.1	Background	1
1.2	Motivations and Objectives	3
1.3	Thesis Scope	6
1.4	Thesis contribution	8
1.5	Structure of the thesis	12
2	Literature Review	13
2.1	Software Question Answering Community	13
2.2	Mining SQA Community	16
2.3	Deep Learning for Software Engineering	26
3	Code2Que: Improving Question Titles from Mined Code Snippets	31
3.1	Introduction	31
3.2	Related Work	36
3.2.1	Deep Source Code Summarization	36
3.2.2	Question Quality Study on Stack Overflow	37
3.2.3	Machine/Deep Learning on Software Engineering	38
3.3	Motivation	39
3.3.1	The Problem and Our Solution	40
3.3.2	User Scenarios	41

3.3.3	Motivating Examples	42
3.4	Approach	44
3.4.1	Task Definition	45
3.4.2	Source-code Encoder	46
3.4.3	Question Decoder	46
3.4.4	Incorporating Attention Mechanism	47
3.4.5	Incorporating Copy Mechanism	48
3.4.6	Incorporating a Coverage Mechanism	50
3.5	Experimental Setup	51
3.5.1	Pre-processing	51
3.5.2	Implementation Details	55
3.5.3	Baselines	56
3.5.4	Evaluation Metrics	57
3.6	Results and Analysis	60
3.6.1	Automatic Evaluation	60
3.6.2	Compared with CODE-NN	66
3.6.3	Human Evaluation	68
3.6.4	Practical Manual Evaluation	70
3.6.5	Ablation Analysis	73
3.6.6	Parameters Settings	76
3.6.7	Efficient Analysis	79
3.7	Discussion	81
3.7.1	Question Quality Improvement	82
3.7.2	Deep Sequence to Sequence Approach	83
3.7.3	Question Generation Task	84
3.8	Threats to Validity	85

3.9	Summary	87
4	DeepAns: Best Answer Recommendation via Question Boosting	88
4.1	Introduction	88
4.2	Approach	92
4.2.1	Question Boosting	93
4.2.2	Label Establishment	96
4.2.3	Answer Recommendation	97
4.2.4	DeepAns Algorithm	101
4.3	Experiment Setup	102
4.3.1	Data Preparation	102
4.3.2	Implementation Details	104
4.3.3	Baselines	106
4.3.4	Evaluation Methods	107
4.4	Automatic Evaluation Results	109
4.4.1	Effectiveness Analysis	109
4.4.2	Ablation Analysis	112
4.4.3	Parameters Tuning Analysis	114
4.5	User Study Setup and Results	117
4.6	Discussion	127
4.6.1	Strength of Our Approach	127
4.6.2	Threats to Validity	128
4.6.3	Outlier Cases Study	130
4.7	Related Work	131
4.7.1	Best Answer Retrieval	131
4.7.2	Query Expansion in SE	132
4.7.3	Deep Learning in SE	133

4.8	Summary	133
5	Que2Code: Searching Code Solutions from Stack Overflow Posts	135
5.1	Introduction	135
5.2	Motivation	139
5.3	Approach	142
5.3.1	Overview of Approach	142
5.3.2	Semantically-Equivalent Question Retrieval	144
5.3.3	Best Code Snippet Recommendation	147
5.3.4	Experimental Settings	152
5.4	Automatic Evaluation	153
5.4.1	Effectiveness of QueryRewriter	154
5.4.2	Context Analysis	162
5.4.3	Ablation Analysis	164
5.4.4	Parameter Tuning Analysis	165
5.4.5	Effectiveness of CodeSelector	167
5.4.6	Component-Wise Evaluation	174
5.4.7	Robustness Analysis	175
5.5	Human Evaluation	177
5.5.1	Human Evaluation Preliminary	177
5.5.2	User Study on Question Relevance	179
5.5.3	User Study on Code Usefulness	182
5.5.4	Qualitative Analysis	184
5.6	Practical Usage	188
5.7	Discussion	190
5.7.1	Strengths of Our Approach	190
5.7.2	Threats to Validity	191

5.8	Related Work	193
5.8.1	Code Search in Software Engineering	193
5.8.2	Duplicated Questions in Stack Overflow	194
5.8.3	Query Reformulation in Software Engineering	195
5.8.4	Question Answering in CQA Sites	196
5.9	Summary	196
6	Conclusion and Future Work	198
6.1	Key Contributions	198
6.2	Future Work	201
6.3	Summary	202
	Bibliography	203

List of Figures

2.1	Example of questions and answers in SQA community	15
3.1	Example Code Snippet & Question Pairs	34
3.2	Example of Problem Questions Title (for Python)	43
3.3	Example of Problem Questions Title (for Java)	44
3.4	Workflow of Code2Que Model	46
3.5	Attention & Copy & Coverage Mechanism	49
3.6	Volinplots of Code Distribution	53
3.7	Volinplots of Question Distribution	54
3.8	Examples of output generated by each model	64
3.9	User Study Case (Human Evaluation)	70
3.10	User Study Case (Practical Manual Evaluation)	71
3.11	Practical Manual Evaluation Example	73
3.12	Ablation Analysis Example	75
3.13	BLEU4 Score under different vocab threshold	78
3.14	ROUGE-L Score under different vocab threshold	79
3.15	BLEU4 Score under different sizes of word embeddings	80
3.16	ROUGE-L Score under different sizes of word embeddings	80
3.17	CODE2QUE Web Service Tool	82
4.1	Example Post on Ask Ubuntu	90

4.2	Question boosting process	94
4.3	Label Establishing Process	96
4.4	Overall architecture of the answer recommendation model.	98
4.5	Sensitivity Analysis on Ask-ubuntu (left) and Super-user (right)	114
4.6	Robustness Analysis on Ask-ubuntu (left) and Super-user (right)	116
4.7	Evaluation Examples of Question Boosting.	119
4.8	Examples of Solved Questions	123
4.9	Examples of Unresolved Questions	124
4.10	Examples of Unanswered Questions	126
4.11	Outlier Examples in Label Establishment	130
5.1	Motivating Example of Query Mismatch	139
5.2	Motivating Example of Information Overload	140
5.3	Workflow of Que2Code	143
5.4	QueryRewriter Workflow	145
5.5	CodeSelector Workflow	147
5.6	Preference-Pairs Construction	149
5.7	CodeSelector Workflow	151
5.8	Volinplots of Question Distribution for Python(left) and Java(right) Dataset	156
5.9	Parameter Tuning Analysis	166
5.10	Robustness Analysis on Python (left) and Java (right)	176
5.11	Example of Question Relevance User Study	179
5.12	Example of Code Usefulness User Study	182
5.13	Qualitative Analysis	186
5.14	Prototype of Que2Code	190

List of Tables

1.1	Answer Hungry Statistics in SQA Sites	4
3.1	Dataset Statistics	52
3.2	Number of Training/Validation/Testing Samples	52
3.3	Clone Detection Analysis	55
3.4	Automatic evaluation(Python dataset)	61
3.5	Automatic evaluation(Java dataset)	61
3.6	Automatic evaluation(Javascript dataset)	61
3.7	Automatic evaluation(C# dataset)	62
3.8	Automatic evaluation(SQL dataset)	62
3.9	Automatic evaluation(CODE-NN dataset)	68
3.10	Human Evaluation(Python dataset)	69
3.11	Human Evaluation(Java dataset)	69
3.12	Practical Manual Evaluation (Python dataset)	71
3.13	Practical Manual Evaluation (Java dataset)	72
3.14	Ablation evaluation (Python dataset)	76
3.15	Ablation evaluation (Java dataset)	76
3.16	Vocab Size & Training Time(per epoch)	77
4.1	Number of Training/Validation/Testing Samples	103
4.2	Automatic evaluation (Ask Ubuntu)	110

4.3	Automatic evaluation (Super-User)	111
4.4	Automatic evaluation (SO-Python)	111
4.5	Automatic evaluation (SO-Java)	111
4.6	Ablation Evaluation (Ask Ubuntu)	113
4.7	Ablation Evaluation (Super User)	113
4.8	Human Evaluation of Question Boosting Results	118
4.9	Human Evaluation - Ask Ubuntu	121
4.10	Human Evaluation - SO (Python)	121
5.1	Duplicate Questions Statistics	155
5.2	Effectiveness evaluation (Python)	159
5.3	Effectiveness evaluation (Java)	159
5.4	Context Analysis of $P@1$ (Python)	163
5.5	Context Analysis of $P@1$ (Java)	163
5.6	Ablation Analysis	164
5.7	QC Dataset Statistics	168
5.8	Effectiveness evaluation of CodeSelector (Python)	171
5.9	Effectiveness evaluation of CodeSelector (Java)	172
5.10	Component-Wise Evaluation (Python)	173
5.11	Component-Wise Evaluation (Java)	174
5.12	User Study on Question Relevance	181
5.13	User Study on Code Usefulness	185

Chapter 1

Introduction

1.1 Background

Community Question and Answer. The past decade has witnessed significant social and technical value of Community Question and Answer (CQA) platforms. Enormous amount of knowledge sharing occurs every day in CQA communities, members of these communities can ask/answer questions and search through the archived historical question-answer (QA) pairs. The CQA systems have made a substantial headway in answering complex questions, such as reasoning, advice-seeking and open-minded questions. The CQA systems are perceived mainly as a successful example of collective intelligence due to their high popularity, millions of answered questions as well as universal availability. There are two types of CQA communities:

- **General CQA sites.** The general CQA sites are community based question answering systems for general topics (e.g., Yahoo! Answers¹, Quora², Zhihu³). For these general CQA sites, users may ask questions of any topic, “*What is Melbourne famous for?*”. The general CQA sites have little restrictions, to answer this kind of question, no much professional knowledge is needed.
- **Domain CQA sites.** Besides the general CQA sites, domain CQA sites are related

¹<https://answers.yahoo.com/>

²<https://www.quora.com/>

³<https://www.zhihu.com/>

to a specific domain area (e.g., HealthTap, StackOverflow). Regarding the domain CQA systems, the questions and answers are mostly within the same topical domain, enabling more comprehensive interaction between users on fine-grained topics. In such systems, users are more likely to ask questions on professional topics and answer questions matching their own expertise. Only experts with professional domain knowledge are able to answer questions like, “*How do I check if a list is empty?*” from Stack Overflow, or “*How do I know when skipped heart beats are dangerous?*”. Compared with general CQA sites, professional knowledge is needed.

Software Question Answering Community. In this thesis, we focus on a special type of domain CQA sites, i.e., **Software Question Answering (SQA)** communities. SQA sites are designed for software developers, and the questions will be answered by users who are software domain experts. The SQA sites are heavily used by software developers as a popular way to seek programming-related information from peers. The SQA community itself has a diverse set platforms covering different software engineering topics, such as Stack Overflow (with a focus on programming-related questions), Ask Ubuntu (with a focus on Ubuntu operating system), Super User (with a focus on computer software and hardware) and Server Fault (with a focus on servers and networks) etc,. Taking Stack Overflow as an example, Stack Overflow aims to serve diverse software development topics on tools, platforms and other related software development issues. The Stack Overflow community serves more than 40M professionals and novice developers every moths and has more than 6M registered users, approximately 12M questions, 20M answers, 51M comments and 46K tags on various issues/topics of software development.

1.2 Motivations and Objectives

Motivations. Currently, the existing SQA systems are perceived mainly as a successful example of collective intelligence due to their high popularity, millions of archived questions and answers and universal availability. In spite of – or perhaps even because of – the great success of SQA and active user participation, the phenomenon of being “*answer hungry*” is still one of the biggest problems within these SQA platforms. This concept means that a very large number of questions posted in SQA remain *unanswered and/or unresolved*. In order to find out the degree of the “*answer hungry*” problem for SQA sites, we quantitatively analyzed the prevalence of this problem in real SQA sites. We choose three popular SQA site (e.g., Stack Overflow, Ask Ubuntu and Super User) for our empirical study. Since it is too expensive to run the empirical experiment on the whole Stack Overflow dataset, we only focus on Python and Java related programming languages in Stack Overflow, which refer to *SO(Python)* and *SO(Java)* respectively.

To investigate the proportion of the *unanswered* and *unresolved* questions, we first counted the number of questions that have received at least one answer, and refer to these questions as *Answered Questions*. Questions not receiving any answers are referred to as *Unanswered Questions*. For those *Answered Questions*, we further divided them into two groups of *Resolved Questions* and *Unresolved Questions* based on whether any answer within the question thread has been marked or not as the accepted answer by the asker. Then, we further empirically studied the average time interval for accepting an answer, which is the time difference between the time a question is created and the time an answer post is accepted. Table 1.1 presents the statistical results of our collected data⁴. From the table, we have the following observations:

1. **A large proportion of questions do not receive any answers in these technical SQA sites.** Consider Ask Ubuntu and Super User as examples – around 22% ques-

⁴For duplicated questions, we only keep the master ones, and remove the others.

Table 1.1: Answer Hungry Statistics in SQA Sites

Ask Ubuntu	# Questions	315,924
	# Unanswered Questions	69,528
	# Resolved Questions	106,301
	# Unresolved Questions	140,095
	Avg Accepting Time	18.63 (days)
Super User	# Questions	380,940
	# Unanswered Questions	73,584
	# Resolved Questions	160,200
	# Unresolved Questions	147,156
	Avg Accepting Time	25.69 (days)
SO (Python)	# Questions	1,236,748
	# Unanswered Questions	175,859
	# Resolved Questions	674,360
	# Unresolved Questions	386,529
	Avg Accepting Time	7.78 (days)
SO (Java)	# Questions	1,581,814
	# Unanswered Questions	213,963
	# Resolved Questions	808,040
	# Unresolved Questions	559,811
	Avg Accepting Time	8.52 (days)

tions in Ask Ubuntu and 19% questions in Super User do not get any response since the time questions have been created, leaving the askers unsatisfied.

2. **A large amount of questions are still unresolved.** For instance, 31.3% questions in SO (Python) and 35.4% questions in SO (Java) remain to be unresolved. This phenomenon is probably caused by the following reasons: (a) no good answer was provided within the current question thread, (b) even provided with good answers, it is common for the less experienced users to forget marking a potential answer as a solution.
3. **Developers usually have to wait a long time before getting satisfied answers to their questions.** It takes on average more than 18 days and 25 days to receive an accepted answer in Ask Ubuntu and Super User SQA sites respectively, and the Stack Overflow developers have to wait more than one week before getting a suitable answer to their programming tasks, which is far too slow and costly to adapt to the rapidly software development.

Research objectives. In summary, the *answer hungry* phenomenon widely exists and has been one of the biggest challenges in technical Q&A forums. We attributed this *answer hungry* problem to the following several reasons: (1) the questions posted by developers are of poor quality (e.g., ambiguous, inaccurate or uninformative), which may discourage the potential experts. (2) the SQA search engine are hardly recommend the appropriate answers to the unanswered and/or unresolved questions. (3) even though the appropriate answer is returned by the search engine, developers may fail to filter out irrelevant results and tend to get lost in the massive amount of information. Three key research questions are raised regarding the above three reasons as follows:

- **Research Question 1:** *How can we improve the quality of questions in SQA community?*

- **Research Question 2:** *How can we recommend appropriate answers to technical questions?*
- **Research Question 3:** *How can we search for the best code solutions in SQA for a programming task?*

1.3 Thesis Scope

This thesis focuses on this open issue of *answer hungry* problem in SQA sites. We aim to build intelligent and efficient learning systems that can alleviate this *answer hungry* problem in SQA sites. Considering the tremendous number of historical data, as time goes on, have been archived in the software QA communities, it is therefore preferable to build system by exploring the data accumulated in software Q&A sites. In particular, we developed three intelligent systems to address the aforementioned research questions respectively:

- **CODE2QUE** – Improving question titles from mined code snippets.
- **DEEPANS** – Best answer recommendation via question boosting.
- **QUE2CODE** – Semantic code search in Stack Overflow.

CODE2QUE (Chapter3). Sometimes it is hard for developers, especially the novice users, to describe the problem they meet due to their lack of knowledge or terminology behind the problems, this leads to a significant number of questions in SQA sites are of low-quality and not attractive to other potential experts. We thus proposed a system, named **CODE2QUE**, which can help developers in writing high quality questions by automatically generating question titles for a given code snippet using a deep sequence-to-sequence learning approach. **CODE2QUE** is fully data-driven and uses an *attention* mechanism to perform better content selection, a *copy* mechanism to handle the rare-words problem and

a *coverage* mechanism to eliminate word repetition problem. We evaluate **CODE2QUE** on Stack Overflow over a variety of programming languages (e.g., Python, Java, Javascript, C# and SQL) and the experimental results show that our approach significantly outperforms several state-of-the-art baselines in both automatic and human evaluation.

- *This work has led to a research paper – **Generating question titles for Stack Overflow Posts from mined code snippets**, which has been published in the ACM Transactions on Software Engineering and Methodology (TOSEM) in 2020.*
- *This work also led to an associated tool demo paper – **Code2Que: A tool for improving question titles from mined code snippets in stack overflow** presented at The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) in 2021.*

DEEPANS (Chapter4). One of the major reasons for the “*answer hungry*” problem in SQA community is the search engines are hardly to recommend relevant answers for the unanswered and/or unresolved questions. However, with the rapid growth of the SQA community, the SQA communities have accumulated a huge amount of question answer (QA) pairs. Therefore, developers have large chances to directly get the answers by searching from the repositories, rather than the time-consuming waiting. In this work we proposed a novel neural network based approach, named **DEEPANS**, to identify the most relevant answer among a set of answer candidates. Our approach follows a three-stage process: question boosting, label establishment and answer recommendation. Given a post, we first generate a clarifying question as a way of question boosting. We automatically establish the *positive*, *neutral*⁺, *neutral*⁻ and *negative* training samples via label establishment. When it comes to answer recommendation, we sort answer candidates by the matching scores calculated by our neural network-based model. Our experimental results show that our approach significantly outperforms several state-of-the-art baselines in both automatic

evaluation and human evaluation.

- *This work has led to a research paper – **Technical Q&A Site Answer Recommendation via Question Boosting**, which has been published in the ACM Transactions on Software Engineering and Methodology (TOSEM) in 2021.*

QUE2CODE (Chapter5). Following our previous work, sometimes even if the search engine successfully returns a target post for a given user question, there is often too much noisy and redundant information associated with the returned posts, and the code solution can be easily buried in all this overwhelming amount of information and developers may get lost with the irrelevant information. There is a need need by developers to only receive the most suitable and useful code solutions to their current programming tasks. In this work, we present a query-driven code search tool, named **QUE2CODE**, that identifies the best code solutions for a user query from Stack Overflow posts. Both the automatic and human evaluation results demonstrate the promising performance of our approach over a set of state-of-the-art baselines.

- *This work has led to a research paper – **I Know What You Are Searching For: Code Snippet Recommendation from Stack Overflow Posts**, which has been submitted to the ACM Transactions on Software Engineering and Methodology (currently under major revision).*

1.4 Thesis contribution

This section summarizes the main contributions of the thesis. This thesis builds on top of materials from our past papers about **CODE2QUE**, **DEEPANS** and **QUE2CODE**.

Chapter3 introduces the **CODE2QUE**, a tool for improving question titles from mining code snippets in Stack Overflow, we make the following contributions:

1. We propose a novel question generation task based on sequence-to-sequence learning approach, which can assist developers in writing high-quality question titles from given code snippets. Our approach is enhanced with *attention*, *copy* and *coverage* mechanism to perform better content selection, manage rare words in the input corpus and avoid meaningless repetitions. To the best knowledge, this is the first work which investigates the possibility of improving the low-quality questions in Stack Overflow.
2. We performed comprehensive evaluation on one of the most popular SQA communities, i.e., Stack Overflow, to demonstrate the effectiveness and superiority of our approach. Our system outperforms strong baselines by a large margin and achieves state-of-the-art performance.
3. We collected more than 1M $\langle \text{code snippet}, \text{question} \rangle$ pairs from Stack Overflow, which covers a variety of programming languages (e.g., Python, Java, Javascript, C# and SQL). We have released our code⁵ and datasets [Gao, 2020] to facilitate other researchers to repeat our work and verify their ideas. We also implemented a web service tool, named CODE2QUE to facilitate developers and inspire the follow-up work.

Chapter4 introduces the **DEEPANS**, a tool for recommending answers to the unanswered/unresolved questions of the SQA sites, in this work we make the following contributions:

1. Previous studies neglect the value of interactions between the question asker and the potential helper. We argue that a clarifying question between the question and answers is an important aspect of judging the relevance and usefulness of the QA pair. Therefore, we train a sequence-to-sequence model to generate useful clarifying

⁵<https://github.com/beyondacm/Code2Que>

questions for a given post, which can fill the lexical gap between the questions and answers. To the best of our knowledge, this is the first successful application of generating clarifying questions for technical Q&A sites.

2. We present a novel method to constructing *positive*, *neutral*⁺, *neutral*⁻, *negative* training samples via four heuristic rules, which can greatly save the time consuming and labor intensive labeling process.
3. We develop a weakly supervised neural network model for the answer recommendation task. For any question answer pairs, we fit the QA pair into our model to calculate the matching score between them; the higher matching score is estimated by our model, the better chance the answer will be selected as the best answer. In particular, the Q&A sites can employ our approach as a preliminary step towards marking the potential solution for the unanswered/unresolved question. This can avoid unnecessary time spent by developers to browse questions without an accepted solution.
4. Both our quantitative evaluation and user study show that DEEPANS can help developers find relevant technical question answers more accurately, compared with state-of-the-art baselines. We have released the source code of DEEPANS and the dataset of our study to help other researchers replicate and extend our study.

Chapter5 introduces the **QUE2CODE**, a tool for searching code solutions from the Stack Overflow posts, in this work we make the following contributions:

1. All previous studies of question routing in CQA systems work on finding similar questions. However, it is hard to measure the relevance between different questions automatically and experts are often asked to manually rate the relevance score. In our study, we propose a new task of semantically-equivalent question retrieval. By utilizing duplicate question pairs archived in Stack Overflow, we present a novel model

and an evaluation method to automatically evaluate the semantically-equivalent questions without a labor-intensive labeling process.

2. All current studies that have investigated code snippet searching rely on calculating a matching score between a query and a code snippet. We argue that code snippet recommendation is more about predicting relative orders rather than precise relevance scores. Hence, we propose a novel pairwise learning to rank model to recommend code snippet from Stack Overflow posts, and we first use the BERT model for searching for code snippets in Stack Overflow.
3. Our experimental results show that our QUE2CODE is more effective for code snippet recommendation than several state-of-the-art baselines. We have released the source code of our QUE2CODE and our dataset to help other researchers replicate and extend our study.

In summary, this Ph.D. thesis proposed three novel models, i.e., CODE2QUE, DEEPANS, and QUE2CODE to address the “*answer hungry*” problem in SQA community. These three models deal with the “*answer hungry*” problem from three perspectives: (i) helping developers to write high-quality question titles for a given code snippet; (ii) recommending best answers to the unanswered/unresolved questions in SQA sites; (iii) searching the useful code solutions for newly posted questions in SQA community. With the help of our proposed models, (i) developers can use our tool CODE2QUE to write better question titles to describe the problem they met, which is able to obtain attention from potential experts; (ii) developers can use our tool DEEPANS to find appropriate answers to their unanswered/unresolved questions, which can help them to get answers from the repositories rather than time-consuming waiting. (iii) developers can use our tool QUE2CODE to search code solutions to their newly posted questions directly, which can save their time from browsing the irrelevant results and getting lost in the massive amount of information.

1.5 Structure of the thesis

In this thesis, Chapter 1 introduces the introduction and motivation about the thesis. Chapter 2 surveys the related literature which includes different software engineering tasks and qualitative study in mining SQA sites, such as Stack Overflow. In Chapter 3, we present **CODE2QUE**, an approach to assist developers in writing clear and informative question titles for a given code snippet. In Chapter 4, we present **DEEPANS**, an approach to help developers effectively identify the relevant answer among a set of answer candidates. In Chapter 5, we present **QUE2CODE**, a Stack Overflow code search engine to recommend code snippets for developer's query questions. Chapter 6 gives the conclusion, some limitations and future research directions.

Chapter 2

Literature Review

In this chapter, we provide the necessary background to understand the purpose, key concerns, and technical details of the three research studies we conducted in this dissertation. We conduct a systematic literature review about mining the SQA sites in the context of software development. We categorized the relevant research into two main categories: mining SQA community and usage and deep learning for software engineering. It can help the readers to get a systematic overview of the work in general and specifically to know future research directions.

2.1 Software Question Answering Community

The community question and answer (CQA) platforms have become one of the fastest-growing and user-generated-portals (UCG). The CQA system has risen as an enormous market for the fulfillment of complex information needs. Compared with the general CQA sites, which covers a wide range of general topics (e.g., Yahoo! Answers, Quora, Zhihu), this thesis focuses on the software question answering (SQA) community. Software question and answering sites are designed for the users of software developers, developers will search and post their daily problems on SQA sites and these questions will be answered by users who are software domain experts. Developers may resort to different SQA platforms for solving different problems. For example, developers often go to Stack

Overflow (<https://stackoverflow.com/>) to seek help of programming-related questions; they may utilize Ask Ubuntu (<https://askubuntu.com/>) when they encounter ubuntu system related problems; if they met problems with respect to servers and networks, developers can post their problems on Server Fault platform (<https://serverfault.com/>); the software and hardware engineers may resort the Super User (<https://superuser.com/>) if they are computer power users. Overall, the SQA community has been heavily used by developers as a popular way to seek information and support via the internet.

Taking Stack Overflow, one of the most popular SQA platforms, as an example, we illustrate the user scenarios of using SQA platforms as follows. Stack Overflow allows users to register, create posts (i.e., questions and answers), leave comments on posts (either questions or answers), revise posts, vote on posts, add tags to posts, and search or browse the posts created by others. The SQA community users can provide code snippets or other resources (e.g., screenshots or urls) to better present their questions. The other domain experts can answer the existing posted questions according to their experience. Each question may receive multiple answers from different domain experts, however, only one answer can be marked as “accepted answer” by the asker who created this post, which means this answer correctly solved the asker’s problem. The score of a post (i.e., a question or and answer) represented the up votes and down votes this post obtained. Fig. 2.1 shows an example of a question and its associated accepted answer in Stack Overflow. The developer posted a question, i.e., “*How can I add new keys to a dictionary*”, on June 21, 2009 and received an answer from another domain expert. Then this answer was marked as accepted by the asker. This question received 3,150 votes from the community members and was associated tags “*python*”, “*dictionary*”, “*lookup*”. The answerer provided code snippet to solve this problem which received 4021 votes from the the community members.

How can I add new keys to a dictionary?

Asked 12 years, 4 months ago Active 16 days ago Viewed 4.5m times

Question Titles

Is it possible to add a key to a Python dictionary after it has been created?

3150

It doesn't seem to have an `.add()` method.

python dictionary lookup

Question Tags

Asker and creation date

556


Share Follow

Question Votes Last Edited date and User

edited Feb 24 '20 at 15:39

 Peter Mortensen
29k ● 21 ● 96 ● 124

asked Jun 21 '09 at 22:07

 Ifaraone
46.1k ● 16 ● 48 ● 70

53 If you are here trying to figure out how to add a key *and* return a *new* dictionary, you can do this on python3.5+: `{**mydict, 'new_key': new_val}`. – cs95 Dec 19 '18 at 13:17

Either use the accepted answer: `d['mynewkey'] = 'mynewvalue'` or you could use `val = d.setdefault('mynewkey', 'mynewvalue')` – Ol Dawg Jan 7 '20 at 19:32

2 If you just use `<dictionary>[<key>] = <value>` and if the key doesn't exist, the key will be automatically added. – Jyotirmay Kumar Jha Dec 30 '20 at 5:23

Re @cs95, this also creates a new dict with an additional key: `dict(existing_dict, new_key=new_val)` per stackoverflow.com/a/46647897/1840471 – Max Ghenis Oct 1 at 13:38

@MaxGhenis thanks, that works well for python < 3.5 (with a caveat), I've added a community wiki answer [here](#). – cs95 Oct 1 at 21:13

Add a comment

Question Comments

21 Answers

Accepted Answer

Active Oldest Votes

You create a new key/value pair on a dictionary by assigning a value to that key

4021

```
d = {'key': 'value'}
print(d) # {'key': 'value'}

d['mynewkey'] = 'mynewvalue'

print(d) # {'key': 'value', 'mynewkey': 'mynewvalue'}
```

Answer code block

If the key doesn't exist, it's added and points to that value. If it exists, the current value it points to is overwritten.

last edited date and user

answerer and creation date

Share Follow

edited Feb 12 at 9:08

 Boris
8,030 ● 7 ● 68 ● 69

answered Jun 21 '09 at 22:09


 Paolo Bergantino
457k ● 76 ● 514 ● 433

Figure 2.1: Example of questions and answers in SQA community

2.2 Mining SQA Community

The key aspects of SQA sites are **Questions**, **Answers** and **Users**. For example, the Stack Overflow community itself serves more than 40M professionals and novice programmers every month and has more than 6M registered users, approximately 12M questions, and more than 20M answers on various issues/topics of software development. Software engineering researchers are investigating to explore these different aspects of the SQA to solve different problems, e.g., *how to improve question quality in SQA community and improve the accuracy in question retrieval, how to classify SQA answers and evaluate their quality?, how to motivate community members for long-term participation?*. In this section, we discuss different studies with respect to the above three key aspects (i.e., **Questions**, **Answers**, **Users**).

Software QA Community Questions. The success of SQA community depends heavily on the willingness of the developers to post their questions and experts to answer others' questions. High-quality questions help to improve the popularity of the SQA community and contribute to efficient problem solving. Therefore, the importance of high-quality questions in SQA sites has been recognized and investigated in many studies.

Nasehi et al. [Nasehi et al., 2012] performed a qualitative study to investigate the feature of code fragments in Stack Overflow posts. They found that additional explanations with code examples are as useful as the code examples themselves. Their findings can be used to create more usable artifacts for potential users. After that, Yao et al. [Yao et al., 2013] investigated the question quality in SQA sites, the results showed that there is a correlation between the question quality and answer quality, i.e., good answers are more likely to be given in response to good questions. Based on their observations, they also used a family of algorithms to predict the quality of questions and answers, which can be used to identify the high-quality and low-quality posts in their early stage.

Ganguly et al [Ganguly and Jones, 2015] investigated the problem of retrieving a set of questions which are most likely to be related to a newly created question. It is thus likely for a moderator to link the previous questions to the new one. They viewed this problem as an Information Retrieval task where the intention is to retrieve a list of relevant documents in response to a new query, and they proposed a supervised model (SPLDA) to estimate the distributions of tags per topic. Ponzanelli et al [Ponzanelli et al., 2014] presented an approach to classify technical forum questions as well as understand what fundamentally influences and characterizes it. They devised three sets of metrics to capture textual features of questions (e.g., from simple textual features to more complex readability metrics) and the reputation of asker (e.g., popularity of a user in the community). They also empirically studied how the designed metrics influence the quality of Stack Overflow questions and found that the metrics of author's popularity are the best predictors of a question's quality. Their approach can identify poor-quality questions at their creation time, thus reducing the effort and time cost of the reviewing process.

Jiarpakdee [Jiarpakdee et al., 2016] investigated several properties/features on question quality and which of them has the most impact by creating prediction models that can forecast whether a question is expected to get any answers. From the evaluation results, they conclude that community-based and affective features play an important role in the question quality identification, e.g., *Favorite Vote Count* from community-based features is the most influential feature. Squire et al. [Squire and Funkhouser, 2014] explored the role of source code and non-source code text on Stack Overflow. They discussed whether the presence of source code (or how much) actually will produce the "best" Stack Overflow posts. They found that high scoring questions tend to have a lower code-text ratio than do high scoring answers. They thus recommended one part of code to every three parts of text (1:3) in Stack Overflow answers is ideal, while a ratio of about 1:9 in Stack Overflow questions is ideal.

Baltadzhieva et al. [Baltadzhieva and Chrupała, 2015] investigated which features (e.g., question tags, length of question title and body, presence of code snippet and the user reputation) can influence question quality, they also explored the lexical terms that determine the high and low quality questions. They found that the terms predicting high quality are terms expressing excitement, negative experience or terms regarding exceptions. Terms predicting low quality questions are terms containing spelling errors or indicating off-topic questions and interjections. Duijn et al. [Duijn et al., 2015] proposed an approach to improve the classification of high and low quality questions based on the analysis of code fragments in Stack Overflow questions. They verified the code-to-text ratio is one of the most important factor, they also found that several metrics related to code readability also contribute to the quality of a question. Arora et al. [Arora et al., 2015] proposed an approach to perform question classification by retrieving similar previously asked questions, and then using the text from these previously asked similar questions to predict the quality of the current question. By automating flagging questions as good or bad can speed up the moderation process and thus saving time and human effort.

Asaduzzaman et al. [Asaduzzaman et al., 2013] empirically investigated the reasons why Stack Overflow questions are left unanswered. They found several significant reasons, such as unable to find experts, small length and unclear questions, propriety technology questions, without code snippets example, off-topic questions etc.,. They provided a taxonomy for the unanswered questions to help understand why there is a delay in answering questions and whether a question requires further information. They also built a classifier to predict how long a question will remain unanswered in Stack Overflow. Moderators can use their classifier to predict questions likely to have delayed responses and can consider taking actions to promote an earlier response, such as editing the question or routing the question to an expert. Correa et al. [Correa and Sureka, 2013] conducted the first study regarding the “closed” questions in Stack Overflow. They used a machine

learning framework and built a predictive model to identify “closed” question at the time of question creation. Feature analysis revealed that *Stack Overflow urls* and *code snippet length* as the top differentiating features to predict a “closed” question. Galina et al. [Lezina and Kuznetsov, 2013] presented a classifier to predict whether or not a question will be closed given the question as submitted, along with the reason why the question was closed. Rahman et al. [Rahman and Roy, 2015] investigated 3,956 unresolved questions by using an exploratory study, they also proposed a prediction model by employing five metrics related to user behaviour, topics and popularity of questions. Their approach can predict if a best answer for a question in Stack Overflow might remain unaccepted or not. Saha et al. [Saha et al., 2013] investigated the reasons why the questions remain unanswered by applying a combination of statistical and data mining techniques. They found that quality attributes such as number of views, favorites, scores and questioners’ reputation are useful in predicting whether a question will be answered or not. Their preliminary study indicate that most questions remain unanswered because they apparently are of little interest to the SQA community. Correa et al. [Correa and Sureka, 2014] investigated the the phenomena of “deleted questions” in Stack Overflow to gain insights about the nature of poor quality questions. They developed a framework to predict the the probability of a question to be deleted at the time of question creation. They experimented with 47 features based on the member profile, community, content, and stylistic features. Their approach can help SQA community to detect questions with poor quality and also give immediate feedback to his/her questions, which can help those asking to revise their questions and improve them in order to avoid deletion. Xia et al. [Xia et al., 2016] proposed a two stage hybrid approach, named *DelPredictor*, which combines text processing and classification techniques to predict deleted questions in Stack Overflow.

Prior works have also investigated how to improve the question tagging system in SQA community. Saha et al. [Saha et al., 2013] mined the data from Stack Overflow questions

and used a discriminative model to automatically suggest relevant question tags for the questioner. Their approach can suggest tags closely related to the tags added by the users, and find missing tags (the tags that are neglected to be added by the users) for the questioner. Xia et al. [Xia et al., 2013] developed a model, named *TagCombine*, to perform tag recommendation by analysing objects in SQA community. *TagCombine* used three different components (i.e., multi-label ranking component, similarity based ranking component, tag-term based ranking component) to calculate the matching score for different question tags. Rekha et al. [Rekha et al., 2014] presented a hybrid auto-tagging system for Stack Overflow. Their proposed system is capable of recommending tags to the questioner as soon as the question is posted on Stack Overflow based on the content of the question. Following that, Wang et al. [Wang et al., 2018] developed a model, named *ENTAGREC++*, which integrated the historical tag assignments to software objects, as well as the information of the users, and initial set of tags that a user may provide for tag recommendation.

Software QA Community Answers. Gkotsis et al. [Gkotsis et al., 2014] proposed a method to address the problem of determining the best answer in SQA sites by focusing on the content. They perform the best answer identification and prediction by solely using various shallow textual features, they ran an evaluation on 21 StackExchange websites covering around 4 million questions and more than 88 million answers, the experimental results showed their approach is robust, effective, and widely applicable. Gantayat et al. [Gantayat et al., 2015] studied the synergy between the act of acceptance of an answer and voting by the community. They found that 81% of the Stack Overflow questions have multiple answers, the accepted answer is also the top-voted answer, which means the number of votes on a post plays an influencing factor on their choice of acceptance. They also found that in the cases where the accepted answer is not the top voted answer, the asker is biased towards longer posts that use similar terms and include code snippets, while

community votes lean towards short, concise answers that include external links and have a better readability score.

Calefato et al. [Calefato et al., 2015] investigated how Stack Overflow users can increase the chance of getting their answer accepted. They focused on actionable factors that can be acted upon by users when writing an answer and making comments. They found the evidence that related to information presentation, time and affect all have an impact on the success of answers.

Singh et al. [Singh and Simperl, 2016] proposed the Suman system to detect the answers for the unanswered questions posted on Stack Overflow. The Suman system used the combination of keywords based on semantic search, text based search and crowdsourced data to search for relevant answers. The system is also capable of recommending the expert who are proficient in answering these questions, thus help to reduce the queue of unanswered questions. Their evaluation also indicated that the participants approved the algorithm rating given to the unanswered questions. Yao et al. [Yao et al., 2015] proposed the early detection of high-quality questions/answers. Such detection can help discover high-quality questions that would be widely recognized by the users, as well as identify high-quality answers that would gain much positive feedback from users.

Xu et al. [Xu et al., 2017] proposed a model, named *AnswerBot*, to help developers quickly capture the key points of several answer posts relevant to a technical question before they read the details of the posts. Their approach contains three main steps, i.e., relevant question retrieval, useful answer paragraph selection, diverse answer summary generation. The experimental results showed that the answer summaries generated by their approach are relevant, useful and diverse to developer's technical questions, and can effectively retrieve relevant questions and salient answer paragraphs for summarization.

Zheng et al. [Zheng and Li, 2017] proposed a new approach to predict the best answers to the questions raised on Stack Overflow by exploring the heterogeneous data sources

on the forum. They extracted different groups of features from multiple data sources and combined them for final prediction via multi-view learning. Experimental results show that their proposed model is effective for identifying the best answers in Stack Overflow.

Calefato et al. [Calefato et al., 2019] conducted an empirical study of assessment of best-answer prediction models in SQA sites. They approached the best-answer prediction problem as a binary-classification task. They assessed 26 best-answer prediction models in two steps, they first studied how the models perform when predicting best answers in Stack Overflow, they then assessed the model's performance in cross-platform settings, where the prediction models are trained on Stack Overflow and tested on other technical QA sites.

Novielli et al. [Novielli et al., 2014] assessed the suitability of the state-of-the-art sentiment analysis tool, already applied in social computing, for detecting affective expressions in Stack Overflow. they examined the role of emotional lexicon regarding the 7M Stack Overflow questions. They found that the questions on Stack Overflow do have an emotional style, which affects answer quality and the time to respond.

Software QA Community Users. The Software QA community is one of the most popular community due to its number of registered users, daily visits, and above all the satisfaction level of users. Prior works have investigated possible ways of expert identification by exploring the user files in SQA community.

Ginsca et al. [Ginsca and Popescu, 2013] performed an in-depth analysis of the user profile information in Stack Overflow, they found that the more complete a user profile is, the higher the chances are for that user to provide good quality answers. They have also presented a novel answer ranking model which incorporated the user profile information or user activities. Tian et al. [Tian et al., 2013b] proposed an approach to predict the best answerer for a new question on SQA site by using both user interest and user expertise relevant to the topics of the given question.

Chang et al. [Chang and Pal, 2013] presented a question routing scheme which considers compatibility, availability and expertise of users. Their approach focused on routing questions to a group of users, i.e., the users are willing to collaborate and provide useful answers to the questions. Their experimental results on Stack Overflow datasets showed the effectiveness of routing a question to a team of compatible users.

Riahi et al. [Riahi et al., 2012] focused on finding experts for a newly posted question. They first built expert profiles based on their answering history, these profiles are then used in comparison with a newly posted question. They modeled the interests of users by tracking their answer history in the community, for each user, a profile is created by combining those questions answered by the user. Based on the user profiles, the relation between the answerer and a new question is measured by using a number of different methods (e.g., Dirichlet smoothing, TF-IDF, the Latent Dirichlet Allocation (LDA) and Segmented Topic Model (STM)).

Posnett et al. [Posnett et al., 2012] studied empirically tenure of posters and quality of answers in Stack Exchange community. They found that as the community grows in numbers, the overall quality of answers decreases in general, indicating more and more answers are given by the non-expert and higher scores are more difficult to obtain. They also found that the number of prior posts do not increase one's answer quality, which means that expertise is present from beginning, and doesn't increase with the time spent with the community.

Morrison et al. [Hanrahan et al., 2012] have studied the correlation between age and Stack Overflow reputation, they investigated the degree to which older age programmers obtain knowledge available latest technology. The output indicated that the reputation score of programmers gets higher well into the age of 50s, and at the age of 30s leans towards exploring less new area compared to those who are younger or older from them.

Yang et al. [Yang et al., 2013] proposed a Topical Expertise Model to jointly model

topics and expertise in SQA sites. They proposed *CQARank* to textual features with link analysis for deriving the user expertise and interests score under various topics. Their model is applicable for various tasks such as expert finding, relevant answer retrieval and similar questions recommendation.

Yang et al. [Yang et al., 2014b] contributed a novel metric for expert identification in SQA communities, which provides a better characterisation of users' expertise by focusing on the quality of their contributions. They identified two groups of users, namely sparrows and owls, their experimental results contribute new insights to the study of expert behaviour in QA platforms.

Murgia et al. [Murgia et al., 2016] investigated the possibility of building human-bot interaction in Stack Overflow. They developed a bot emulating an expert user responsible to answer the questions related to resolving 50 Git error messages in Stack Overflow posts.

The users of SQA community gain specific rewards and badges based on their knowledge contribution to the community. The reputation of expert in SQA community is important for other to decide to accept or reject a solution for a specific issue. There are a lot of studies exploring the role of reputation and reward system on SQA community.

Grant et al. [Grant and Betts, 2013] investigated how user badges, a collection of visible awards associated with public user profiles, can be used to influence user behaviour. The results of these works confirmed that badges drive users to behavior in the ways entirely consistent as predicted, i.e., an increase in user activity was noticed before a badge is earned compared to the duration onwards. Besides, the users who receive badges, a significant number of change is made regarding their contribution compared to those who have not earned any badges. Bosu et al. [Bosu et al., 2013] analyzed the Stack Overflow data from four perspectives to understand the dynamics of reputation building on Stack Overflow. Their experimental results provide guidelines to new Stack Overflow contributors who want to earn high reputation scores quickly.

Bazelli et al. [Bazelli et al., 2013] investigated the question and answers in Stack Overflow to determine the developer’s personality traits via using the Linguistic Inquiry and Word Count (LIWC). They explored the personality traits of Stack Overflow authors by categorizing them into different groups based on their reputations. Based on the textual analysis of Stack Overflow posts, they found that top reputed authors are more extroverted compared to medium and low reputed users. They also found that authors of up voted posts express significantly less negative emotions than authors of down voted posts.

Hart et al. [Hart and Sarma, 2014] investigated the role of social reputation and other characteristics plays in the information filtering process of technical novices in the context of Stack Overflow. The experimental results show that technical novices assess information quality based on the intrinsic qualities of the answer, such as presentation and content, suggesting that novices are wary to rely on social cues in SQA context. Halavais et al. [Halavais et al., 2014] adapted the social learning analytics (SLA) perspective to study the network effects on badge selection in Stack Overflow. By examining the badges and “tags” used on Stack Overflow, they found that more general more general badges are closely related to tenure on this site, while numerous “tag” badges provide for more socially-determined difference.

Sinha et al. [Sinha et al., 2015] focused on the dynamics of how much the community values a user, they showed that the application of network analytic techniques, i.e., quadratic assignment procedure (QAP) that is capable of assisting in suggesting how users in SQA sites incline to fall into same reputation categories with the evolution of time.

In summary, prior works have investigated mining the SQA community from different perspectives, from questions, answers and users. This thesis focuses on mining SQA community for alleviating the “answer hungry” phenomenon, we explored different deep learning techniques to help developers to improve their question quality, searching for a better answer as well as the code solutions.

2.3 Deep Learning for Software Engineering

In recent years, an interesting research direction in software engineering is to use deep learning to solve many software engineering tasks, such as comment generation, API and answer recommendation, tag recommendation, code search, query reformulation etc.,. In this section, we included the different types of software engineering tasks and different software phases by applying deep-learning based approaches.

Software Requirements. Moran et al. [Moran et al., 2018] have developed an approach for automatically prototyping software GUIs, and implemented this approach as a tool named *ReDraw* for Android. Their approach is capable of accurately detecting and classifying GUI-components in a mock-up artifact, generating hierarchies that are similar to those that a developer, generating apps that are visually similar to mock-up artifacts, and positively impacting industrial workflows. Thaller et al. [Thaller et al., 2019] proposed *Feature Map*, a flexible human and machine-comprehensible software representation based on micro-structures. Their approach can embed the high-dimensional, inhomogeneous vector space of micro-structures into a feature map. Their evaluation result suggested that Feature Map is an effective software representation method, revealing important information hidden in the source code.

Chen et al. [Chen et al., 2018] presented a neural machine translator that combines recent advances in computer vision and machine translation for translating a UI design image into a GUI skeleton. Their approach learns to extract visual features in UI images, encode these features' spatial layouts, and generate GUI skeletons in a unified neural network framework, without requiring manual rule development.

Software Development. A lot of prior works focus on the generating suitable code representation for source code. Zhang et al. [Zhang et al., 2019] proposed a novel AST-based

Neural Network (ASTNN) for source code representation. *ASTNN* splits each large AST into a sequence of small statement trees, and encodes the statement trees to vectors by capturing the lexical and syntactical knowledge of statements. Based on the sequence of statement vectors, a Bi-RNN is used to leverage the naturalness of statements and finally produce the vector representation of a code fragment. Wang et al. [Wang et al., 2020c] presented a new graph neural architecture, named *GINN*, for learning models of source code. Their models have an easier time to distill the key features for program representation, thus can better serve the downstream tasks.

Another task of applying deep-learning models for software development is code generation. For example, Svyatkovskiy et al. [Svyatkovskiy et al., 2020] proposed *IntelliCode Compose*, which is a code completion tool of predicting sequences of code tokens of arbitrary types, generating up to entire lines of syntactically correct code. It applied the state-of-the-art generative transformer model trained on 1.2 billion lines of source code. Bunel et al. [Bunel et al., 2018] presented two novel contributions to improve the state-of-the-art program synthesis techniques. The first contribution uses Reinforcement Learning to optimize for generating any consistent program, which helps in improving generalization accuracy of the learned programs. The second contribution is that they incorporate syntax checking as an additional conditioning mechanism for pruning the space of programs during decoding. The experimental results show that incorporating syntax leads to significant improvements with limited training datasets.

Researchers also investigated to perform better code search task by using deep-learning based approach. Sachdev et al. [Sachdev et al., 2018] proposed a neural code search method which combined the of token-level embeddings and conventional information retrieval techniques TF-IDF. They found that the basic word embedding techniques can achieve good performance on code search task. Gu et al. [Gu et al., 2018] proposed a supervised technique, named DeepCS, for code searching using deep neural networks. They

used multiple sequence-to-sequence-based networks to capture the features of the natural language queries and the code snippets.

Software Testing and Debugging Deep-learning techniques have also been widely used to support the software testing and debugging process. Wen et al. [Wen et al., 2018] utilized Recurrent Neural Network (RNN), which is a deep learning technique, to encode features from sequence data automatically. They proposed a novel approach, named *FENCES*, which extracts six types of change sequences covering different aspects of software changes via fine-grained change analysis. It approaches defects prediction by mapping it to a sequence labeling problem solvable by an RNN.

Xu et al. [Xu et al., 2019] proposed a framework, named *LDFR*, by learning deep feature representations from the defect data. They use a hybrid loss function to train a DNN model to learn top-level feature representation. The experimental results show that compared with 27 baselines, *LDFR* performs significantly better in terms of five indicators.

Hoang et al. [Hoang et al., 2019] proposed an end-to-end deep learning framework, named *DeepFit*, that automatically extracts features from commit messages and code changes and use them to predict defects just-in-time. Lam et al. [Lam et al., 2017] proposed a novel approach that uses deep neural network in combination with rVSM, an information retrieval technique. rVSM collects the feature on the textual similarity between bug reports and source files. A DNN is used to relate the terms in bug reports to potentially different code tokens and terms in source files.

Li et al. [Li et al., 2019b] proposed a deep learning approach, named *DeepFL*, to automatically learn the most effective existing/latent features for precise fault localization. The experimental results on 395 real bugs showed the effectiveness of their approach in defect localization. Liu et al. [Liu et al., 2017] presented a novel deep learning approach to automatically generate the text inputs for the mobile testing. It produces the most relevant input

values in a context. They have leveraged the Word2Vec to achieve the app-independence. The evaluation over 50 IOS apps confirms the effectiveness and efficiency of our designs.

Software Maintenance Great effort has been dedicated to supporting the software maintenance tasks, such as clone detection, comment generation, self-admitted technical debt (SATD) detection. White et al. [White et al., 2016] presented a novel way to detect code clones. Their approach combined two different RNNs, i.e., RtNN and RvNN, for automatically linking patterns mined at the lexical level and patterns mined at the syntactic level. The evaluation on file-level and function-level showed the effectiveness of their approach in detecting code clones.

Gao et al. [Gao et al., 2019a] presented a tree embedding technique to conduct clone detection. Their approach first conducted tree embedding g to obtain a node vector for each intermediate node in the AST, which captures the structure information of ASTs. They then compose a tree vector from its node vectors using a lightweight method. Lastly Euclidean distances between tree vectors are measured to determine code clones.

Buch et al. [Büch and Andrzejak, 2019] developed an AST-based Recursive Neural Network. They traversed the ASTs to form data sequences as the input of LSTM, the experimental results show that simply averaging all node vectors of a given AST yields strong baseline aggregation scheme.

Researches also investigated the possibility of generating code comments for source code to better maintain the source code. Hu et al. [Hu et al., 2018] proposed a new approach, named *DeepCom*, an attention-based Seq2Seq model, to automatically generate comments for Java methods. *DeepCom* takes ASTs sequences as input. These ASTs are converted to specially formatted sequences using a new structure-based traversal (SBT) method. SBT can express the structural information and keep the representation lossless at the same time. LeClair et al. [LeClair et al., 2019] proposed a neural model that combines

the words from code with code structure from an AST. Their model processed each data source as a separate input, which allows the model to learn code structure independent of the text in code. This process provided coherent summaries in many cases even when zero internal documentation is provided.

Technical debt (TD) is terminology that developers takes suboptimal solutions to achieve short-term goals that may affect long-term software quality. Detecting the technical debt can help to improve software maintenance. Potdar et al. [Potdar and Shihab, 2014] proposed the self-admitted technical debt (SATD) concept (e.g., TODO, FIXME, HACK) for the first time, which refers to the TD introduced by a developer intentionally and documented by source code comments. Ren et al. [Ren et al., 2019] proposed a CNN-based approach for classifying code comments as SATD or non-SATD. To improve the explainability of our model's prediction results, they exploited the computational structure of CNNs to identify key phrases and patterns in code comments that are most relevant to SATD. Zampetti et al. [Zampetti et al., 2020] presented the first step towards the automated recommendation of SATD removal strategies. They built a multi-level classifier capable of recommending six SATD removal strategies. *SARDELE* combines a convolutional neural network trained on embeddings extracted from the SATD comments with a recurrent neural network trained on embeddings extracted from the SATD-affected source code. The experimental results suggested that SATD removal follows recurrent patterns and indicate the feasibility of supporting developers in this task with automated recommenders.

In summary, deep learning techniques have been widely used to support various software engineering tasks, which covers software design, software development, software testing and debugging as well as software maintenance. This thesis focuses on applying deep learning models for assisting developers using Software QA community more efficiently and effectively.

Chapter 3

Code2Que: Improving Question Titles from Mined Code Snippets

Gao, Z., Xia, X., Grundy, J.C. , Lo, D., Li, Y.Y-F. Generating Question Titles for Stack Overflow from Mined Code Snippets, ACM Transactions on Software Engineering and Methodology, Vol. 29, No. 4, September 2020, ACM. <https://doi.org/10.1145/3401026>.

3.1 Introduction

In recent years, question and answer (Q&A) platforms have become one of the most important user generated content (UGC) portals. Compared with general Q&A sites such as Quora¹ and Yahoo! Answers², Stack Overflow³ is a vertical domain SQA (Software Question Answering) site, its content covers the specific domain of computer science and programming. SQA sites, such as Stack Overflow, are quite open and have little restrictions, which allow their users to post their problems in detail. Most of the questions will be answered by users who are often domain experts.

Stack Overflow (SO) has been used by developers as one of the most common ways to seek coding and related information on the web. Millions of developers now use Stack Overflow to search for high-quality questions to their programming problems, and Stack Overflow has also become a knowledge base for people to learn programming skills by

¹<https://www.quora.com/>

²<https://answers.yahoo.com/>

³<https://stackoverflow.com/>

browsing high-quality questions and answers. The success of Stack Overflow and of community-based question and answer sites in general depends heavily on the will of the users to answer others' questions. Intuitively, an effectively written question can increase the chance of getting help. This is beneficial not only for the information seekers, since it increases the likelihood of receiving support, but also for the whole community as well, since it enhances the behavior of effective knowledge sharing. A high-quality question is likely to obtain more attention from potential answerers. On the other hand, low-quality questions may discourage potential helpers [Mamykina et al., 2011, Calefato et al., 2018, Nie et al., 2017, Anderson et al., 2012, Yang et al., 2014a, Jin and Servant, 2019].

To help users effectively write questions, Stack Overflow has developed a list of quality assurance guidelines⁴ for community members. However, despite the detailed guidelines, a significant number of questions submitted to SO are of low-quality [Correa and Sureka, 2014, Arora et al., 2015]. Previous research has provided some insight into the analysis of question quality on Stack Overflow, for example, Correa and Sureka [Correa and Sureka, 2014] investigated closed questions on SO, which suggest that the good question should contain enough code for others to reproduce the problem. Arora et al. [Arora et al., 2015] proposed a novel method for improving the question quality prediction accuracy by making use of content extracted from previously asked similar questions in the forum. More recent work [Trienes and Balog, 2019] studied the way of identifying unclear questions in CQA websites. However, all of the work focuses on predicting the poor quality questions and how to increase the accuracy of the predictions, more in-depth research of dealing with the low-quality questions is still lacking. To the best of our knowledge, this is the first work that investigates the possibility of automatically improving low-quality questions in Stack Overflow. Considering information seekers may lack the knowledge and terminology related to their questions and/or their writing may be poor, formulating a clear question

⁴<https://stackoverflow.com/help/how-to-ask>

title and questioning on the key problems could be a non-trivial task for some developers. Lacking important terminology and pool expression may happen even more often when the developer is less experienced or less proficient in English.

Among the Stack Overflow quality assurance guidelines, one of which is that developers should attach code snippets to questions for the sake of clarity and completeness of information, which lead to an impressive number of code snippets together with relevant natural language descriptions accumulated in Stack Overflow over the years. Some prior work has investigated retrieving or generating code snippets based on natural language queries, as well as annotating code snippets using natural language (e.g., [Franks et al., 2015, Allamanis et al., 2015, Giordani and Moschitti, 2009, Iyer et al., 2016, Keivanloo et al., 2014, Ling et al., 2016, Oda et al., 2015, Desai et al., 2016, Locascio et al., 2016, Yin and Neubig, 2017, Wong et al., 2013]). However, to the best of our knowledge, there have been no studies dedicated to the question generation⁵ task in Stack Overflow, especially generating questions based on a code snippet.

Fig. 3.1 shows some example code snippets and corresponding question titles in Stack Overflow. Generating such a question title is often a challenging task since the corpus not only includes natural language text, but also complex code text. Moreover, some rare tokens occur among the code snippet, such as “setUpClass” and “Paramiko” illustrated in the aforementioned examples.

We propose an approach to help developers write high-quality questions based on their code snippets by automatically generating question titles from given code snippets. We frame this question generation task in Stack Overflow as a sequence-to-sequence learning problem, which directly maps a code snippet to a question. To solve this novel task, we propose an end-to-end sequence-to-sequence system, enhanced with an *attention* mechanism [Bahdanau et al., 2014] to perform better content selection, a *copy*

⁵“question generation” in this paper is to generate the question titles for a Stack Overflow post.

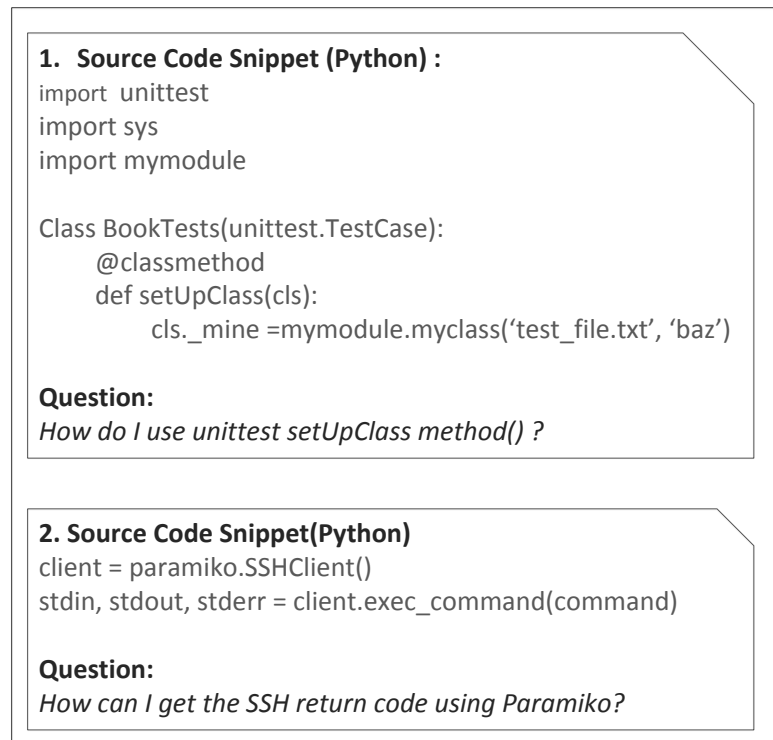


Figure 3.1: Example Code Snippet & Question Pairs

mechanism [Gu et al., 2016a] to handle the rare-words problem, as well as a *coverage* mechanism [Tu et al., 2016] to avoid meaningless repetition. Our system consists of two components: a source-code encoder and a question decoder. Particularly, the code snippet is transformed by a source-code encoder into a vector representation. When it comes to the decoding process, the question decoder reads the code embeddings to generate the target question titles. Moreover, our approach is fully data-driven and does not rely on hand-crafted rules.

To demonstrate the effectiveness of our model, we evaluated it using automatic metrics such as BLEU [Papineni et al., 2002] and ROUGE [Lin, 2004] score, together with a human evaluation for naturalness and relevance of the output. We also performed a practical manual evaluation to measure the effectiveness of our approach for improving the low-quality questions in Stack Overflow. From the automatic evaluation, we found that our approach significantly outperforms a collection of state-of-the-art baselines, includ-

ing the approach based on information retrieval [Robertson and Walker, 1994], a statistical machine translation approach [Koehn et al., 2007], and an existing sequence-to-sequence architecture approach in commit message generation [Jiang et al., 2017]. For human evaluation, questions generated by our system are also rated as more natural and relevant to the code snippet compared with the baselines. The practical manual evaluation shows that our approach can improve the low-quality question titles in terms of Clearness, Fitness and Willingness.

In summary, this work makes the following three main contributions:

- We propose a novel question generation task based on a sequence-to-sequence learning approach, which can help developers to phrase high-quality question titles from given code snippets. Enhanced with the *attention* mechanism, our model can perform the better content selection, with the help of *copy* mechanism and *coverage* mechanism, our model can manage rare word in the input corpus and avoid the meaningless repetitions. To the best of our knowledge, this is the first work which investigates the possibility of improving the low-quality questions in Stack Overflow.
- We performed comprehensive evaluations on Stack Overflow datasets to demonstrate the effectiveness and superiority of our approach. Our system outperforms strong baselines by a large margin and achieves state of the art performance.
- We collected more than 1M $\langle \text{code snippet}, \text{question} \rangle$ pairs from Stack Overflow, which covers a variety of programming languages (e.g., Python, Java, Javascript, C# and SQL). We have released our code⁶ and datasets [Gao, 2020] to facilitate other researchers to repeat our work and verify their ideas. We also implemented a web service tool, named CODE2QUE to facilitate developers and inspire the follow-up work.

⁶<https://github.com/beyondacm/Code2Que>

The rest of this chapter is organized as follows. Section 5.8 presents key related work on question generation and relevant techniques. Section 3.3 presents the motivation of this study. Section 5.3 presents the details of our approach for the question generation task in Stack Overflow. Section 5.4 presents the experimental setup, the baseline methods and the evaluation metrics used in our study. Section 4.4 presents the detailed research questions and the evaluation results under each research question. Section 5.7 presents the contribution of the paper and discusses the strength and weakness of this study. Section 4.6.2 presents threats to validity of our approach. Section 5.9 concludes the paper with possible future work.

3.2 Related Work

Due to the great value of Stack Overflow in helping software developers, there is a growing body of research conducted on Stack Overflow and its data. This section discusses various work in the literature closely related to our work, i.e., deep source code summarization, the empirical study of Stack Overflow on quality assurance, and different tasks by mining the Stack Overflow dataset. It is by no means a complete list of all relevant papers.

3.2.1 Deep Source Code Summarization

A number of previous works have proposed methods for mining the ⟨natural language, code snippet⟩ pairs, these techniques can be applied to tasks such as code summarization as well as commit message generation. (e.g., [Iyer et al., 2016], [Hu et al., 2018], [Jiang et al., 2017], [Wan et al., 2018]).

One similar work with ours is Iyer et al. [Iyer et al., 2016]. They proposed Code-NN, which uses an attentional sequence-to-sequence algorithm to summarize code snippets. This work is similar to our approach because our approach also uses an sequence-to-sequence model. However, there are three key differences between our approach and

Code-NN. First, the goal of Code-NN is summarizing source code snippets while the goal of our approach is generating questions from code snippets. Second, the Code-NN only incorporates attention mechanism while our approach also employs copy mechanism and coverage mechanism, which is more suitable for the specific task of question generation. Third, Code-NN needs to parse the code into AST, while most code snippets in SO are not parsable (e.g., the example code in Fig. 3.8). Followed by Iyer's work, Hu et al. [Hu et al., 2018] proposed to use the neural machine translation model on the code summarization with the assistance of the structural information (i.e., the AST). And Wan et al. [Wan et al., 2018] applied deep reinforcement learning (i.e., tree structure recurrent neural network) to improve the performance of code summarization. Their approach also use AST as the input. All of the aforementioned studies rely on the AST structure of the source code, and note that most of the code in Stack Overflow are not parsable. Thus, the AST-based approaches can not apply to our work.

3.2.2 Question Quality Study on Stack Overflow

The general consensus is that the quality of user-generated content is a key factor to attract users to visit knowledge-sharing websites. Many studies have investigated the content quality in Stack Overflow (e.g., [Nasehi et al., 2012, Yao et al., 2013, Yang et al., 2014a, Ponzanelli et al., 2014, Correa and Sureka, 2013, Correa and Sureka, 2014, Arora et al., 2015, Anderson et al., 2012, Li et al., 2012, Liu et al., 2013, Zhang et al., 2018, Duijn et al., 2015, Trienes and Balog, 2019]).

For example, Nasehi et al. [Nasehi et al., 2012] manually performed a qualitative assessment to investigate the important features of precise code examples in answers of 163 SO posts. Yao et al. [Yao et al., 2013] investigated quality prediction of both Q&As on SO. The output revealed that answer quality is strongly positively associated with that of its question. Yang et al. [Yang et al., 2014a] found that the number of edits on a question

is a very good indicator of question quality. Ponzanelli [Ponzanelli et al., 2014] developed an approach to do automatic categorization of questions based on their quality. Correa et al. [Correa and Sureka, 2013] studied the closed questions in Stack Overflow, finding that the occurrence of code fragments is significant.

All of the above mentioned studies are either predicting quality of the post or increasing the accuracy of predictions. Different from the existing research, our approach is related to improve the quality of the questions. To the best of our knowledge, this is the first work which investigates the possibility of improving the low quality questions using code snippets in Stack Overflow.

3.2.3 Machine/Deep Learning on Software Engineering

Recently, an interesting direction of software engineering is to use machine/deep learning for different tasks to improve software development. Such as code search (e.g., [Gu et al., 2018, Li et al., 2019a, Husain et al., 2019, Allamanis et al., 2015]), clone detection (e.g., [Wang et al., 2020b, Gao et al., 2020, White et al., 2016, Büch and Andrzejak, 2019, Gao et al., 2019b]), program repair (e.g., [White et al., 2019, Mesbah et al., 2019, Vasic et al., 2019, Chen et al., 2019]), document (such as API and questions/answers/tags) recommendation (e.g., [Gu et al., 2016b, Gu et al., 2017, Xia et al., 2013, Wang et al., 2015, Wang et al., 2018, Zhu et al., 2015, Gkotsis et al., 2014, Xu et al., 2017, Singh and Simperl, 2016]).

For code search tasks, Gu et al. [Gu et al., 2018] proposed a deep code search model which uses two deep neural networks to encode source code and natural language description into a vector representation and then uses a cosine similarity function to calculate their similarity. Allamanis et al. [Allamanis et al., 2015] proposed a system that uses Stackoverflow data and web search logs to create models for retrieving C# code snippets given natural language questions and vice versa. For clone detection tasks, white et al. [White et al., 2016] first proposed a deep learning-based clone detection method to

identify code clones via extracting features from program tokens. For program repair tasks, White et al. [White et al., 2019] propose an automatic program repair approach, DeepRepair, which leverages a deep learning model to identify the similarity between code snippets. For document recommendation tasks, Xia et al. [Xia et al., 2013] developed a tool, called TagCombine, an automatic tag recommendation method which analyzes objects in software information sites. Gkotsis et al. [Gkotsis et al., 2014] developed a novel approach to search and suggest the best answers through utilizing textual features. Ganguly et al. [Ganguly and Jones, 2015] examined the retrieval of a set of documents, which are closely associated with a newly posted question. Chen et al. [Chen et al., 2016] studied cross-lingual question retrieval to assist non-native speakers more easily to retrieve relevant questions.

Although the aforementioned studies have utilized machine/deep learning for different software development activities, to our best knowledge, no one has yet considered the question generation task in Stack Overflow. In contrast to all previous work, we propose a novel approach to generate a question by a given code snippet. Our work is first to tackle such a task for helping developers to generate a question when presenting a given code snippet.

3.3 Motivation

In this section, we first summarise the problem and our solution in this study. Following that, we present some example user scenarios of employing our approach in the software development process. We then show some motivating examples from Stack Overflow of the sorts of problems our work addresses.

3.3.1 The Problem and Our Solution

Despite the detailed guidelines provided by the community, a very large number of questions in Stack Overflow are of low-quality [Arora et al., 2015, Correa and Sureka, 2014]. These poorly asked questions are often ambiguous, vague, and/or incomplete, and hardly attract potential experts to provide answers, thus hindering the progress of knowledge generation and sharing. In order to improve question quality, we need to improve title, body and tags. In this work, we focus on improving titles. The motivation for our work is that improving low-quality question titles can potentially be helpful in increasing the likelihood of getting help for the information seekers, as well as reducing the manual effort for quality maintenance of the CQA community. We propose a novel approach to assist developers in posting high-quality questions by generating question titles for a given code snippet. Our approach provides benefit for the following tasks: (i) *Question Improvement*: many developers can not post clear and/or informative questions due to their lack of knowledge and terminology related to the problem, and/or their poor english writing skills. Our approach can generate high-quality question titles for helping developers to summarize the key problems behind their presented code snippet. (ii) *Edits Assistance*: the SO community has employed a collaborative editing mechanism to maintain a satisfactory quality level for the post. However, the editing process may require several interactions between the asker and other community members, thus delaying the answering and even causing questions to sink in the list of open issues. Our approach can be used as an automatic edit assistance tool to improve the question formulation process and reduce the manual effort for quality maintenance. (iii) *Code Embeddings*: Another byproduct of our approach is the code embeddings generated by our approach. In this study, we have collected more than 1M code snippets which covers various programming languages such as Java, Python, Javascript, C#, etc. All the code snippets are embedded into a high-dimensional vector space by our approach. A variety of applications such

as code search (e.g., [Gu et al., 2018, Li et al., 2019a, Husain et al., 2019]) , summarization (e.g., [Iyer et al., 2016, Hu et al., 2018, Jiang et al., 2017, Wan et al., 2018]), retrieval (e.g., [Chen et al., 2016, Allamanis and Sutton, 2013, Xu et al., 2018]), and API recommendation (e.g., [Gu et al., 2016b, Gu et al., 2017]) can benefit from the code embeddings used in our study.

3.3.2 User Scenarios

We implement our model as a standalone web application tool, called CODE2QUE. Developers can copy and paste their code snippet to our tool to generate a question title for the code snippet. Meanwhile, by utilizing the vector representation of the code snippets, CODE2QUE also retrieves a list of top related questions in Stack Overflow and recommends them to the developers. The usage scenarios of our proposed tool are as follows:

Without Tool. Consider Bob who is a developer, who is learning a new development framework. He is also a non-native English speaker with poor English writing skills. Daily, Bob encounters various programming problems during development. He locates the code that is the root cause of the problem, but he cannot figure it out. Due to his lack of the knowledge and terminology of the development framework being used, he does not even know how to most effectively search for answers to the problem on the Internet. Therefore, he creates a question in Stack Overflow, provides his code snippet in the question body according to the Stack Overflow guidelines, and then tries his best to write a question title to summarize the problem. Unfortunately, his question title turns out to be very unclear and uninformative, and there are few users attracted by his question. Bob waits for a long time but does not get any help.

With Tool. Now consider that Bob adopts tool CODE2QUE. Before he searches on the Internet, Bob copies his code snippet to our CODE2QUE tool to generate a question title for the code snippet. Bob uses the generated question as a query to search on the internet. The

searching results are now closely related to the development framework, even though he is not very familiar with it. Bob can also quickly review a list of related questions in Stack Overflow which have a similar problem code snippet. After going through these results, Bob can gain a better understanding of the problem that he is trying to solve and quickly fix the problem by himself. Moreover, Bob can also go back to his earlier poorly asked questions, Bob can use our tool as an edit assistance tool on question titles for reformulating these low-quality questions. Bob provides the code snippet in the question body and writes a question title based on the question title generated by our tool and the knowledge he learned from the results. This time, his question title is much more clear and informative and Bob's question soon attracts an expert of the development framework. With the help of this expert, Bob successfully figures his problem out.

3.3.3 Motivating Examples

A large number of questions have been closed by community members because their question titles are unclear and need further clarifying. For example, the screenshots in Fig. 3.2 and Fig. 3.3 show two examples of problematic Stack Overflow question titles. Developers posted a question "*Fibonacci sequence in Python3.2*" and "*I am creating a notepad in java ... to paste it at location of cursor*" in Stack Overflow. They attached their code snippet and tried to explain the key meaning of their problems. However, such question titles are still very uninformative (in Fig. 3.2) and confusing (in Fig. 3.3). Both of these questions have been marked as having lack of clarity and need to be further improved upon. Such titles run a real risk of not being found by the ideal people to answer them, may make potential question answering users lose interest, or make users who may answer them have to painstakingly browse the additional paragraph to understand the key point. All reduce the likelihood of them giving help.

Using the tool CODE2QUE described in this paper, we can provide a way to automate

Fibonacci sequence in Python3.2 [closed]

Asked 5 years, 9 months ago Active 5 years, 9 months ago Viewed 3k times



-5



1



Closed. This question needs [details or clarity](#). It is not currently accepting answers.

Want to improve this question? Add details and clarify the problem by [editing this post](#).

Closed 5 years ago.

I really need your help. I know this question has been asked countless times already but I still cant find the answer...

I need to programm a fibonacci sequence recursively in a bla-bla.py file, this is what I've got so far:

```
print("Unendlicher Fibonacci-Generator Rekursiv")
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
for n in fib(n):
    print (str(n))
```

Can someone tell me what the correct code is for the recursive function?

[python](#) [fibonacci](#) [nameerror](#)

[share](#)
[improve this question](#)

edited May 19 '14 at 12:10

asked May 19 '14 at 11:55



[milami](#)

37 ● 1 ● 6

Figure 3.2: Example of Problem Questions Title (for Python)

I am creating a notepad in java i have been able to copy the selected text but is enable to paste it at location of cursor [closed]

Ask Question

Asked 5 years, 7 months ago Active 5 years, 7 months ago Viewed 363 times

Closed. This question needs [details or clarity](#). It is not currently accepting answers.

Want to improve this question? Add details and clarify the problem by [editing this post](#).
Closed 5 years ago.

```
private void TextAreaMouseReleased(java.awt.event.MouseEvent evt) {  
    if (TextArea.getSelectedText() != null){  
        String s = TextArea.getSelectedText();  
    }  
}
```

This is the code i am using for copying

share improve this question

edited Jul 31 '14 at 17:13
vandale
3,368 ● 2 ● 16 ● 36

asked Jul 31 '14 at 17:09
Surbhit
1 ● 1

Figure 3.3: Example of Problem Questions Title (for Java)

the process of improving such poor quality question titles, which is potentially helpful in reducing the manual effort for the quality maintenance of CQA forums. Based on the developer’s code snippet, the generated question title by our tool is “*how to find the fibonacci series through recursion?*” for the code snippet shown in Fig. 3.2 and “*how to change the string value in textarea field using java?*” for the code snippet shown in Fig. 3.3. These newly generated question titles are much more clear and informative to readers, and also questioning on the key problems of the user’s concern. This is helpful for the potential helpers to understand the key problems of the question better and also for the askers to formulate a related question better.

3.4 Approach

In this section, we firstly define the task of question generation, then present the details of Stack Overflow question generation system. Fig. 5.3 demonstrates the workflow used by our model. A Long Short Term Memory (LSTM) encoder-decoder architecture, is enhanced by *attention* mechanism [Bahdanau et al., 2014], *copy* mecha-

nism [Gu et al., 2016a] and *coverage* mechanism [Tu et al., 2016]. In general, our model consists of two components: **A Source-code Encoder** and **A Question Decoder**. The source code snippet is transformed by Source-code Encoder into a vector representation, which is then read by a Question Decoder to generate the target question titles. Our model is a differentiable Seq2Seq model with aforementioned three mechanism, i.e., *attention* mechanism, *copy* mechanism and *coverage* mechanism, which can be trained in an end-to-end fashion with gradient descent.

3.4.1 Task Definition

The motivation for our work is to improve the low-quality questions in Stack Overflow. Considering many developers may not be able to describe the problems due to their lack of knowledge and terminology, and/or they are not native english speakers, we propose a novel task in this paper - automatic generation of question titles from a code snippet, the central theme of which is helping developers to create better question titles based on their targets and code snippets. We formulate this task as a sequence-to-sequence learning problem.

Given \mathbf{C} is the sequence of tokens within a code snippet, our target is to generate a Question \mathbf{Q} , which is relevant, natural, syntactically and semantically correct. To be more specific, our main objective is to learn the underlying conditional probability distribution $P_\theta(\mathbf{Q}|\mathbf{C})$ parameterized by θ . In other words, the goal is to train a model θ using $\langle \text{code snippet}, \text{question} \rangle$ pairs such that the probability $P_\theta(\mathbf{Q}|\mathbf{C})$ is maximized over the given training dataset. More formally given a code snippet \mathbf{C} as a sequence of tokens (x_1, x_2, \dots, x_M) of length M , and a question title \mathbf{Q} as a sequence of natural language words (y_1, y_2, \dots, y_N) of length N . Mathematically, our task is defined as finding \bar{y} , such that:

$$\bar{y} = \operatorname{argmax}_{\mathbf{Q}} P_\theta(\mathbf{Q}|\mathbf{C}) \quad (3.1)$$

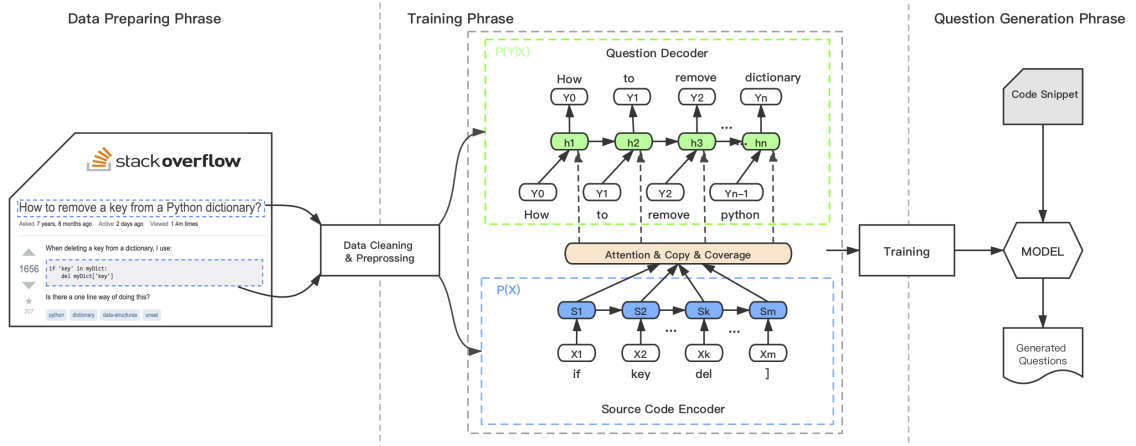


Figure 3.4: Workflow of Code2Que Model

where $P_\theta(\mathbf{Q}|\mathbf{C})$ is defined as:

$$P_\theta(\mathbf{Q}|\mathbf{C}) = \prod_{i=1}^L P_\theta(y_i | y_1, \dots, y_{i-1}; x_1, \dots, x_M) \quad (3.2)$$

$P_\theta(\mathbf{Q}|\mathbf{C})$ can be seen as the conditional log-likelihood of the predicted question title \mathbf{Q} given the input code snippet \mathbf{C} .

3.4.2 Source-code Encoder

Source code token in the code snippet is fed sequentially into the encoder, which generates a sequence of hidden states. Our encoder is a two-layer bidirectional LSTM network,

$$\begin{aligned} \overrightarrow{\mathbf{f}w}_t &= \overrightarrow{\text{LSTM}}_2 \left(x_t, \overrightarrow{\mathbf{h}}_{t-1} \right) \\ \overleftarrow{\mathbf{b}w}_t &= \overleftarrow{\text{LSTM}}_2 \left(x_t, \overleftarrow{\mathbf{h}}_{t-1} \right) \end{aligned}$$

where x_t is the given input source code token at time step t , and $\overrightarrow{\mathbf{h}}_t$ and $\overleftarrow{\mathbf{h}}_t$ are the hidden states at time step t for the forward pass and backward pass respectively. The hidden states (from the forward and backward pass) of the last layer of the source-code encoder are concatenated to form a state s as $s = [\overrightarrow{\mathbf{f}w}_t; \overleftarrow{\mathbf{b}w}_t]$.

3.4.3 Question Decoder

Our question decoder is a single-layer LSTM network, initialized with the state s as $s = [\overrightarrow{\mathbf{f}w}_t; \overleftarrow{\mathbf{b}w}_t]$. Let $qword_t$ be the target word at time stamp t of the ground truth question

title. During training, at each time step t the decoder takes as input the embedding vector y_{t-1} of the previous word $qword_{t-1}$ and the previous state s_{t-1} , and concatenates them to produce the input of the LSTM network. The output of the LSTM network is regarded as the decoder hidden state s_t , as follows:

$$\mathbf{s}_t = \text{LSTM}_1(y_{t-1}, \mathbf{s}_{t-1}) \quad (3.3)$$

The decoder produces one symbol at a time and stops when the END symbol is emitted. The only change with the decoder at testing time is that it uses output from the previous word emitted by the decoder in place of $word_{t-1}$ (since there is no access to a ground truth then).

3.4.4 Incorporating Attention Mechanism

We model the attention [Bahdanau et al., 2014] distribution over words in the source code snippets. We calculate the attention (a_i^t) over the i^{th} code snippet token as :

$$e_i^t = v^t \tanh(W_{eh}h_i + W_{sh}s_t + b_{att}) \quad (3.4)$$

$$a_i^t = \text{softmax}(e_i^t) \quad (3.5)$$

Here, v^t , W_{sh} and b_{att} are model parameters to be learned, and h_i is the concatenation of forward and backward hidden states of source-code encoder. We use this attention a_i^t to generate the context vector c_t^* as the weighted sum of encoder hidden states :

$$\mathbf{c}_t^* = \sum_{i=1, \dots, |\mathbf{x}|} a_i^t \mathbf{h}_i \quad (3.6)$$

We further use the c_t^* vector to obtain a probability distribution over the words in the vocabulary as follows,

$$P = \text{softmax}(\mathbf{W}_v[s_t, c_t^*] + b_v) \quad (3.7)$$

where W_v and b_v are model parameters. Thus during decoding, the probability of a word is $P(qword)$. During the training process for each word at each timestamp, the loss associated with the generated question title is :

$$Loss = -\frac{1}{T} \sum_{t=0}^T \log P(qword_t) \quad (3.8)$$

The *attention* mechanism allows the model to focus on the most relevant parts of the input sequence as needed. For example in Fig. 5.3, at time step 2, the context vector c_t^* amplifies related hidden states h_k with high scores, and drowning out unrelated hidden states with low scores. For such a case, it enables the question decoder to focus on the word “del” when it generates the word “remove”. This ability to amplify the signal from the relevant part of the input sequence makes attention models produce better results than models without attention.

3.4.5 Incorporating Copy Mechanism

A *copy* mechanism [Gu et al., 2016a] is used to facilitate copying some tokens from the source code snippet to the target generated question title. As illustrated in Fig. 3.1, some words such as “setUpClass” are naturally going to be much less frequent than other words. Thus it is highly unlikely for a decoder that is solely based on a language model to generate such a word with very rare occurrences in a corpus. In such cases, the possibly rare words in the input sequence might be required to be *copied* from our source code snippet to the target generated question title. We incorporate a *copy* mechanism to handle such rare word problem for Stack Overflow question generation.

In order to learn to copy (from source) as well as to generate words from the vocabulary (using the decoder), we calculate $p_{cg} \in [0, 1]$. This is the decision of a binary classifier that determines whether to generate a word from the vocabulary or to copy the word directly from the input code snippet, based on attention distribution a_i^t :

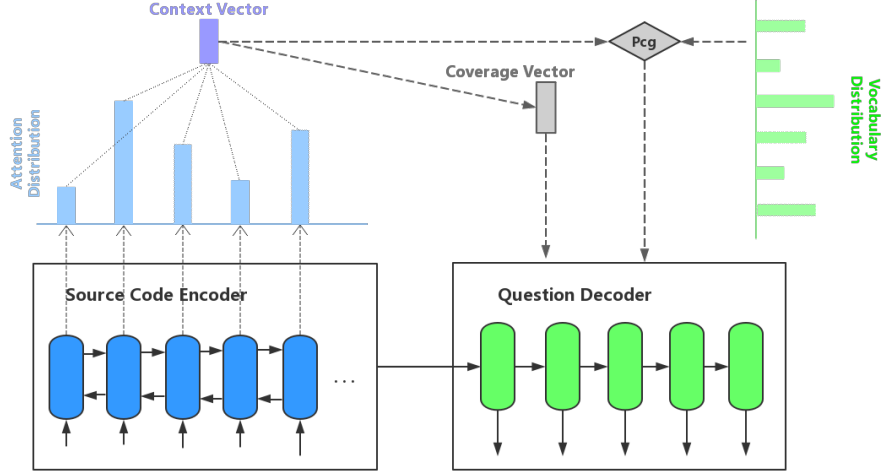


Figure 3.5: Attention & Copy & Coverage Mechanism

$$p_{cg} = \text{sigmoid}(W_{eh}^T c_t^* + W_{sh}^T s_t + W_x x_t + b_{cg}) \quad (3.9)$$

Here W_{eh} , W_{sh} , W_x and b_{cg} are trainable model parameters. The final probability of decoding a word is specified by the mixture model :

$$p^*(qword) = p_{cg} \sum_{i:w_i=qword} a_i^t + (1 - p_{cg})p(qword) \quad (3.10)$$

where $p^*(qword)$ is the final distribution over the union of the vocabulary and the input sequence. As discussed earlier, Equation (10) addresses the rare words issue, since a word not in our vocabulary will have probability $p(qword) = 0$. Therefore, in such cases, our model will replace the $\langle unk \rangle$ token for out-of-vocabulary words with a word in the input sequence having the highest attention obtained using attention distribution a_i^t . The *copy* mechanism allows the model to locate a certain segment of the input sequence and puts that segment into the output sequence. p_{cg} is a soft switch to choose between generating a word from vocabulary or copying a word from the input sequence. For example, in Fig. 3.1, the rare word “setUpClass” in the question title is copied from the input source code snippet. For such a rare word, *copy* mechanism increases the copy-mode probability

and decreases the generate-mode probability, which can correctly catch the rare word and put it to the output sequence.

3.4.6 Incorporating a Coverage Mechanism

Repetition is a common problem for sequence-to-sequence models and to discourage meaningless repetitions, we maintain a word coverage vector cov , which is the sum of attention distributions over all previous decoder timesteps:

$$cov^t = \sum_{t'=0}^{t-1} a^{t'} \quad (3.11)$$

Intuitively, cov^t is a distribution over source code snippet tokens that represents the degree of coverage that those tokens have received from the *attention* mechanism so far. Note that no word is generated before timestamp 0, and hence cov^0 will be a zero vector then. The update equation (4) is now modified to be:

$$e_i^t = v^t \tanh (W_{cv} cov_i^t + W_{eh} h_i + W_{sh} s_t + b_{att}) \quad (3.12)$$

Here, W_{cv} are trainable parameters that ensure the *attention* mechanism's current decision is informed by a reminder of its previous decisions. The *coverage* mechanism allows our model to solve the word repetition problem in the output sequence (see Figure 3.12). The *coverage* mechanism ensures that the *attention* mechanism's current decision is informed by a reminder of its previous decisions (summarized in cov^t). This should make it easier for the *attention* mechanism to avoid repeatedly attending to the same locations, and thus avoid generating repetitive text.

Following the incorporation of the copy and *coverage* mechanism in our attentional sequence-to-sequence architecture, the final loss function will be:

$$Loss = \frac{1}{T} \sum_{t=0}^T \log P^*(qword_t) + \lambda L_{cov} \quad (3.13)$$

where λ is a reweighted hyperparameter and the coverage loss L_{cov} is defined as:

$$L_{cov} = \sum_i \min(a_i^t, cov_i^t) \quad (3.14)$$

Once the model is trained, we do inference using a beam search. The beam search is parametrized by the possible paths number k . The inference process stops when the model generates the END token which stands for the end of the sentence.

3.5 Experimental Setup

In this section, we firstly describe the evaluation corpus of the task. We then introduce the implementation details of our neural generation approach, the baselines to compare, and their experimental settings. Lastly, we explain the evaluation metrics.

3.5.1 Pre-processing

We experiment with our neural question generation model on the latest dump of the Stack Overflow (SO) dataset, which is publicly available⁷. Each post comprises a short question title, a detailed question body, and one or more associated answers and multiple tags.

In this study, we performed our experiment on a variety of programming languages, which include Python, Java, Javascript, C# and SQL. To do that, we used the *Python*, *Java*, *Javascript*, *C#* and *SQL* tag for collecting questions associated with the corresponding programming language respectively. Then we removed all questions whose question scores were less than 1. This is reasonable since our goal is to generate high-quality questions to help developers. We extracted code snippets (using `<code>` tags) within the post’s question body and corresponding post question title. We added the resulting `<question, code snippet>` pairs to our corpus.

Data Preprocessing We tokenized the code snippet with respect to each programming language for pre-processing respectively. We adopted the NLTK toolkit [Bird and Loper, 2004] to separate tokens and symbols. One of the challenging tasks during the tokenization was the structural complexity of the code snippet in our dataset. We stripped out all comments

⁷<https://archive.org/details/stackexchange>

Table 3.1: Dataset Statistics

Languages	#Code Tokens	#Question Tokens	Avg.Code Length	Avg.Question Length
Python	2,367,148	109,329	84.7	11.2
Java	3,371,946	123,994	103.2	10.8
Javascript	2,814,729	121,854	94.1	10.8
C#	2,340,202	100,178	82.1	11.0
SQL	1,483,056	48,668	84.1	10.1

Table 3.2: Number of Training/Validation/Testing Samples

Python	# pairs (Train)	186,976	# pairs (Test-Raw)	3,000
	# pairs (Val)	3,000	# pairs (Test-Clean)	2,940
Java	# pairs (Train)	250,708	# pairs (Test-Raw)	3,000
	# pairs (Val)	3,000	# pairs (Test-Clean)	2,963
Javascript	# pairs (Train)	290,610	# pairs (Test-Raw)	3,000
	# pairs (Val)	3,000	# pairs (Test-Clean)	2,940
C#	# pairs (Train)	178,830	# pairs (Test-Raw)	3,000
	# pairs (Val)	3,000	# pairs (Test-Clean)	2,974
SQL	# pairs (Train)	150,002	# pairs (Test-Raw)	3,000
	# pairs (Val)	3,000	# pairs (Test-Clean)	2,980

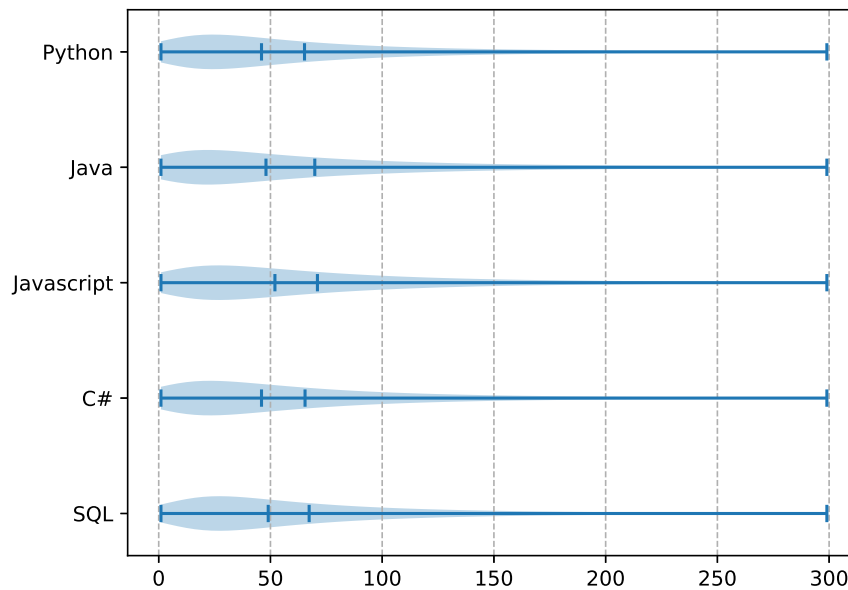


Figure 3.6: Violinplots of Code Distribution

by using the regular expression for different programming languages. After that, in order to avoid being context-specific, numbers and strings within a code snippet and replaced them with special tokens “VAR”, “NUMBER” and “STRING” respectively. Table 3.1, Fig. 3.6 and Fig. 3.7 shows some data statistics on the processed dataset. We can see that the length of Java and Javascript code snippets are much longer than the other programming languages. On average, Java and Javascript code snippets contain 103 and 94 tokens respectively, while the code snippets of the other three programming languages are just around 84 tokens long. On the other hand, the question titles of all the programming languages are approximately at the same level, the overall average of the question titles are 11 tokens long.

Data Filtering Users can post different types of questions in SO, such as “how to X” and “What/Why is Y”. In our preliminary study, we targeted questions which include interrogative keywords such as “how”, “what”, “why”, “which”, “when”. For the above collection of question-code pairs, only the pairs where the aforementioned keywords appear in the question title were kept. After that, we removed pairs where the code snippets are too

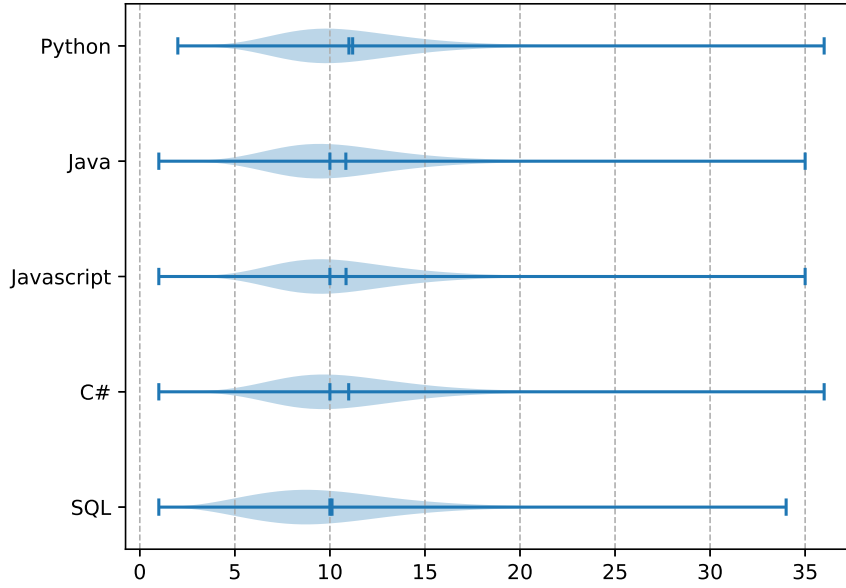


Figure 3.7: Violinplots of Question Distribution

long or too short. Based on the interquartile range (IQR) of the violin plots in Fig. 3.6 and Fig. 3.7, we only preserved pairs where the token range from 16 tokens to 128 tokens for code snippet and the token range from 4 tokens to 16 tokens for question titles. At this stage, we collected more than 1M $\langle question, code snippet \rangle$ pairs in total for Python, Java, Javascript, C# and SQL programming languages. We randomly sampled 3,000 pairs for validation and 3,000 pairs for testing respectively, and kept the rest for training. The details of the training, validation and testing samples for each programming language are summarized in Table 4.1.

Clone Detection Considering that there may be duplicate and/or very similar $\langle code snippet, question \rangle$ pairs between the training set and testing set, this may mislead the evaluation results. We further conducted a primitive clone detection analysis to remove the noisy examples from our testing data set. A lot of methods have been proposed for clone detection in recent years (e.g., [Wang et al., 2020b, Gao et al., 2020, White et al., 2016, Büch and Andrzejak, 2019, Gao et al., 2019b]). We followed the approach proposed by [Gao et al., 2019b] for clone detection. For each code snippet, we compose a numerical

Table 3.3: Clone Detection Analysis

Similarity	Python	Java	Javascript	C#	SQL
$s_i \in [0.0, 0.2)$	2,153	2,241	1,939	2,328	2,359
$s_i \in [0.2, 0.4)$	512	473	651	422	413
$s_i \in [0.4, 0.6)$	195	187	272	182	159
$s_i \in [0.6, 0.8)$	80	62	78	42	49
$s_i \in [0.8, 1.0]$	60	37	60	26	20

vector by summing up the word embedding vectors for all the relevant tokens within the code snippet. Then the similarity between two code snippets C_1 and C_2 can be calculated as follows:

$$Distance(C_1, C_2) = Euclidean(e_1, e_2) \quad (3.15)$$

$$Similarity(C_1, C_2) = 1 - Distance(C_1, C_2) \quad (3.16)$$

where e_1 and e_2 are the corresponding code embedding vectors of C_1 and C_2 . Each code snippet C_i in the testing set is queried against all the code snippets in the training set, the maximum similarity score s_i associated with the C_i is retrieved. The results of s_i with respect to each programming language are summarized in Table 3.3. If the similarity score s_i is over a threshold δ (δ is set to 0.8 in this study), then the code snippet C_i is viewed as a code clone and will be deleted from our testing set. From the table we can see that the number of clone code snippets is very small, while most code snippets get relatively low similarity scores. After removing all the examples with similarity scores above 0.8 from the testing set, we reconstructed a clean testing set for each programming language, the final results are summarized in Table 4.1. The clean testing set is used for the final evaluation of this study.

3.5.2 Implementation Details

We implemented our system in Python using Tensorflow framework. We added special START and END tokens for each sequence in our training set. The vocabulary size for

the Java and Python dataset were set to 50,000 and 80,000 respectively. We use a two-layer bidirectional LSTM for the encoder and a single-layer LSTM for the decoder. We set the number of LSTM hidden states to 256 in both encoder and decoder. We choose the word embeddings of 300 dimensions. Optimization is performed using stochastic gradient descent (SGD) with a learning rate of 0.01. We fix the batch size for updating to be 32. During decoding, we perform beam search with beam size of 10. We train the model for 30 epochs. Our hyper-parameters were tuned on the validation set, the evaluation results were reported on the test set. We discuss the details of the parameter tuning in Section 4.4.

3.5.3 Baselines

To demonstrate the effectiveness of our proposed approach, we compared it with several competitive baseline approaches. We adapted these approaches slightly for our problems, i.e., generating question titles from a given code snippet. We briefly introduced these approaches and the experimental settings as below. For each method mentioned below, the involved parameters were carefully tuned, and the parameters with the best performance were used to report the final comparison results.

1. **IR** stands for the information retrieval baseline. For a given code snippet c_i , it retrieves the *question titles* associated with the code c_j that is closest to the input code c_i from the training set. We use TF-IDF [Robertson and Walker, 1994] metric to calculate the distance between two code snippets, and build a nearest neighbor model to retrieve the most similar instance from the training set.
2. **MOSES** [Koehn et al., 2007] is a widely used phrase-based statistical machine translation system. Here, we treat a tokenized code snippet as the source language text, and the corresponding question title as the target language text. We run the translation from code snippets to question titles. We train a 3-gram language model on target side texts using KenLM [Heafield, 2011], and perform tuning with MERT

on dev set.

3. **NMT** Jiang et al. [Jiang et al., 2017] proposed an sequence-to-sequence approach to generate commit message from code, we refer to it as NMT in our study. We choose NMT as one comparing approach since its promising performance in commit generation. NMT model take source code as inputs and associated question title as outputs. Hyperparameters are tuned with validation set.
4. **CODE-NN** Iyer et al. [Iyer et al., 2016] proposed an attention-based Long Short Term Memory (LSTM) neural network, named CODE-NN, to generate descriptive summaries for C# code snippets and SQL queries. In order to use CODE-NN, the C# code fragments and SQL statements first need to be parsed by the modified version of parser. Considering code snippets in SO are usually incomplete and not parsable, and it is non-trivial to design specific parser to parse code snippets of various programming languages, we tried our best to apply our approach to the CODE-NN dataset, which include 60k+ C# (title, query) pairs and 30k+ SQL (title, query) pairs respectively.

3.5.4 Evaluation Metrics

We evaluate our task with automatic evaluation, and also perform human evaluation via a user study.

1. **Automatic Evaluation** To evaluate different models, We adopt BLEU-1, BLEU-2, BLEU-3, BLEU-4 [Papineni et al., 2002], ROUGE-1, ROUGE-2 and ROUGE-L [Lin, 2004] scores. BLEU is a precision-oriented measure commonly used in translation tasks, which measures the average n -gram precision on a set of reference sentences, with a penalty for overly short sentences. BLEU- n is the BLEU score that uses up to n -grams for counting co-occurrences. ROUGE is a recall-oriented measure widely used in summarization tasks, which used to evaluate n -grams recall of

the summaries with gold-standard sentences as references. ROUGE-1 and ROUGE-2 measures the unigram and bigrams between the system and reference summaries. ROUGE-L is a longest common subsequence measure metric, it does not require consecutive matches but in-sequence matches that reflect sentence level word order. We conducted a large scale automatic evaluation over various kinds of programming languages, i.e., Python, Java, Javascript, C# and SQL. In our work, we regard the generated *question titles* as candidates, and the original human written question titles as gold-standard references.

2. **Human Evaluation** Since automatic evaluation of generated text does not always agree with the actual human-perceived quality and usefulness of the results, we also perform human evaluation studies to measure how humans perceive the generated questions. To do this, we consider two modalities in our user study : *Naturalness* and *Relevance*. *Naturalness* measures the grammatical correctness and fluency of the question title generated. *Relevance* measures how relevant the *question title* is to the code snippet, and indicates the factual divergence of the code snippet to the reference question titles. We randomly sampled 50 $\langle \text{code snippet}, \text{question} \rangle$ pairs from Python and Java test results respectively, for each code snippet, we provided 5 associated *question titles*: one was generated by human (the ground truth *question title*), while the others were generated by baseline methods and our approach. Then we invited 5 evaluators, including 4 Ph.D students and 1 Masters student, all of whom are not co-authors, majoring in Computer Science and have industrial experience with Python as well as Java programming (ranging from 1-3 years). All of the five evaluators have at least one year of studying/working-experience in English speaking countries.

Each participant was asked to manually rate generated question titles on a scale between 1 and 5 (5 for the best results) across the above modalities. The volunteers were blinded as to which question title was generated by our approach.

3. **Practical Manual Evaluation** Following the human evaluation, we also performed a practical manual evaluation to further analyze whether our approach can generate better question titles for *low-quality* questions in Stack Overflow. To do this, we randomly sampled 50 *low-quality* $\langle \text{code snippet}, \text{question} \rangle$ pairs from our Python and Java datasets before the data preprocessing. It is worth mentioning that different from human evaluation, these sampled posts were not included in our training and/or testing set, because all the questions with score less than 1 were removed before training processing. For each code snippet, we applied our approach to generate a question title for manual annotation. We conducted pairwise comparison between two question titles (one was generated by humans, one was generated by our tool) for the same code snippet. For each pairwise comparison, we asked the same 5 evaluators to decide which one is better or non-distinguishable in terms of the following three metrics: *Clearness*, *Fitness*, *Willingness to Respond*. *Clearness* measures whether a question title is expressed in a clear way. Unclear questions are ambiguous, vague, and/or incomplete. *Fitness* measures whether a question title is reasonable in logic with the provided code snippet, and whether it is questioning on the key information. Unfit question titles are either irrelevant to the code snippet or universal questions. *Willingness to Respond* measures whether a user is willing to respond to a specific question. This metric is used to justify how likely the generated questions can elicit further interactions. If people are willing to respond, the interactions can go further. Each metric is evaluated independently on each pairwise comparison. Also the two question titles were randomly shuffled and the participants do not know which question is generated by our approach.

3.6 Results and Analysis

To gain a deeper understanding of the performance of our approach, we conduct analysis on our evaluation results in this section. For quantitative analysis, firstly we study the experimental results of automatic evaluation, then we examine the outcome of human evaluation. Specifically, we mainly focus on the following research questions:

- *RQ-1*: How effective is our approach under automatic evaluation?
- *RQ-2*: How effective is our approach compared with the CODE-NN model?
- *RQ-3*: How effective is our approach under human evaluation?
- *RQ-4*: How effective is our approach for improving low-quality questions?
- *RQ-5*: How effective is our use of *attention* mechanism, *copy* mechanism and *coverage* mechanism under automatic evaluation?
- *RQ-6*: How effective is our approach under different parameter settings?
- *RQ-7*: How efficient is our approach in practical usage?

3.6.1 Automatic Evaluation

The automatic evaluation results of our proposed model and aforementioned baselines are summarized in Table 3.4, 3.5, 3.6, 3.7, 3.8 for Python, Java, Javascript, C#, and SQL respectively. The best performing system for each column is highlighted in boldface. As can be seen, **our model outperforms all the other methods considerably** in terms of BLEU score and ROUGE score. BLEU score measures precision of the system. To be more specific, it measures how many words (and/or n-grams) in the machine generated question titles appear in the ground truth question titles. For ROUGE scores, it measures the recall of the system i.e. how many words(and/or n-grams) in the ground-truth question

Table 3.4: Automatic evaluation(Python dataset)

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L
IR _{TFIDF}	20.2 ± 1.1%	17.7 ± 0.4%	18.4 ± 0.3%	18.0 ± 0.2%	24.4 ± 1.4%	6.9 ± 0.6%	21.8 ± 1.2%
Moses	20.4 ± 1.4%	18.1 ± 0.8%	17.8 ± 0.7%	17.4 ± 0.6%	26.9 ± 1.3%	6.2 ± 0.5%	20.4 ± 1.1%
NMT	28.9 ± 1.7%	21.9 ± 0.7%	21.3 ± 0.3%	20.3 ± 0.2%	34.1 ± 2.2%	10.6 ± 1.1%	31.2 ± 1.9%
Ours	35.8 ± 2.0%	30.1 ± 0.9%	26.8 ± 0.4%	24.2 ± 0.3%	39.9 ± 2.5%	12.6 ± 2.5%	36.7 ± 2.4%

Table 3.5: Automatic evaluation(Java dataset)

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L
IR _{TFIDF}	18.1 ± 1.1%	17.2 ± 0.5%	18.0 ± 0.4%	17.6 ± 0.3%	22.2 ± 1.3%	6.2 ± 0.7%	19.9 ± 1.2%
Moses	18.5 ± 1.0%	17.3 ± 0.6%	17.1 ± 0.5%	16.7 ± 0.4%	25.2 ± 1.5%	5.3 ± 0.4%	20.6 ± 1.2%
NMT	25.0 ± 1.6%	20.7 ± 0.7%	20.9 ± 0.3%	20.2 ± 0.2%	30.0 ± 2.0%	9.6 ± 1.1%	27.3 ± 1.8%
Ours	31.8 ± 1.8%	27.5 ± 0.7%	25.2 ± 0.3%	23.3 ± 0.2%	35.4 ± 2.2%	10.0 ± 1.8%	32.6 ± 2.1%

titles appear in the machine generated questions titles. From the table, we can observe the following points:

1. In general, encoder-decoder architecture baselines, i.e., NMT and our proposed methods, outperform both the IR based approach and the statistical machine translation approach (e.g., Moses) by a large margin. For IR based approach, it retrieves questions from existing database according to similarity score, which relies heavily on whether similar code snippets can be found and how similar the code snippets are. As a result, it is unable to consider the context of the code snippet, which is reflecting that memorizing the training set is not enough for this task. For the phrase-based statistical approaches which use separately engineered subcomponents, the encoder-decoder model uses the vector representation for words and internal states, semantic and structural information can be learned from these vectors by taking global context into consideration.
2. Regarding the BLEU score, our approach is significantly better than the other methods (e.g., traditional IR method, phrase-based statistical method, and NMT meth-

Table 3.6: Automatic evaluation(Javascript dataset)

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L
IR _{TFIDF}	18.7 ± 1.1%	17.6 ± 0.4%	18.3 ± 0.3%	17.9 ± 0.2%	22.6 ± 1.3%	6.2 ± 0.6%	20.2 ± 1.1%
Moses	18.9 ± 1.2%	18.8 ± 0.7%	18.7 ± 0.7%	18.3 ± 0.6%	25.7 ± 1.2%	5.8 ± 0.4%	20.1 ± 1.0%
NMT	28.1 ± 1.6%	22.0 ± 0.6%	21.5 ± 0.3%	20.5 ± 0.2%	32.8 ± 1.9%	10.3 ± 1.0%	30.4 ± 1.7%
Ours	33.2 ± 1.9%	26.4 ± 0.8%	24.1 ± 0.4%	22.1 ± 0.3%	37.3 ± 2.2%	11.7 ± 1.8%	34.7 ± 2.1%

Table 3.7: Automatic evaluation(C# dataset)

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L
IR _{TFIDF}	18.0 ± 1.0%	17.1 ± 0.4%	17.9 ± 0.3%	17.6 ± 0.2%	21.9 ± 1.3%	6.3 ± 0.6%	19.9 ± 1.1%
Moses	18.5 ± 1.0%	16.8 ± 0.7%	16.6 ± 0.6%	16.3 ± 0.6%	25.4 ± 1.2%	6.0 ± 0.4%	20.0 ± 1.0%
NMT	24.4 ± 1.7%	19.3 ± 0.7%	19.8 ± 0.2%	19.3 ± 0.2%	29.4 ± 1.6%	9.7 ± 0.8%	27.1 ± 1.4%
Ours	30.9 ± 1.8%	27.7 ± 0.7%	25.3 ± 0.3%	23.4 ± 0.2%	34.8 ± 2.3%	10.2 ± 1.9%	31.8 ± 2.2%

Table 3.8: Automatic evaluation(SQL dataset)

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-L
IR _{TFIDF}	15.6 ± 1.0%	17.6 ± 0.4%	18.4 ± 0.3%	17.9 ± 0.3%	19.3 ± 1.2%	3.7 ± 0.6%	16.4 ± 1.0%
Moses	17.3 ± 0.9%	16.6 ± 0.7%	16.5 ± 0.6%	16.2 ± 0.6%	21.4 ± 1.1%	3.4 ± 0.3%	15.0 ± 0.8%
NMT	22.0 ± 1.3%	20.4 ± 0.5%	20.7 ± 0.4%	19.9 ± 0.2%	26.6 ± 1.7%	7.4 ± 1.0%	22.9 ± 1.5%
Ours	26.8 ± 1.6%	23.8 ± 0.6%	22.6 ± 0.3%	21.2 ± 0.2%	30.5 ± 2.0%	8.4 ± 1.3%	26.3 ± 1.9%

ods) and achieves understandable results [Seljan et al., 2012]. For example, it improves over NMT methods on BLEU-4 by **19.2%**⁸ on Python dataset and **15.3%** on Java dataset. We attribute this to the following reasons: firstly, our approach is based on a sequence-to-sequence architecture and hence it is superior to the statistical baselines[Koehn et al., 2007]. Secondly, compared with NMT baseline which is solely based on the sequence-to-sequence approach, besides using the encoder-decoder architecture, our approach also incorporates an *attention* mechanism to perform better content selection, a *copy* mechanism to manage the rare-words problem in source code snippet, as well as a *coverage* mechanism to eliminate meaningless repetitions, which makes it superior to the NMT baselines. According to [Seljan et al., 2012], the bleu-1 score above 0.30 generally reflect understandable results and above 0.50 reflect good and fluent translations, the bleu score of our approach can be considered as acceptable, but there is still a large gap compared with ground truth question titles.

- Regarding the ROUGE score, the advantage of our proposed model is also clear. The potential explanation is that baseline methods, such as Moses, NMT, even with a much larger vocabulary, still has a large number of out of vocabulary words. Our model, augmented with the *copy* mechanism to handle the rare-words problem, beats

⁸The improvement ratio is defined within <https://www.d.umn.edu/~gshute/arch/improvements.xhtml>

these baselines by a large margin. This further justifies that the *copy* mechanism generally helps when dealing with the question generation tasks. It also signals that out of vocabulary tokens within code snippet convey much valuable information when generating question titles.

4. The proposed approach performs best on the Python dataset and worst on the SQL dataset. This is in part because, compared with Python code snippet, SQL code snippets only contain a set of keywords and functions, and thus generating question titles for SQL code snippet is more challenging for solely relying on the compositional structures in the input.

Examples of the Automatic Evaluation. We examine several sample outputs by hand to perform a further qualitative analysis. Fig. 3.8 shows some examples of the question titles generated by human (Golden questions), the baselines (e.g., IR, Moses and NMT) and our approach for the given code snippets in the test set. We have the following interesting observations:

1. We see a large gap between our approach and other baselines. **Our approach generates syntactically and semantically correct and relevant question titles in most cases**, while the outputs of every other model are less meaningful and/or more irrelevant. This is consistent with our previous automatic evaluation results. For the IR method, often the question titles are unable to connect to the code snippet. For example in the third sample, the ground truth question is about “find difference between two values”, while the IR methods retrieved the question of “how to calculate the diff between two dates in django”. The statistical machine translation model, such as Moses, is unable to generate a syntactically correct question title. For example, in the sixth and seventh sample, the question titles generated by Moses are incomplete and meaningless. For the NMT method, although it can generate the question titles

ID	Code Snippet	Questions
1	<pre>import unittest import sys class BookTests(unittest.TestCase): @classmethod def setUpClass(cls): cls._mine = mymodule.myclass('test_file.txt', 'baz')</pre>	<p>Golden : How do I use unittest setUpClass method() ?</p> <p>IR : how to run code when a class is subclassed?</p> <p>Moses : how do i import a class booktests unit test how to mine and python</p> <p>NMT : how do i write a pytest from testsuit ?</p> <p>Ours : how do i use a unittest setup class method() ?</p>
2	<pre>import win32gui ImportError : No module named win32gui</pre>	<p>Golden : How to use the win32gui module with Python ?</p> <p>IR : import error: no module named numpy</p> <p>Moses : python win com guiimport guiwin gui importerror module in windows ?</p> <p>NMT : how to choose window to a python script to window with window ?</p> <p>Ours : how to install win32gui . in windows without</p>
3	<pre>def distance(x, y): if x >= y: result = x - y else: result = y - x return result</pre>	<p>Golden : How do I find the difference between two values without knowing which is larger?</p> <p>IR : How to calculate diff between two dates in django</p> <p>Moses : how do I get return to print to console in this code ?</p> <p>NMT : how to make a python program that is not a list of list ?</p> <p>Ours : How to find the absolute distance of two point in python ?</p>
4	<pre>In[2] : mimetypes. guess_extension('image/jpeg', strict=False) Out[2] : '.jpe'</pre>	<p>Golden : Why the various JPEG Extensions?</p> <p>IR : How to load JPE image file?</p> <p>Moses : how to extension in how to include header in python</p> <p>NMT : how to get the index of a list of a list of list in python ?</p> <p>Ours : how to safely get the file extensions from a file use django ?</p>
5	<pre><?xml version="1.0" encoding="UTF-8"?> <MyDocument xmi:version="2.0"> <Thingamabob name="A" hasDohicky="//@Dohicky.0"> <Dingus/> </Thingamabob> <Dohicky name="B"/> </MyDocument></pre>	<p>Golden : In XML what do you call this: //@Dohicky.0 and how to address it in Java</p> <p>IR : Why should you use XML CDATA blocks?</p> <p>Moses : how to use uniqueconstraint with single table inherit jpa ?</p> <p>NMT : how to get the current time in java ? is not abl to get the ip address and host name ?</p> <p>Ours : how to get the attribute of node and its value in xml use dom in java ?</p>
6	<pre>String path = "/puppy.png" try { BufferedImage image = ImageIO.read(getClass().getResourceAsStream(path)); } catch (Exception ex) { ex.printStackTrace(); }</pre>	<p>Golden : java input == null why ?</p> <p>IR : how do I generate random integers within a specific range in java?</p> <p>Moses : file pixel in java ?</p> <p>NMT : how to read a file from a file in java ? is not abl to do so</p> <p>Ours : how to get the path of an image in java ?</p>
7	<pre>webbrowser.open("STRING") gmail_user = raw_input("Please enter your Gmail username:;")</pre>	<p>Golden : How can I disable webbrowser message in python ?</p> <p>IR : how to throw custom 404 messages in python</p> <p>Moses : how to input</p> <p>NMT : how to open a file from a file use python ?</p> <p>Ours : how to I open the web browser message when python2 ?</p>
8	<pre>def test1(): exec('print "hi from test1"') def test2(): exec('print "hi from test2"') def subfunction(): return True</pre>	<p>Golden : Why doesn't exec work in a function with a subfunction?</p> <p>IR : Why does Python code run faster in a function?</p> <p>Moses : how to test work in a function with a subfunction ? python</p> <p>NMT : in python, why doesn't the alternative of a function with a subfunction ?</p> <p>Ours : how python, what is this, ? this function ? some subfunction ?</p>

Figure 3.8: Examples of output generated by each model

in the right format in some cases, it still fails to replicate the critical tokens (e.g., example1) because of the difficulty brought by the unseen words in the code snippet.

2. **Our approach handles out of vocabulary words well**, and it can generate acceptable question titles for a code snippet with rare words. In contrast, the baseline methods often fail in such cases. For example, in the first sample, in which the focus should be put on “setUpClass” method in the code snippet, Our model successfully captures this rare phrase, while other baselines return non-relevant descriptions. It is quite interesting that our model automatically learns to select informative tokens in the code snippet, which shows the extractive ability of our model. At the same time, our approach often generates words to “connect” those critical tokens, showing its aspect of abstractive ability.
3. **A large number of the question titles generated by our model produce meaningful output for simple code snippets**. Note that in some cases, the generated question titles are not exactly inline with the standard ones, yet still make sense by looking at the meaning of the code snippet. For example, in the second case, the ground truth question title is “How to use win32gui module with Python”, our system generates a question title about “how to install win32gui”. This is reasonable given the source code contains “ImportError” while “import win32gui”. In the third case, our approach generates a question title of “how to find the absolute distance of two point in python”, this is because the code snippet defines a function that returns the distance of two points. For such cases, it is reasonable to generate different question titles that look at the code snippet from different aspects. Our question titles can also be viewed as correct and meaningful by looking at the meanings of the code snippet.
4. Sometimes, **our approach can generate *question titles* that are more clear and informative than the ground truth question titles**, such as samples 4-6. For exam-

ple, in the fourth sample, the ground truth question title is “why the various JPEG extensions?” which is uninformative and unclear to the potential helpers, after using our tool the question title can be rephrased as “how to safely get the file extensions from a file” which is more attractive and informative than the original ones.

5. However, **outputs from our system are not always “correct”**. For example, in the last second sample, the ground truth question title is “How can I disable the web browser message in python”, however, our system output an “opposite” *question title* of “How to I open the web browser message when python2”. This example reveals that in some cases, question titles can be generated incorrectly by only looking at the implementation details of the code snippet. This is because we can not judge the developers’ intent just through the code snippet attached to the question.
6. Also, **outputs from our system are not always “perfect”**. The gap between ground truth question titles and machine generated question titles is still large. For example, in the last sample, The question quality of our model degrades on longer and compositional inputs. This indicates that there is still a large room for our question generation system to improve. It would be interesting to further investigate how to interpret why certain irrelevant words are generated in the question title. For example, in the second and fifth samples, there are some irrelevant words at the end of generated questions. We will address such problems in the future.

Answer to RQ-1: How effective is our approach under automatic evaluation? - we conclude that our approach is effective under automatic evaluation and beats the baselines by a large margin.

3.6.2 Compared with CODE-NN

CODE-NN trained a neural attention model generate summaries of C# and SQL code fragment, they have published their C# and SQL datasets, which include 66,015 (title, query)

pairs for C# and 32,337 pairs for SQL. It is worth emphasizing that CODE-NN removed all the non-parsable code snippets and retained only the parsable code snippets. We retrained our approach on the CODE-NN datasets, the automatic evaluation results of our approach and CODE-NN model are summarized in Table 3.9. Because CODE-NN use the BLEU-4 metric for evaluation, we only report the BLEU-4 score in our table. Apart from that, we also explored the effectiveness of transferring our trained model to the new datasets. We further applied the C# and SQL model already obtained to the CODE-NN datasets. This is reasonable because CODE-NN extracted the code snippet only from the accepted answers containing exactly one code snippet, while our approach extracted the code snippet from the questions, so training dataset of our approach will not contaminate the CODE-NN datasets. In other words, our model does not see any test case in the CODE-NN dataset during the training process. From the table, we can observe the following points:

1. In general, our approach and CODE-NN outperforms the other baselines by a large margin. The results are consistent with our previous evaluation. This further justifies the encoder-decoder architecture approach is helpful to learn the semantic and structural information from the code snippet.
2. The neural models, i.e., CODE-NN and ours, have better performance on C# than SQL. This is probably due to the following reasons: First, generating question titles for SQL code snippets is a more challenging task since the SQL code snippet only has a handful of keywords and functions, and the generation models need to rely on other structural aspects. Second, the size of the SQL training data (32,337 pairs) is much smaller than the size of the C# training data (66,015 pairs), it is more difficult to train a good neural model if there is lack of sufficient training data.
3. By using CODE-NN datasets, our model performs better than CODE-NN. It improves BLEU-4 score by 7.8% on C# dataset and 10.8% on SQL dataset. We attribute

Table 3.9: Automatic evaluation(CODE-NN dataset)

Model	BLEU-4 (C# Dataset)	BLEU-4 (SQL Dataset)
IR	13.7	13.5
Moses	11.6	15.4
CODE-NN	20.5	18.4
Ours	22.1	20.4
Ours (Transfer)	21.3	18.4

this to the *copy* mechanism and *coverage* mechanism incorporated into our approach, which is able to handle the low frequency tokens and reduce the redundancy during the generation process.

4. By transferring existing trained models to the CODE-NN datasets, it is notable that even without training directly on the CODE-NN datasets, we can still achieve comparable results compared with the CODE-NN model. We attribute this to the advantage of our model as well as the larger datasets constructed with our approach. We have collected more than 170K *(code snippet, question)* pairs for C# and more than 150K pairs for SQL. The CODE-NN datasets only include 60k+ C# pairs and 30k+ SQL pairs. This verifies the importance of using big training data for applying deep learning-based methods in software engineering.

Answer to RQ-2: How effective is our approach compared with CODE-NN? - we conclude that our approach is more effective compared with Code-NN.

3.6.3 Human Evaluation

Fig. 3.9 shows one example in our human evaluation study. We obtain 250 groups of scores from human evaluation for Python and Java Dataset respectively. Each group contains 4 pairs of scores, which were rated for candidates produced by IR, Moses, Seq2Seq and our approach. Each pair contains a score for the *Naturalness* modality and a score for *Relevance* modality. We regard a score of 1 and 2 as low-quality, a score of 3 as medium quality, and a score of 4 and 5 as high-quality. Regarding human evaluation study results,

Table 3.10: Human Evaluation(Python dataset)

Model	Naturalness	Low _N	Medium _N	High _N	Relevance	Low _R	Medium _R	High _R
IR	3.91	13.2%	15.6%	71.2%	2.22	66.4%	19.2%	14.4%
Moses	2.44	62.4%	17.2%	20.4%	2.73	40.8%	30.8%	28.4%
NMT	3.38	22.0%	28.4%	49.6%	2.90	35.6%	32.4%	32.0%
Ours	3.75	18.4%	12.8%	68.8%	3.55	18.8%	22.8%	58.4%

Table 3.11: Human Evaluation(Java dataset)

Model	Naturalness	Low _N	Medium _N	High _N	Relevance	Low _R	Medium _R	High _R
IR	3.56	19.6%	22.8%	57.6%	2.29	68.4%	14.4%	17.2%
Moses	2.37	62.4%	18.4%	19.2%	2.24	65.2%	21.6%	13.2%
NMT	2.96	28.0%	45.2%	26.8%	2.66	47.2%	27.6%	25.2%
Ours	3.42	22.0%	27.2%	50.8%	3.25	28.8%	24.4%	26.8%

the responses from all evaluators is then averaged for each modality. We also count the proportion of each quality type within each modality. The quality distribution and average score of Naturalness and Relevance across each methods are presented in Table 3.10 and Table 3.11. From the table, several points stand out:

1. From Naturalness prospective, **IR performs a slightly better than our approach.** This is reasonable since it retrieves other similar question titles which are all also written by humans. However its output lacks the explanation to the actual input code snippet, which also explains its surprisingly low score on Relevance.
2. From Relevance prospective, **the question titles generated by our approach are much more appreciated** by the volunteers. Its superior performance in terms of Relevance further supports our claim that it manages to select content from input more effectively.
3. In general, **our model performs well across both dimensions.** The results of human evaluation are consistent with automatic evaluation results. The considerable proportion of high-quality questions generated by our approach with respect to the *Naturalness* and *Relevance* also reconfirms the effectiveness of our system.

Answer to RQ-3: How effective is our approach under human evaluation? In

<pre> DefaultHttpClient httpClient = new DefaultHttpClient(); CookieStore cookieStore = httpClient.getCookieStore(); BasicClientCookie cookie = new BasicClientCookie("abc", "123"); // Prepare a request object HttpGet httpget = new HttpGet("http://abc.net/restofurl"); cookieStore.addCookie(cookie); httpClient.setCookieStore(cookieStore); // Execute the request HttpResponse response = httpClient.execute(httpget); // Examine the response status log.info("Http request response is: " + response.getStatusLine()); List<Cookie> cookies = cookieStore.getCookies(); for (int i=0; i<cookies.size();i++) { if (cookies.get(i).getName().toString().equals("abc")) { log.info("cookie is: " + cookies.get(0).getValue().toString()); } } </pre>		
Please rate each Candidate for N(Naturalness) and R(Relevance) from 1-5 (5 is the best)		
Reference : Using apache httpclient how to set cookie for http request ?		
Candidate1 : how to connect android app with mysql database through php	N:	R:
Candidate2 : how to use the java httpclient . x how to imit send from us ?	N:	R:
Candidate3 : how to get the url from a http post request ? is not work	N:	R:
Candidate4 : how to get cookie from apache httpclient ?	N:	R:

Figure 3.9: User Study Case (Human Evaluation)

general, for considering the combination of both modality, i.e., *Naturalness* and *Relevance*, our model beats the baselines by a large margin.

3.6.4 Practical Manual Evaluation

Fig. 3.10 shows one example of our practical manual evaluation study. We collected 50 pairs of question titles (one was generated by humans and one was generated by our approach) for Python and Java respectively for comparison purposes. For each pairwise comparison, we got 5 groups of selections from the evaluators. Each group contains three user selections with respect to the *Clearness*, *Fitness* and *Willingness* measures respectively. We calculated the proportion of the user selection according to each evaluation metric. Table 3.12 and Table 3.13 show the results of the practical manual evaluation for Python and Java respectively. From the table we can see that:


```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
pd.read_csv('C:/Python34/libs/kospi.csv')
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xb3 in position 0: invalid start byte

```

Please rate which Candidate is *better* with the following metrics:

Candidate 1: Problems in csv import

Candidate 2: How do I read a csv file correctly in python ?

Clearness: Candidate-1 Candidate-2 Non-distinguishable

Fitness: Candidate-1 Candidate-2 Non-distinguishable

Willingness: Candidate-1 Candidate-2 Non-distinguishable

Figure 3.10: User Study Case (Practical Manual Evaluation)

Table 3.12: Practical Manual Evaluation (Python dataset)

Ours vs. Human	Win (%)	Lose (%)	Non-distinguishable (%)
Clearness	52.4	33.2	14.4
Fitness	55.2	24.0	20.8
Willingness	61.2	31.6	7.2

1. The question titles generated by our approach outperform the poor quality question titles in terms of all the metrics. This demonstrates that our approach produces more clear and/or appropriate question titles, which is potentially helpful for improving the low-quality questions in Stack Overflow.
2. Particularly, our question titles have substantially better willingness scores, indicating that developers are more willing to respond to our questions. This shows that question titles generated by our model are more likely to elicit further interactions, which is helpful to increase the likelihood of receiving answers.

Examples of Practical Manual Evaluation: Fig. 3.11 presents three examples of manual evaluation results. From these cases we can see that:

1. The question titles with poor scores in Stack Overflow are often unclear (e.g., Example1) and/or inappropriate (e.g., Example2). For such cases, the question titles

Table 3.13: Practical Manual Evaluation (Java dataset)

Ours vs. Human	Win (%)	Lose (%)	Non-distinguishable (%)
Clearness	42.8	34.0	23.2
Fitness	47.2	39.6	13.2
Willingness	49.2	26.8	24.0

generated by our approach are more clear and attractive, such as Example1, and also questioning on key information. For example, the newly generated question titles in Example2 are much more appreciated by the evaluators than the original ones, which increases the likelihood and willingness of the developers to offer help.

2. Not all of the poor quality question titles can be improved by our approach. Notably for some posts, our approach suffered from semantic drift, that is the questions generated by our approach do not align well with the developers’ intent. Such as in Example3, the developer’s problem was more about “writing with large data”, while the semantics of our question generated has drifted to the problem of “java with byte-buffer”. This is because the string variable “very large data” has been replaced by STR during data preprocessing, such information loss hinders the learning process of our approach.
3. Even though the results generated by our approach are still not perfect, our approach is the first step on this topic and we also release our code and dataset to inspire further follow-up work.

Answer to RQ-4: How effective is our approach for improving low-quality questions? In general, for a large number of low-quality questions in Stack Overflow, our approach can improve the quality of the question titles via *Clearness*, *Fitness* and *Willingness* measures.

Example1 (6795345) — Question Score: -3	Example2 (4099140) — Question Score: -3	Example3 (876602) — QuestionScore: -3
<pre>import urllib2, urllib from BeautifulSoup import BeautifulSoup import re import urlparse ... raw = urllib.urlopen(url) soup = BeautifulSoup(raw) parse = list(urlparse.urlparse(url)) for ender in soup.findAll(ender): links = "%(src)s"% ender if ".jpg" in links: end = ".jpg" if ".jpeg" in links: end = ".jpeg" if ".gif" in links: end = ".gif" if ".png" in links: end = ".png" i += 1 urllib.urlretrieve(links, "%s%s" % (i, end))</pre>	<pre>import org.xsocket.connection.*; import java.io.IOException; public class SocketClient { public static void main(String[] args) { try { IBlockingConnection bc = new BlockingConnection("127.0.0.1", 8090); String req = "Hello server"; bc.write(req + "\n\n"); } catch (IOException e) { System.out.println("missing"); } } } C:\Users\Wildfire\Desktop>java -cp xSocket-2.8.14.jar SocketClient.java Exception in thread "main" java.lang.NoClassDefFoundError: SocketClient/java</pre>	<pre>message = "very large data"+"n"; ByteBuffer buf = ByteBuffer.wrap(message.getBytes()); int nbytes = channel.write(buf);</pre>
Human: Need help with a Python scraper	Human: Can compile but not run the code	Human: problem with writing large data using java nio socket channel
Ours: how to extract all links from url using beautiful soup?	Ours: java - how do i handle the noclassdeffounderror ?	Ours: how to write nbytes in java with bytebuffer ?
Ours vs. Human: Clearness(5:0) Fitness(4:1) Willingness(5:0)	Ours vs. Human: Clearness(4:1) Fitness(4:1) Willingness(5:0)	Ours vs. Human: Clearness(2:2) Fitness(1:4) Willingness(3:2)

Figure 3.11: Practical Manual Evaluation Example

3.6.5 Ablation Analysis

We added an *attention* mechanism, a *copy* mechanism and a *coverage* mechanism to our sequence-to-sequence architecture. The ablation analysis is to verify the effectiveness of the three mechanisms, to be more specific, we compare our approach with several of its incomplete variants:

- **Model_{Atten+Copy}** removes the *coverage* mechanism from our approach.
- **Model_{Atten+Coverage}** removes the *copy* mechanism from our approach.
- **Model_{Atten}** removes the *copy* and *coverage* mechanism from our approach.
- **Model_{Basic}** removes all the *attention*, *copy* and *coverage* mechanism from our approach.

The ablation analysis results are presented in the Table 3.14 and Table 3.15. We can observe the following points:

1. By comparing the results of **Model_{Basic}** and **Model_{Atten}**, it is clear that incorporating an *attention* mechanism is able to improve the overall performance. For example,

by adding *attention* mechanism, the average BLEU-4 score of the Attention-based model was improved by 9% and 13.3%, ROUGE-L score was improved by 6.8% and 10.8% on Python and Java dataset respectively. We attribute this to the ability of *attention* mechanism to perform better content selection, which can focus on the more salient part of the source code snippet.

2. By comparing **Model_{Atten}** with **Model_{Atten+Copy}** and **Model_{Atten+Coverage}**, we can measure the performance improvements achieved due to the incorporation of *copy* mechanism and *coverage* mechanism respectively. Better performance can be achieved by solely adding *copy* or *coverage* mechanism to the attention-based model. This signals that both *copy* and *coverage* mechanism do have contributions to the performance improvements.
3. Without *copy* mechanism, there is a drop overall in every evaluation measure, the ROUGE-L score drops 13% and 9.4% on Python and Java dataset respectively. On the other hand, without *coverage* mechanism, we see a consistent but sufficiently lower drop in each evaluation measure, the ROUGE-L drops 12.3% on Python and 3.8% on Java.
4. By comparing the results of our approach with each of the variant model, we can see that no matter which type of mechanism we dropped, it does hurt the performance of our model. This verifies the importance and effectiveness of these three mechanisms.

Examples of Ablation Analysis To gain further insight into our approach, we further illustrate some examples from the ablation analysis to show the effect of employing the *attention*, *copy* and *coverage* mechanism. The results are shown in Fig. 3.12, we can see that:

1. Question titles generated by the basic model are of low-quality. Comparing the results of the basic model and attention model, we can see that by adding the *attention*

<p>Example1: Code Snippet (Python)</p> <pre>def get_client_ip(request): x_forwarded_for = request.META.get('HTTP_X_FORWARDED_FOR') if x_forwarded_for: ip = x_forwarded_for.split(',')[0] else: ip = request.META.get('REMOTE_ADDR') return ip</pre>	<p>Human: How to get user public ip in django?</p> <p>Basic (Drop 3): how to check if a variable exists in a django template ?</p> <p>Attention: how do i get the url from a post post request ?</p> <p>Attention + Coverage (Drop Copy): Where in my django app do i get ip address ?</p> <p>Attention + Copy (Drop Coverage): Where in my django app do i implement this get_client_ip () function function?</p> <p>Ours: How to use get_client_ip () function find ip address in django app?</p>
<p>Example2: Code Snippet (Java)</p> <pre>public static List<Product> getCartList() { List<Product> cartList = new Vector<Product>(cartMap.keySet().size()); for(Product p : cartMap.keySet()) { cartList.add(p); } Gson gson = new Gson(); // convert your list to json String jsonCartList = gson.toJson(cartList); System.out.println("jsonCartList: " + jsonCartList); return jsonCartList; }</pre>	<p>Human: How to convert list data into json in java</p> <p>Basic (Drop 3): how to sort a list of strings in java ?</p> <p>Attention: how to convert a list of objects to a hashmap a hashmap?</p> <p>Attention + Coverage (Drop Copy): how to convert a list of objects to a hashmap ?</p> <p>Attention + Copy (Drop Coverage): Why does gson.toJson () return null ?</p> <p>Ours: how to convert List < Product > to json in java?</p>

Figure 3.12: Ablation Analysis Example

mechanism, the generated question titles are more meaningful and relevant for the given code snippet. The *attention* mechanism enables the model to focus on the relevant parts of the input sequence as needed. As shown in Example1, the model will focus on the “request” related segment in source code when it generates “post request” for the question title.

2. Repetition is a common problem for attentional sequence to sequence models (e.g., [Tu et al., 2016, Sankaran et al., 2016, Suzuki and Nagata, 2016]). Meaningless repeated words are produced during the generation process (highlighted with yellow color). We introduce a *coverage* mechanism for discouraging such repetitions in our generator by quantitatively emphasizing the coverage of sentence words while decoding. As can be seen in Example2, “a harshmap” has been meaningless repeated twice, employing the *coverage* mechanism can effectively discourage such repetitions.
3. We observe that a *high-quality* question title is generated using our approach. Recall

that a code snippet usually contains tokens (highlighted with a blue color) with very rare occurrences. It is difficult for a decoder to generate such words solely based on language modeling. For such cases, we incorporate the *copy* mechanism to copy the rare tokens from the code snippet to the question title. In the first example, the method name *get_client_ip* has been properly picked up from the source code snippet to the generated question titles.

Answer to RQ-5: How effective is our use of *attention* mechanism, *copy* mechanism and *coverage* mechanism under automatic evaluation? In summary, all the three mechanisms, i.e., *attention* mechanism, *copy* mechanism, *coverage* mechanism, are effective and helpful to enhance the performance of our approach.

Table 3.14: Ablation evaluation (Python dataset)

Measure	Model _{Basic}	Model _{Atten}	Model _{Atten+Coverage}	Model _{Atten+Copy}	Ours
BLEU-1	25.1 ± 1.5%	28.6 ± 1.7%	29.6 ± 1.8%	31.5 ± 1.9%	35.8 ± 2.0%
BLEU-2	20.2 ± 0.7%	22.3 ± 0.8%	24.6 ± 0.6%	27.8 ± 0.8%	30.1 ± 0.9%
BLEU-3	19.1 ± 0.4%	21.7 ± 0.4%	23.8 ± 0.5%	25.4 ± 0.4%	26.8 ± 0.4%
BLEU-4	18.7 ± 0.3%	20.3 ± 0.3%	22.3 ± 0.2%	23.1 ± 0.2%	24.2 ± 0.3%
ROUGE-1	32.8 ± 2.0%	34.1 ± 2.3%	35.3 ± 2.2%	35.4 ± 2.4%	39.9 ± 2.5%
ROUGE-2	9.1 ± 0.8%	10.2 ± 1.2%	10.6 ± 2.1%	10.8 ± 2.0%	12.6 ± 2.5%
ROUGE-L	29.2 ± 5.8%	31.2 ± 2.0%	31.9 ± 2.1%	32.2 ± 2.2%	36.7 ± 2.4%

Table 3.15: Ablation evaluation (Java dataset)

Measure	Model _{Basic}	Model _{Atten}	Model _{Atten+Coverage}	Model _{Atten+Copy}	Ours
BLEU-1	20.5 ± 1.0%	25.2 ± 1.6%	27.8 ± 1.6%	29.7 ± 1.7%	31.8 ± 1.8%
BLEU-2	16.4 ± 0.6%	20.7 ± 0.7%	25.0 ± 0.6%	26.1 ± 0.6%	27.5 ± 0.7%
BLEU-3	17.8 ± 0.4%	21.1 ± 0.3%	23.6 ± 0.3%	24.4 ± 0.3%	25.2 ± 0.3%
BLEU-4	18.1 ± 0.2%	20.5 ± 0.2%	22.0 ± 0.1%	22.6 ± 0.2%	23.3 ± 0.2%
ROUGE-1	28.3 ± 1.3%	30.5 ± 2.0%	31.2 ± 2.0%	33.2 ± 2.1%	35.4 ± 2.2%
ROUGE-2	6.9 ± 0.5%	7.9 ± 1.1%	8.2 ± 1.2%	8.7 ± 1.5%	10.0 ± 1.8%
ROUGE-L	24.6 ± 1.1%	27.3 ± 1.8%	28.8 ± 1.9%	30.6 ± 2.0%	31.8 ± 2.2%

3.6.6 Parameters Settings

One of the key parameter of our approach is the vocabulary size. The encoder-decoder architecture models need a fixed vocabulary for the source input and target output. To

Table 3.16: Vocab Size & Training Time(per epoch)

	Threshold	Vocab Size	Training Time
Python	1	58,536	766.9
	2	49,656	719.1
	3	36,277	663.7
	5	22,244	593.8
	7	16,368	549.2
	10	12,142	539.1
	100	2,503	499.9
	Java	Threshold	Vocab Size
1		221,160	2218.3
2		131,862	1692.3
3		79,048	1074.1
5		54,352	962.2
7		38,670	898.4
10		27,341	831.4
100		4,642	723.8

generate all the possible words, the basic Seq2Seq model has to include all the vocabulary tokens that appeared in the training set, which requires a lot of time and memory to train the models. One advantage of our model is that, with the help of *copy* mechanism, our approach can copy words from source input to the target output. We can maintain a small size vocabulary which exclude the low frequency words, but also get better performance and generalization ability.

We set different word frequency threshold, i.e., 1, 2, 3, 5, 7, 10, 100, for constructing the vocabulary. Setting word frequency threshold to 1 means the vocabulary is constructed with words that appeared at least twice in the training set. Different models were trained under these parameters on the Python and Java datasets separately. The vocabulary size and training time under different threshold are summarised in Table 3.16. Fig. 3.13 and Fig. 3.14 shows the influence of different threshold settings on the BLEU-4 score and ROUGE-L score. We have the following observations from these figures:

1. Our approach achieves its best performance on Java dataset when the similarity

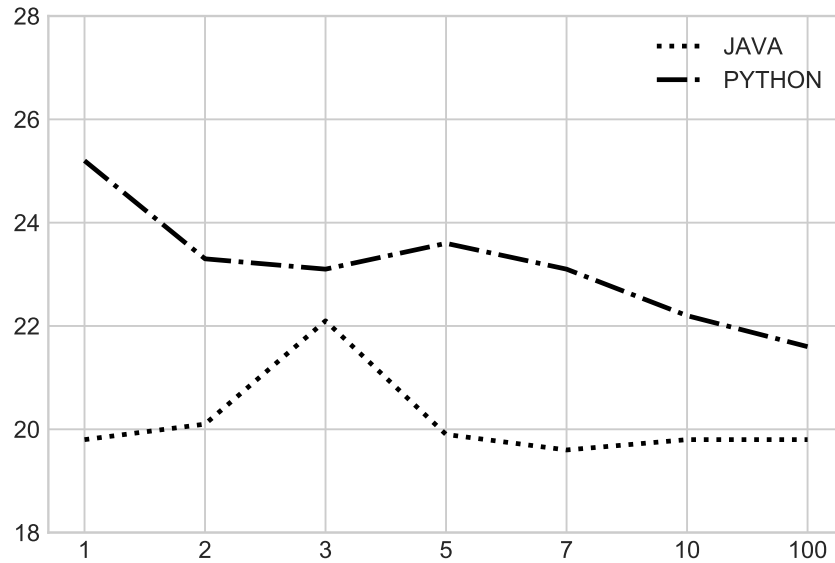


Figure 3.13: BLEU4 Score under different vocab threshold

threshold set to 3, the corresponding vocabulary size is 79,048. When the vocabulary size is too big, i.e., 221,160 with threshold equals 1, the BLEU4 and ROUGE-L score becomes lower. This is because some non-generic words will be included in the fixed vocabulary, which leads to difficulties for our approach to learn how to copy words from the input source sequence.

2. The results of our approach are best on Python dataset when the word frequency threshold set to 1, the corresponding vocabulary size is 58,536. Compared with the results of the Java dataset, the optimum vocabulary size settings of our approach can be around 60000.
3. When the word frequency threshold rockets up to 100, the vocabulary size decreases to 2,503 and 4,626 on Python and Java dataset respectively. Even with a much smaller vocabulary size, our approach can still have a comparable performance against Basic Seq2Seq model, which further supports the generalization ability of our approach.

Another parameter of our approach is the dimension of word embeddings. We choose five different word embedding sizes, i.e., 100, 200, 300, 400, 500, and qualitatively com-

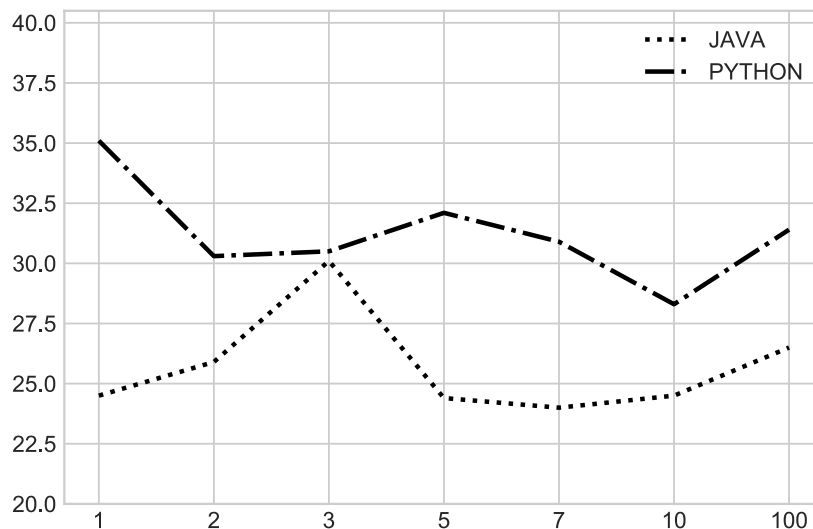


Figure 3.14: ROUGE-L Score under different vocab threshold

pare the performance of our approach in these different word embeddings. Fig. 3.15 and Fig. 3.16 show the influence of different word embedding sizes on the BLEU-4 and ROUGE-L score. One can clearly see that our approach achieves the best BLEU-4 and ROUGE-L score when the embedding size is set to 300. Too large word embedding size may not be helpful to improve the accuracy.

3.6.7 Efficient Analysis

The experiment was conducted on an Nvidia GeForce GTX 1080 GPU with 8GB memory. The time cost of our approach is mostly for the training process which takes approximately 8 to 10 hours for different datasets. The testing process on around 3,000 examples takes one to three minutes, while generating a single question title only costs 20 to 60ms.

Considering that the query for generating a question title using our approach is efficient, we have implemented our approach as a standalone web-based tool, named CODE2QUE, to facilitate developers in using our approach and to inspire follow up research. Fig. 3.17 shows the web interface of CODE2QUE. Developers can copy and paste their code snippet

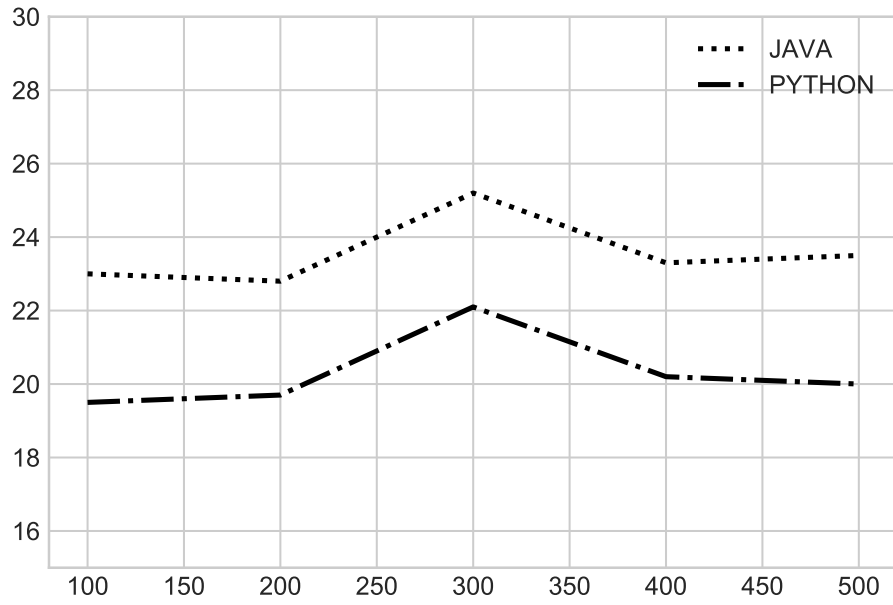


Figure 3.15: BLEU4 Score under different sizes of word embeddings

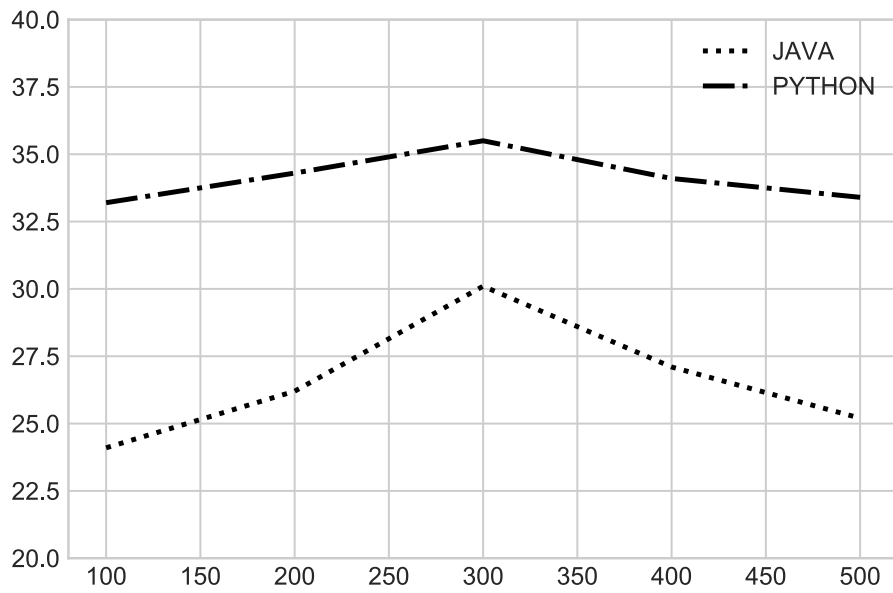


Figure 3.16: ROUGE-L Score under different sizes of word embeddings

into our web application. CODE2QUE embeds the code snippet via source code encoder and generates the question titles for the developers. We below describe the details of the input and output of such a process.

- **Input:** the input to the CODE2QUE is a code snippet, which is an ordered sequence of source code lines. We have provided support for different types of programming languages (e.g., Python, Java, Javascript, C# and SQL) for users to select. The input box in Fig. 3.17 shows an example of a Python code snippet. After inputting the code snippet, the developers can click the “Generate” button to submit their query.
- **Output:** the output of CODE2QUE is in two parts: (i) Generated Questions: CODE2QUE will generate a question title using our backend model according to the code snippet and programming language they choose. For example, “*how to extract text from html pages using html2text*” is generated for the given code snippet. (ii) Retrieved Questions: After the developer submits his/her code snippet to the server, the code snippet is converted into a vector by our backend Source Code Encoder, then CODE2QUE searches through our codebase and returns the top3 questions with similar code snippets. The link to these questions on the Stack Overflow website is also provided for reference. Developers can use these to quickly browse the related questions to have a better understanding of the problem.

Answer to RQ-7: How efficient is our approach in practical usage? In summary, our approach is efficient enough for practical use and we have implemented a web service tool, named CODE2QUE, to apply our approach for practical use.

3.7 Discussion

In this section, we discuss the main contribution of our work and analyze the strength and potential weakness of our work associated with each contribution.



Figure 3.17: CODE2QUE Web Service Tool

3.7.1 Question Quality Improvement

It is important for CQA forums to maintain a satisfactory quality level for the questions and answers so as to improve community reputation and provide better user experience. Questions are a fundamental aspect of a CQA website. Poorly formulated questions are less likely to receive useful responses, thus hindering the overall knowledge generation and sharing process.

- *Strength of our work.* Previous work related to CQA quality studies focus on question quality prediction. For example, the authors in [Ravi et al., 2014] developed a model for predicting question quality using the content of the question. The authors in [Arora et al., 2015] proposed a method to identify inappropriate questions by using previously asked similar questions. Different from the existing research, our study aims to improve low-quality questions in Stack Overflow. To the best of our knowledge, this is the first work that investigates the possibility of automatically improving

low-quality questions in Stack Overflow.

- *Weakness of our work.* According to our practical manual evaluation results, our approach can improve a large number of low-quality questions in Stack Overflow via *Clearness*, *Fitness* and *Willingness* measures. However, the results generated by our approach are still not perfect, and for some posts, our approach suffers from semantic drift problems. We plan to incorporate more context information for generating better question titles in the future.

3.7.2 Deep Sequence to Sequence Approach

Recently, deep learning has achieved promising results in solving many software engineering tasks, such as code search (e.g., [Gu et al., 2018, Li et al., 2019a, Husain et al., 2019]), code summarization (e.g., [Iyer et al., 2016, Hu et al., 2018, Jiang et al., 2017, Wan et al., 2018]), and API recommendation (e.g., [Gu et al., 2016b, Gu et al., 2017]). Among these works, a number of researchers have applied the sequence to sequence methods for mining the \langle natural language, code snippet \rangle pairs, such as the commit message generation. (e.g., [Iyer et al., 2016, Jiang et al., 2017]).

- *Strength of our work.* A major challenge for question generation tasks in our study is the semantic gap between the code snippet and natural language descriptions. To bridge the gap between code fragment and natural language queries, we employed a deep sequence to sequence approach to build the neural language model for both code snippets and natural language questions. The neural language model automatically learns common patterns from the large scale source code snippets. Furthermore, different from the existing sequence to sequence learning approach, we add *attention*, *copy* and *coverage* mechanism to our sequence-to-sequence architecture to suit our specific task. The *attention* mechanism can perform better content selection from the

input, while the *copy* mechanism can handle the rare word problems among the code snippet, and the *coverage* mechanism can eliminate the meaningless repetitions.

- *Weakness of our work.* Previous works [Iyer et al., 2016, Hu et al., 2018, Wan et al., 2018] have shown that incorporating structural information of the source code (i.e., the AST) can improve the performance of the model, However, considering that the majority of the code snippets are not parsable in Stack Overflow, we do not use the AST structural information at the current stage. We plan to use the program repair algorithm to fix the code snippet in Stack Overflow and employ more contextual information of the source code in the future.

3.7.3 Question Generation Task

Stack Overflow is a collaborative question answering website, its target audience are software developers, maintenance professionals and programmers. Over the recent years, Stack Overflow has attracted increasing attention from the software engineering research community. However, since the questions and answers posted by developers on Stack Overflow are usually unstructured natural language texts containing code snippets, which makes it more challenging for researchers to mine and analyze these posts.

- *Strength of our work.* To improve the software development process, researchers have investigated the Stack Overflow knowledge-base for various software development activities, such as predicting the post quality [Ravi et al., 2014, Yao et al., 2013, Yang et al., 2014a, Arora et al., 2015, Ponzanelli et al., 2014], answer recommendation [Gkotsis et al., 2014, Xu et al., 2017, Singh and Simperl, 2016], code/questions retrieval [Xu et al., 2018, Chen et al., 2016, Allamanis et al., 2015, Ganguly and Jones, 2015, Henβ et al., 2012] etc. However, to the best of our knowledge, this is the first work which investigates the question generation task in Stack Overflow. We first perform

such a task to assist developers to generate a question title when presenting a code snippet.

- *Weakness of our work.* We collected more than 1M $\langle \text{code snippet}, \text{question} \rangle$ pairs from Stack Overflow, which covers a variety of programming languages (e.g., Python, Java, Javascript, C# and SQL). Considering our study is the first step on this topic, we have published our data to inspire further follow-up work. However, even though we have cleaned the data via pre-processing, some data may still be noisy. We plan to improve the dataset quality by further manual checking in the future.

3.8 Threats to Validity

We have identified the following threats to validity among our study:

Internal Validity Threats to internal validity are concerned with potential errors in our code implementation and study settings. For the automatic evaluation, in order to reduce errors, we have double-checked and fully tested our source code. We have carefully tuned the parameters of the baseline approaches and used them in their highest performing settings for comparison, but there may still exist errors that we did not note. Considering such cases, we have published our source code and dataset to facilitate other researchers to replicate and extend our work.

External Validity The external validity relates to the quality and generalizability of our dataset. Our dataset is constructed from the official Stack Overflow data dump which contains a variety of programming languages, such as Python, Java, Javascript, C# and SQL. However, there are still many other programming languages in Stack Overflow which are not considered in our study. We believe that our results will generalize to other programming languages, due to the overall reasonable similarity in code snippets despite particular

language syntax, semantics and APIs. We will try to extend our approach to other programming languages to benefit more users in future studies.

Construct Validity The construct validity concerns the relation between theory and observation. In this study, such threats are mainly due to the suitability of our evaluation measures. For the practical manual evaluation, the manual validation could be affected by the subjectiveness of the evaluators and the human errors. For the human evaluation, the evaluators' degree of carefulness, effort and English skills in the examination process may affect the validity of judgements. We minimized such threats by choosing experienced participants who have at least one year of studying/working experience in English speaking countries, and are familiar with Python and Java programming languages. We also gave the participants enough time to complete the evaluation tasks.

Conclusion Validity The conclusion validity relates to issues that could affect the ability to draw correct conclusions about relations between the treatment and the outcome of an experiment. One issue during the data filtering procedure is that we only keep the questions which contain several keywords, such as “how”, “what”, “why”. However, since the questions in Stack Overflow can be rather complicated, our results do not shed light on how effective our solution is on questions of other kinds. On the other hand, from the human evaluation analysis, we see a key challenge for our current work is that the questions generated by our approach suffered from semantic drift. This is because it is difficult to judge a question poster's intent by solely looking at his/her code snippet. In such a case, more relevant information such as question description, question tags could further be incorporated within our model, which may help to generate a question that is more accurate and precise.

3.9 Summary

In this work, we have proposed a model for the task of automatic question generation based on a given code snippet. Our model is based on sequence-to-sequence architecture, and enhanced with an *attention* mechanism to perform better content selection, a *copy* mechanism to handle the rare-words problem within the input code snippet as well as *coverage* mechanism to discourage the meaningless repetitions. We carried out comprehensive evaluation on Stack Overflow datasets to demonstrate the effectiveness of our approach, compared with several existing baselines, our model achieves the best performance in both the automatic evaluation and human evaluation. We have also released our code and datasets to facilitate other researchers to verify their ideas and inspire the follow up work. For future work, we plan to design better models to generate meaningful question titles by considering extra context information, such as question description. Additional work will be needed to address this context-sensitive question generation task.

Chapter 4

DeepAns: Best Answer Recommendation via Question Boosting

Gao, Z., Xia, X., Lo, D., Grundy, J.C., Technical Q&A Site Answer Recommendation via Question Boosting, ACM Transactions on Software Engineering and Methodology, vol. 30, no. 1, December 2020, ACM. <https://doi.org/10.1145/3412845>.

4.1 Introduction

The past decade has witnessed significant social and technical value of Question and Answer (Q&A) platforms, such as Yahoo! Answers¹, Quora², and StackExchange³. These Q&A websites have become one of the most important user-generated-content (UGC) portals. For example, on the Stack Exchange forums, more than 17 million questions have been asked so far, and more than 11 million pages of these forums are visited daily by users. To keep up with the fast-paced software development process, the technical Q&A platforms have been heavily used by software developers as a popular way to seek information and support via the internet.

StackExchange is a network of online question and answer websites, where each website focuses on a specific topic, such as academia, Ubuntu operating system, latex, etc. There are a lot of technical Q&A sites which are heavily used by developers, such as Stack

¹<https://answers.yahoo.com/>

²<https://www.quora.com/>

³<https://stackexchange.com>

Overflow (with a focus on programming-related questions), Ask Ubuntu (with a focus on Ubuntu operating system), Super User (with a focus on computer software and hardware), and Server Fault (with a focus on servers and networks). These Q&A websites allow users to post questions/answers and search for relevant questions and answers. Moreover, if a post is not clear/informative, users routinely provide useful comments to improve the post. Fig. 4.1 shows an example of an initial post and its associated question comment in Ask Ubuntu Q&A site. By providing the question comment to the original post, it can assist potential helpers to write high quality answers since the question is more informative.

In spite of their success and active user participation, the phenomenon of being "*answer hungry*" is still one of the biggest issues within these Q&A platforms. This concept means that a very large number of questions posted remain *unanswered and/or unresolved*. According to our empirical study in different technical Q&A sites, Ask Ubuntu⁴ and Super User⁵, and Stack Overflow⁶. we found that (1) developers often have to wait a long time, spanning from days to even many weeks, before getting the first answer to their questions. Moreover, around 20% of the questions in Ask Ubuntu and Super User do not receive any answer at all and leave the askers unsatisfied; and (2) even with provided answers, about 44% questions in Ask Ubuntu and 39% questions in Super User are still unresolved, i.e., the question asker does not mark any answer as the accepted solution to their post. In such a case, information seekers have to painstakingly go through the provided answers of various quality with no certainty that a valid answer has been provided.

In this work, we aim to address this answer hungry phenomenon by recommending the *most relevant* answer or *the best* answer for an unanswered or unresolved question by searching from historical QA pairs. We refer to this problem as *relevant answer recommendation*. We propose a deep learning based approach we name DEEPANS, which consists of

⁴<https://askubuntu.com/>

⁵<https://superuser.com/>

⁶<https://stackoverflow.com/>

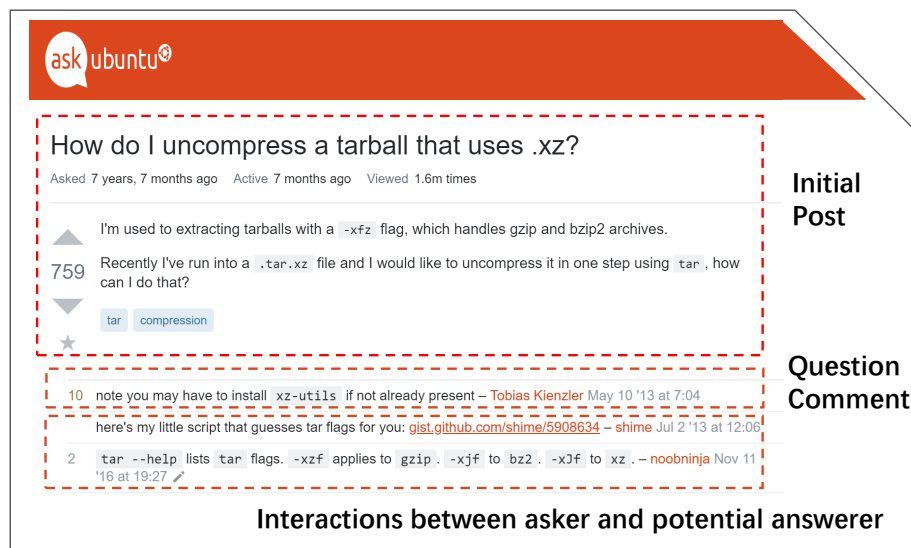


Figure 4.1: Example Post on Ask Ubuntu

three stages: *question boosting*, *label establishment* and *answer recommendation*. Given a post, our first step is to generate useful clarifying questions via a trained sequence-to-sequence model. The clarifying question is then appended to the original post as a way of question boosting, which can help eliminate the isolation between question and answers. Then, in the label establishment phase, for each enriched question, we pair it with its corresponding answers and automatically label the QA pair as *positive*, *neutral⁺*, *neutral⁻* and *negative* samples by leveraging four heuristic rules. In the answer recommendation phase, given a question q and an answer candidate a_i , our goal is calculating the matching degree of the $\langle q, a_i \rangle$ pair. We formulate this problem as a four-category classification problem (i.e., a question and answer pair can be *positive*, *neutral⁺*, *neutral⁻*, or *negative* related). We propose a weakly supervised neural network that can be trained with the aforementioned four kinds of training samples.

The key usage scenarios of DEEPANS are as follows: (1) for unresolved questions which do not have an asker-accepted answer, developers can use DEEPANS to recommend the best answers; and (2) for unanswered questions, developers can use DEEPANS to get the most relevant answers by mining answers to other related questions.

To evaluate the performance of our proposed approach, we conducted comprehensive experiments with four datasets, collected from the technical Q&A sites Ask Ubuntu, Super User and Stack Overflow respectively. The large-scale automatic evaluation results suggest that our model outperforms a collection of state-of-the-art baselines by a large margin. For human evaluation, we asked 5 domain experts for their feedback on our generated clarifying questions and answers. Our user study results further demonstrate the effectiveness and superiority of our approach in solving unanswered/unresolved questions. In summary, this paper makes the following contributions:

- Previous studies neglect the value of interactions between the question asker and the potential helper. We argue that a clarifying question between the question and answers is an important aspect of judging the relevance and usefulness of the QA pair. Therefore, we train a sequence-to-sequence model to generate useful clarifying questions for a given post, which can fill the lexical gap between the questions and answers. To the best of our knowledge, this is the first successful application of generating clarifying questions for technical Q&A sites.
- We present a novel method to constructing *positive*, *neutral*⁺, *neutral*⁻, *negative* training samples via four heuristic rules, which can greatly save the time consuming and labor intensive labeling process.
- We develop a weakly supervised neural network model for the answer recommendation task. For any question answer pairs, we fit the QA pair into our model to calculate the matching score between them; the higher matching score is estimated by our model, the better chance the answer will be selected as the best answer. In particular, the Q&A sites can employ our approach as a preliminary step towards marking the potential solution for the unanswered/unresolved question. This can avoid unnecessary time spent by developers to browse questions without an accepted

solution.

- Both our quantitative evaluation and user study show that DEEPANS can help developers find relevant technical question answers more accurately, compared with state-of-the-art baselines. We have released the source code of DEEPANS and the dataset⁷ of our study to help other researchers replicate and extend our study.

The rest of the paper is organized as follows. Section 5.3 presents the details of our approach to identifying the most relevant answers. Section 5.4 presents the experimental set up and evaluation metrics. Section 4.4 presents the results of our approach on automatic evaluation. Section 4.5 presents the results of our approach on human evaluation. Section 4.6 discusses the strength of our approach and the threats to validity in our study. Section 5.8 presents key related work and techniques of this work. Section 5.9 concludes the paper with possible future work.

4.2 Approach

We present our approach named DEEPANS, which ranks candidate answers from a relevant answer pool and recommends the most relevant answer to developers. In general, our model follows a three-stage process: *Question Boosting*, *Label Establishment*, and *Answer Recommendation*. Particularly, in the question boosting phase, DEEPANS uses an attentional sequence-to-sequence recurrent neural network [Sutskever et al., 2014] to generate possible clarifying questions for a given post. These generated questions are appended to the original post as a way of *question boosting*. Then DEEPANS automatically constructs *positive*, *neutral*⁺, *neutral*⁻ and *negative* training samples for each question and answer pair via four heuristic rules. In the answer recommendation phase, DEEPANS trains another convolutional neural network to calculate the matching score between a given question and

⁷<https://github.com/beyondacm/DeepAns>

a candidate answer, the higher a similarity score is estimated, the more probable the answer will be selected as the best answer.

The underlying principle of applying the recurrent networks for the question boosting task is that compared with CNN neural networks, RNN architectures are dedicated sequence models, and this family of architectures has gained tremendous popularity to prominent applications, e.g., machine translation [Bahdanau et al., 2014, Sutskever et al., 2014]. For the answer recommendation task, we select the convolutional networks. Theoretically, we could also employ the recurrent networks for answer recommendation. However, due to the fact that computing score for each answer in the answer candidate pool is time-consuming, CNN architecture has better performance, lower perplexity, and more importantly, it runs much faster [Dauphin et al., 2017, Kim, 2014] than RNN architecture for text encoding tasks, i.e., we can process all time steps in parallel via convolutional networks in both training and testing processes.

4.2.1 Question Boosting

The task of question boosting is to automatically generate clarifying questions from the title of an initial post. This can be formulated as a sequence-to-sequence learning problem. Given \mathbf{Q} is a sequence of words within the question title of an initial post, our target is to generate a useful clarifying question \mathbf{CQ} , which is relevant, syntactically and semantically correct. To be more specific, the goal is to train a model θ using $\langle q, cq \rangle$ pairs such that the probability $P_\theta(\mathbf{CQ}|\mathbf{Q})$ is maximized over the given training dataset. Mathematically, this query boosting task is defined as finding \bar{y} , such that:

$$\bar{y} = \operatorname{argmax}_{\mathbf{CQ}} P_\theta(\mathbf{CQ}|\mathbf{Q}) \quad (4.1)$$

$P_\theta(\mathbf{CQ}|\mathbf{Q})$ can be seen as the conditional log-likelihood of the clarification question \mathbf{CQ} given the input post \mathbf{Q} . The encoder-decoder architecture has been used in addressing such a problem. We demonstrate an example of the question boosting process in Fig 4.2.

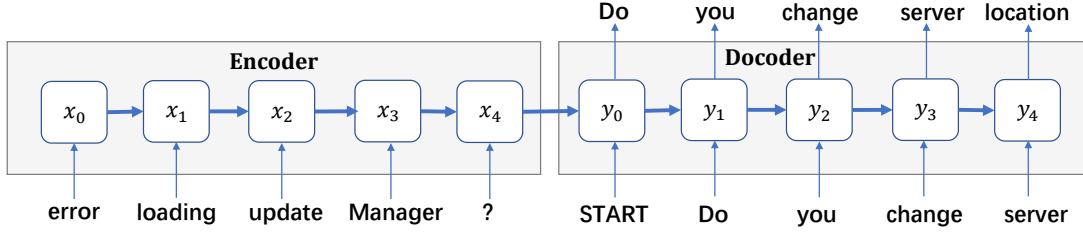


Figure 4.2: Question boosting process

The original post title “error loading update manager ?” is fed into the encoder, and the clarifying question “do you change server location ?” is the decoder target output.

Encoder The sequence of words within a post title is fed sequentially into the encoder, which generates a sequence of hidden states. Our encoder is a two-layer bidirectional LSTM network,

$$\begin{aligned}\overrightarrow{fw}_t &= \overrightarrow{\text{LSTM}}_2(x_t, \overrightarrow{h}_{t-1}) \\ \overleftarrow{bw}_t &= \overleftarrow{\text{LSTM}}_2(x_t, \overleftarrow{h}_{t-1})\end{aligned}$$

where x_t is the given input word token at time step t , and \overrightarrow{h}_t and \overleftarrow{h}_t are the hidden states at time step t for the forward pass and backward pass respectively. The hidden states (from the forward and backward pass) of the last layer of the encoder are concatenated to form a state s as $s = [\overrightarrow{fw}_t; \overleftarrow{bw}_t]$.

Decoder Decoder is single layer LSTM network, initialized with the state s as $s = [\overrightarrow{fw}_t; \overleftarrow{bw}_t]$. Let $qword_t$ be the target word at time stamp t of the clarifying question. During training, at each time step t the decoder takes as input the embedding vector y_{t-1} of the previous word $qword_{t-1}$ and the previous state s_{t-1} , and concatenates them to produce the input of the LSTM network. The output of the LSTM network is regarded as the decoder hidden state s_t , as follows:

$$s_t = \text{LSTM}_1(y_{t-1}, s_{t-1}) \quad (4.2)$$

The decoder produces one symbol at a time and stops when the *END* symbol is emitted. The only change with the decoder at testing time is that it uses output from the previous

word emitted by the decoder in place of $word_{t-1}$ (since there is no access to a ground truth then).

Attention Mechanism To effectively align the target words with the source words, we model the attention [Bahdanau et al., 2014] distribution over words in the target sequence. We calculate the attention (a_i^t) over the i^{th} input token as :

$$e_i^t = v^t \tanh (W_{eh} h_i + W_{sh} s_t + b_{att}) \quad (4.3)$$

$$a_i^t = \text{softmax} (e_i^t) \quad (4.4)$$

Here v^t , W_{sh} and b_{att} are model parameters to be learned, and h_i is the concatenation of forward and backward hidden states of source-code encoder. We use this attention a_i^t to generate the context vector c_t^* as the weighted sum of encoder hidden states :

$$c_t^* = \sum_{i=1, \dots, |x|} a_i^t h_i \quad (4.5)$$

We further use the c_t^* vector to obtain a probability distribution over the words in the vocabulary as follows,

$$P = \text{softmax} (\mathbf{W}_v [s_t, c_t^*] + b_v) \quad (4.6)$$

where W_v and b_v are model parameters. Thus during decoding, the probability of a word is $P(qword)$. During the training process for each word at each timestamp, the loss associated with the generated question is :

$$Loss = -\frac{1}{T} \sum_{t=0}^T \log P(qword_t) \quad (4.7)$$

Once the model is trained, we do inference using beam search [Graves, 2012] and append the generated clarifying question to the original post title. The beam search is pa-

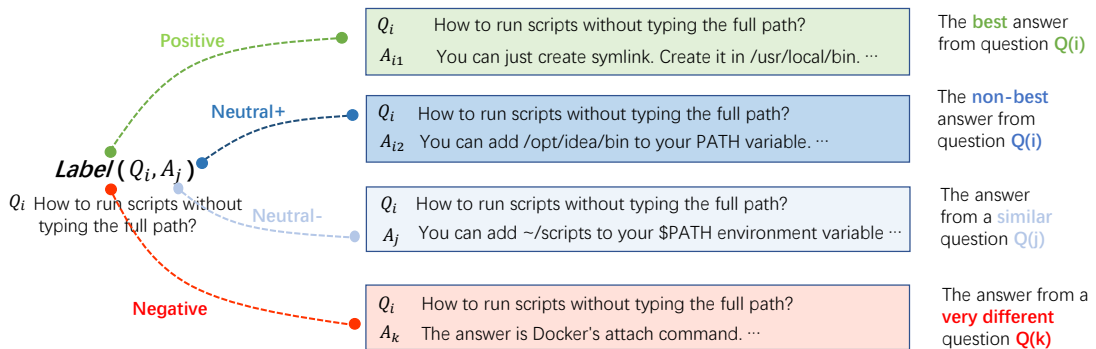


Figure 4.3: Label Establishing Process

parameterized by the possible paths number k . The inference process stops when the model generates the END token, which stands for the end of the sentence.

4.2.2 Label Establishment

According to our empirical study results from Section 2, the *answer hungry* phenomenon widely exists in technical Q&A forums, i.e. only a small proportion of questions have an “resolved” answer, while many others remain unanswered and/or unresolved. Due to the reason of professionalism of technical questions, only the experts with specific knowledge are qualified to evaluate the matching degree between a question and an answer. Therefore it is very hard to find such annotators and/or the creation of training sets requires a substantial manual effort. To address such a problem, We propose a novel scheme to automatically labeling each QA pair as *positive*, *neutral⁺*, *neutral⁻*, and *negative* samples. Fig 4.3 shows an example of our labeling process. We propose four heuristic rules to label the QA pairs:

- *Positive samples*: for a given question Q_i , we pair it with its marked "best" answer (if it has one) A_{i1} , and label this qa pair as *Positive*.
- *Neutral⁺ samples*: for a given question Q_i , we pair it with its non-best answer (answers within the same question thread, except the one marked as the best answer), and label this qa pair as *Neutral⁺*.

- *Neutral⁻ samples*: for a given question Q_i , we randomly select one answer A_j from questions similar to it, then label this question-answer pair as *Neutral⁻*.
- *Negative samples*: for a given question Q_i , we pair it with a randomly selected answer A_k from non-similar questions and label this QA pair as *Negative*.

Since we are recommending answers from candidate answers of questions relevant to the query question, if the retrieved questions are not relevant to the query question, it is unlikely we can select the best answer from the answer candidates pool. We followed the question retrieval method proposed by Xu et al. [Xu et al., 2017] to search for similar questions, which has been proven to be more effective for this task of relevant question retrieval. We used the IDF-weighted word embedding to calculate the similarity score between the query and the question title. Thereafter, a set of similar questions can be identified by selecting the top-k ranked questions.

After this label establishing process, we can gather large amounts of labeled examples, which greatly saves the time-consuming and labor-intensive labeling process.

4.2.3 Answer Recommendation

After collecting large amounts of labeled training data via label establishment, we are able to train the deep learning model based on the four kinds of training samples.

We present a weakly supervised neural network architecture for ranking QA pairs. Fig. 5.3 demonstrates the workflow of our proposed model. The main building blocks of our architecture are two convolutional neural networks [Kim, 2014, Kalchbrenner et al., 2014]. These two underlying sub-models work in parallel, mapping questions and answers to their distributional vectors respectively, which are then used to calculate the final similarity score between them.

Sentence Matrix The input to our model are $\langle q \oplus cq, a \rangle$ pairs, where q and a stands for the question and answer of a labelled QA pair, cq stands for the clarifying questions gener-

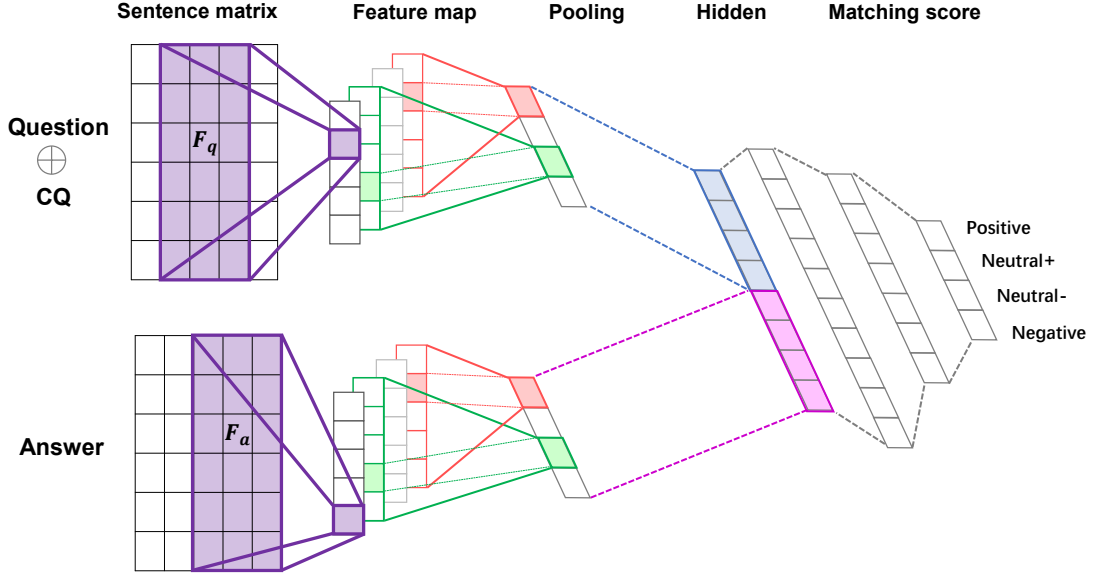


Figure 4.4: Overall architecture of the answer recommendation model.

ated by our question boosting model. The questions (including the original questions and clarifying questions) and answers are parallel sentences, where each sentence is treated as a sequence of words: (w_1, \dots, w_s) , where each word is drawn from a vocabulary \mathbf{V} . Words are represented by distributional vectors $\mathbf{w} \in \mathbb{R}^{1 \times d}$ via looking up in a pre-trained word embedding matrix $\mathbf{W} \in \mathbb{R}^{d \times |\mathbf{V}|}$.

For each input $\langle q \oplus cq, a \rangle$ pair, we build two sentences matrix \mathbf{S}_q and $\mathbf{S}_a \in \mathbb{R}^{d \times |s|}$ for each question and answer respectively, where the i th column represents the word embedding of w_i at corresponding position i in a sentence.

Convolutional feature maps To learn to capture and compose features of individual words in a given sentence from low-level word embeddings into higher level semantic concepts, we apply two identical convolutional neural network blocks to the input sentence matrix \mathbf{S}_q and \mathbf{S}_a respectively.

More formally, the convolution operation $*$ between an input sentence matrix $\mathbf{S}_{q/a} \in \mathbb{R}^{d \times |s|}$ and a filter $\mathbf{F} \in \mathbb{R}^{d \times m}$ (called a filter of size m) results in a vector $\mathbf{c} \in \mathbb{R}^{|s|-m+1}$,

where each component is computed as follows:

$$\mathbf{c}_i = (\mathbf{S} * \mathbf{F})_i = \sum_{k,j} (\mathbf{S}_{[:,i-m+1:i]} \otimes \mathbf{F})_{kj} \quad (4.8)$$

In the above equation, \otimes is the element-wise multiplication and $\mathbf{S}_{[:,i-m+1:i]}$ is a matrix slice of size m along the columns. Note that the convolution filter is of the same dimensionality d as the input sentence matrix. As shown in Fig. 5.3, it slides along the column dimension of \mathbf{S} producing a vector $\mathbf{c} \in \mathbb{R}^{|\mathbf{s}|-m+1}$. Each component \mathbf{c}_i is the result of computing an element-wise product between a column slice of \mathbf{S} and the filter matrix \mathbf{F} , which is then flattened and summed producing a single value. By applying a set of filters (called a filter bank) $\mathbf{F} \in \mathbb{R}^{n \times d \times m}$ to sequentially convolved with the sentence matrix \mathbf{S} will generate a convolutional feature map matrix $\mathbf{C} \in \mathbb{R}^{n \times (|\mathbf{s}|-m+1)}$.

Pooling layer Following that, we pass the output from the convolutional layer to the pooling layer, whose goal is to aggregate the information and reduce the representation. We apply a max pooling operation [Collobert et al., 2011] over the convolutional feature map and take the maximum value $\hat{\mathbf{c}} = \max\{\mathbf{c}_i\}$ as the feature corresponding to a particular filter. The idea is to capture the most important feature - one with the highest value - for each feature map.

Matching score layer The output of the penultimate convolutional and pooling layers x is passed to a series of fully connected layer followed by a softmax layer. It computes the probability distribution over the four kinds of labels (*positive*, *neutral⁺*, *neutral⁻*, *negative*):

$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{x}^T \theta_j}}{\sum_{k=1}^K e^{\mathbf{x}^T \theta_k}} \quad (4.9)$$

where θ_k is a weight vector of the k -th class. \mathbf{x} can be thought of as a final abstract representation of the input QA pair obtained by a series of transformations from the input layer through a series of convolutional and pooling operations.

For the final matching score, we want this score to be high if the input qa pair is *pos-*

itive and *neutral*⁺, and to be low if it is *negative* and *neutral*⁻. Therefore we define the calculation of the similarity score as follows:

$$Score = \omega_{pos} \times P(pos) + \omega_{neu+} \times P(neu^+) - \omega_{neu-} \times P(neu^-) - \omega_{neg} \times P(neg) \quad (4.10)$$

Algorithm 1 DeepAns Algorithm (Offline Training)

Input: Data dump of technical Q&A sites

Output: 1.Question Boosting model; 2.Answer recommendation model

```

1 Extract  $\langle q, cq \rangle$  pairs from data dump Train  $\langle q, cq \rangle$  pairs with attentional-based seq2seq
  model Save the model as Question Boosting model Extract  $\langle q, a \rangle$  pairs from data dump
  for  $q_i, a_i \in \langle q, a \rangle$  pairs do
2   if  $q_i$  has accepted-answer then
3     if  $a_i$  is accepted-answer then
4       Label  $\langle q_i, a_i \rangle$  as Positive
5     end
6   else
7     Label  $\langle q_i, a_i \rangle$  as Neutral+
8   end
9   Select similar answer  $a_j$  then Label  $\langle q_i, a_j \rangle$  as Neutral- Select random answer
     $a_k$  then Label  $\langle q_i, a_k \rangle$  as Negative
10  end
11 end
12 for  $q_i \in$  labelled  $\langle q, a \rangle$  pairs do
13   Generate  $cq_i$  for  $q_i$  using Question Boosting model Append  $cq_i$  to  $q_i$  to make labelled
     $\langle q_i \oplus cq_i, a_i \rangle$  pair
14 end
15 Train labelled  $\langle q \oplus cq, a \rangle$  pairs with CNN-based classification model Save the model as
    Answer Recommendation model

```

Algorithm 2 DeepAns Algorithm (Online Recommendation)

Input: User search query q_{user}

Output: A set of candidate answers with a matching score for each answer

```
16 Generate  $cq$  for  $q_{user}$  using Question Boosting model Search top-k similar questions for
    the given query  $q_{user}$  Add top-k questions to similar question set  $SQ$  for  $q_i \in SQ$  do
17   | for  $a_j \in q_i$  do
18   |   | Add answer to candidate answers set  $CA$ 
19   | end
20 end
21 for  $a_i \in CA$  do
22   | Pair  $a_i$  with expanded query to make a  $\langle q_{user} \oplus cq, a_i \rangle$  pair Fit  $\langle q_{user} \oplus cq, a_i \rangle$  pair
    to Answer Recommendation model Compute the final matching score  $s_i$  via Equa-
    tion 4.10
23 end
24 Rerank answers in  $CA$  via matching scores
```

There are four weights as shown in Equation 4.10. We initially set all the four weights to 1 at the beginning. Then the optimal settings of these weights are carefully tuned on our validation set (detailed in Section 4.4.3). We use the final matching score to measure the relevance between a question and an answer.

4.2.4 DeepAns Algorithm

We divide our model into two components: offline training and online recommendation. The detailed algorithms of *DeepAns* for offline training and online recommendation are presented in Algorithm 1 and Algorithm 2 respectively. To be more specific, during the offline training, we use the data from technical Q&A sites to train the question boosting model (lines 1-3) and answer recommendation model (lines 4-20). When it comes to the online recommendation, for a given user query, we first collect a pool of answer candidates via finding its similar questions (lines 1-8). After that, we use the trained question boosting model to perform query expansion, then pair it with each of the answer candidates and fit them into the trained answer recommendation model to estimate their matching scores (lines 9-14).

4.3 Experiment Setup

In this section, we first describe the data sets used throughout our experiments. We then discuss the baselines we compare to our new *DeepAns* approach and our experimental settings. Lastly, we explain the automatic evaluation process.

4.3.1 Data Preparation

We collected data from the official dump of StackExchange, a network of online question and answer websites. The StackExchange data dump contains timestamped information about the posts, comments as well as the revision history made to the post. Each post comprises a short question title, a detailed question body, corresponding answers and multiple tags. For each post, users can add clarifying questions to posts for further discussion. After receiving one or more answers, the asker can select one answer that is most suitable for their question as the accepted/best answer. We choose three different technical Q&A sites, i.e., Ask Ubuntu, Super User and Stack Overflow for our experiment. These three technical Q&A sites are commonly used by software developers and each one focuses on a specific area. For instance, Ask Ubuntu and Super User focus on Ubuntu system questions and computer software/hardware questions respectively, and Stack Overflow is the most popular programming related Q&A site which has been heavily used by software developers via the internet. As with our previous empirical study, we only focus on the Python and Java related questions in Stack Overflow for this study, referred to as SO (Python) and SO (Java) respectively in this study.

The experimental dataset creation process is divided into three phases: extracting $\langle q, cq \rangle$ pairs, constructing labelled $\langle q, a \rangle$ pairs, and constructing labelled $\langle q \oplus cq, a \rangle$ pairs, where q stands for the question, cq stands for the clarifying question, and a stands for the answer. Table 4.1 describes the statistics of our collected datasets.

1. **Extract $\langle q, cq \rangle$ Pairs:** For each post, we follow the methods described in Section ??

Table 4.1: Number of Training/Validation/Testing Samples

Ask Ubuntu	# $\langle q, cq \rangle$ pairs	68,216	# $\langle q, a \rangle$ pairs	289,062
	# <i>Positive</i> pairs	79,726	# <i>Neutral</i> ⁺ pairs	49,884
	# <i>Neutral</i> ⁻ pairs	79,726	# <i>Negative</i> pairs	79,726
Super User	# $\langle q, cq \rangle$ pairs	87,081	# $\langle q, a \rangle$ pairs	447,221
	# <i>Positive</i> pairs	119,305	# <i>Neutral</i> ⁺ pairs	89,306
	# <i>Neutral</i> ⁻ pairs	119,305	# <i>Negative</i> pairs	119,305
SO (Python)	# $\langle q, cq \rangle$ pairs	311,127	# $\langle q, a \rangle$ pairs	2,372,232
	# <i>Positive</i> pairs	610,948	# <i>Neutral</i> ⁺ pairs	539,388
	# <i>Neutral</i> ⁻ pairs	610,948	# <i>Negative</i> pairs	610,948
SO (Java)	# $\langle q, cq \rangle$ pairs	456,077	# $\langle q, a \rangle$ pairs	3,013,859
	# <i>Positive</i> pairs	734,977	# <i>Neutral</i> ⁺ pairs	808,928
	# <i>Neutral</i> ⁻ pairs	734,977	# <i>Negative</i> pairs	734,977

to extract the clarifying questions. According to our manual analysis results, we summarize a list of keywords associated with non-clarifying questions, such as “edit”, “related”, “vote”, etc. We preprocess our dataset to remove all instances that involve such keywords. We also summarize a list of key phrases associated with the clarifying questions, such as “do you”, “have you”, “how”, “which”, etc. We retained the pairs that include the above key phrases. After that, we pair the original post with its associated clarifying question as $\langle q, cq \rangle$ pairs. We extract a total of 68,216 pairs in Ask Ubuntu, and 87,081 pairs in Super User. The number of $\langle q, cq \rangle$ pairs in Stack Overflow are much larger, we obtain a total of 311,127 pairs for SO (Python) and 456,077 pairs for SO (Java). These collected $\langle q, cq \rangle$ pairs are used to train a sequence-to-sequence model for question boosting.

2. **Construct labelled $\langle q, a \rangle$ Pairs:** To make the $\langle q, a \rangle$ pairs, we first extract the questions that having explicitly marked accepted answers. Then for each question, we pair it with the accept answer to make the *positive sample*, with non-accepted answer to make the *neutral*⁺ *sample*, with an answer to a similar question to make the *neutral*⁻ *sample*, and with an answer to a randomly selected question to make

the *negative sample*. We have to clarify that some questions do not have the non-accepted answers; this is the reason why the number of *neutral*⁺ samples is smaller than the number of other samples, such as Ask Ubuntu, Super User and SO (Python), while some other questions have more than one non-accepted answers, which results in the number of *neutral*⁺ samples is bigger than those of the rest, such as SO (Java). For the final dataset, we construct 289,062 and 447,221 $\langle q, a \rangle$ labelled pairs for Ask Ubuntu and Super User, and 2,372,232 and 3,013,859 $\langle q, a \rangle$ labelled pairs for SO (Python) and SO (Java) respectively. It is obvious that the number of qa pairs in Stack Overflow far outnumber those of other technical Q&A sites. After the label establishment process, we largely expand the labelled dataset for training. We randomly sample 5,000 questions for validation and 5,000 questions for testing respectively, and kept the rest for training. It is worth mentioning that we first used the validation set for model selection regarding the accuracy of QA pairs classification results, which is a middle result of the answer selection target. After that, we reused the validation set for tuning the four weights as shown in Equation 4.10. The testing set was used only for testing the final solution to confirm the actual predictive power of our model with optimal parameter settings.

3. **Construct labelled $\langle q \oplus cq, a \rangle$ Pairs:** For each labelled $\langle q, a \rangle$ pair, we feed the original question to the trained question boosting model to generate a clarifying question. After that, we append the clarifying question to the original question to construct the $\langle q \oplus cq, a \rangle$ pairs. The number of the $\langle q \oplus cq, a \rangle$ pairs is identical with the number of $\langle q, a \rangle$ pairs.

4.3.2 Implementation Details

We implemented our *DeepAns* system in Python using the PyTorch framework. The main parameters of our deep learning model (tuned using the validation dataset) were as follows:

- **Question Boosting:** We train an attentional sequence-to-sequence model for this subtask. Previous studies have shown that the deep sequence-to-sequence model can achieve state-of-the-art performance on different tasks [Sennrich et al., 2016, Iyer et al., 2016, Hu et al., 2018, Gao et al., 2020a]. We also used the parameter settings from [Gao et al., 2020a] for training the $\langle q, cq \rangle$ pairs in this study. We use a two-layer bidirectional LSTM for the encoder and a single-layer LSTM for the decoder. We set the number of LSTM hidden states to be 256 in both encoder and decoder. Optimization is performed using stochastic gradient descent (SGD) with a learning rate of 0.01. During decoding, we perform beam search with a beam size of 10.
- **Answer Recommendation:** Kim et al. [Kim, 2014] have shown that convolutional neural networks trained on top of pre-trained word vectors achieved promising performance for sentence-level classification tasks. Hence in our work, we also followed the experiment settings of their studies. We initialize the word embeddings from our unsupervised corpus and set the dimension of word embedding d to 100. The width m of the three convolution filters is set to 3, 4, 5 and the number of convolution feature maps is set to 100. We use ReLu activation function and a simple max-pooling function. The size of the hidden layer is equal to the size of the join vector obtained after concatenating question and answer vectors from the distributional models.

To train both networks, we used stochastic gradient descent with shuffled mini-batches. The batch size is set to 64. Both network are trained for 50 epochs with early stopping, i.e., we stop the training if no update to the best accuracy on the validation set has been made for the last 5 epochs.

4.3.3 Baselines

To demonstrate the effectiveness of our proposed DEEPANS, we compared it with several comparable systems. We briefly introduced these and how they are used for the task of predicting the best answer among a set of answer candidates. DEEPANS is built with the semantic features of words in their dimensions, we used the average word vector of a sentence as features for training all of the baseline models for a fairer comparison. For each baseline method, their parameters were carefully tuned, and the parameters with the best performance were used to report the final comparison results with our *DeepAns* approach on the same datasets:

- **Learning to Rank** The answer prediction problem of our task is similar to the traditional ranking task [Savenkov, 2015] [Agarwal et al., 2012], where the given question and a set of answer candidates are analogous to a query and a set of relevant entities. Hence our task is transformed to find an optimal ranking order of these answer candidates according to their relevance to a given question. We choose the AdaRank[Xu and Li, 2007] and LambdaMART [Burgess, 2010] as the baseline learning-to-rank methods for our task. We used the *positive*, *neutral+* as the target value to define the order of each example. This is reasonable because the label establishment is part of our model, and the heuristic rules for setting up the *neutral-* and *negative* samples are never used before.
- **Traditional Classifiers** Recently Calefato et al. [Calefato et al., 2019] proposed to approach the best answer prediction problem as a binary-classification task, and in their work they assessed 26 best-answer prediction classifiers in Stack Overflow. We choose the two most effective traditional classifiers from their experimental results, xgbTree and RandomForest, for use in our study. As they were doing binary classification, to adapt to our training data, we kept our *positive samples* as positive and

consider *neutral*⁺ samples as negative. Thereafter, we utilize the classification models to generate an answer ranking list by pairwise comparison between the answer candidates.

- **AnswerBot** Xu et al. [Xu et al., 2017] proposed a framework called AnswerBot to generate an answer summary for a non-factoid technical question. Their user study showed a promising performance for selecting salient answers by their method. We adapted their AnswerBot approach for our task of recommending answers among a set of answer candidates. To be more specific, for a given question, AnswerBot generates a ranked list of candidate answers according to the ranking scores. This ranked list of answers is then used to calculate the precision of answer selection results.
- **IR-DeepAns** To verify the effectiveness of using clarifying questions as a way of question boosting, compared with our sequence to sequence model, we also considered a simple IR-based approach using similar clarifying questions as a query expansion mechanism. For a given question q_i , we first identified the most similar question q_j in $\langle q, cq \rangle$ dataset, and then retrieved the clarifying question cq_j associated with q_j . We applied IDF-weighted word embedding methods to calculate the similarity score between two questions. We feed the q_i and cq_j into our model and name this baseline as IR-DeepAns. This model is close to ours.

4.3.4 Evaluation Methods

Experiment Setup To thoroughly evaluate our model, we conducted a large-scale automatic evaluation experiment. We used IDF-weighted word embedding (described in Section 4.2.2) to calculate the similarity score between two question titles. For each testing qa pair $\langle q_t, a_t \rangle$, we then performed K-NN (K=5) to search for similar questions over the whole

data set for the given testing question q_t . We then constructed an answer candidate pool by gathering the top-5 answers associated with these selected questions. Since the top-similar question extracted by K-NN is always the original post itself, we can ensure that the accepted answer a_t paired with the original post q_t is always in the answer candidate pool. In other words, the answer candidate pool for testing question q_t contains 5 answers, one of which is the accepted answer a_t . In summary, for the 5,000 testing questions of each platform, we constructed $5,000 \times 5$ QA pairs in total to serve as the final evaluation sets. Following this, for each testing question q_t , we first applied the pre-trained question boosting model to generate a clarifying question cq_t . We then paired the given question with each answer in the candidate pool to construct the $\langle q_t \oplus cq_t, a_t \rangle$ pairs. The $\langle q_t \oplus cq_t, a_t \rangle$ pair was fitted into our model to calculate a matching score, and we then generated a ranking order for each group of candidate answers according to their matching scores to the given question.

Evaluation Metrics Since the evaluation answer candidate pool includes the accepted answer, one way to evaluate our approach is to look at how often the accepted answer is ranked higher up among members of the answer candidate pool. Thus we adopted the widely-accepted metric, $P@K$ and $DCG@K$ to measure the ranking performance of our model.

- $P@K$ is the precision of the best answer in top-K candidate answers. Given a question, if one of the top-k ranked answers is the best answer, we consider the recommendation to be successful and set $success(best_i \in topK)$ to 1, otherwise, we consider the recommendation to be unsuccessful and set $success(best_i \in topK)$ to 0. The $P@K$ metric is defined as follows:

$$P@K = \frac{1}{N} \sum_{i=1}^N [success(best_i \in topK)] \quad (4.11)$$

- $DCG@K$ is another popular top-K accuracy metric that measures a recommender

system performance based on the graded relevance of the recommended items and their positions in the candidate set. Different from $P@K$, the intuition of $DCG@K$ is that highly-ranked items are more important than low-ranked items. According to this metric, a recommender system gets a higher reward for ranking the correct answer at a higher position. The $success(best_i \in topK)$ is same with the previous definition, while the $rank_{best_i}$ is the ranking position of the best answer i . The $DCG@K$ is defined as follows:

$$DCG@K = \frac{1}{N} \sum_{i=1}^N \frac{[success(best_i \in topK)]}{\log_2(1 + rank_{best_i})} \quad (4.12)$$

4.4 Automatic Evaluation Results

To gain a deeper understanding of the performance of our approach, we conducted an analysis on our large-scale automatic evaluation results. Specifically, we mainly focus on the following research questions:

- *RQ-1*: How effective is our approach under automatic evaluation?
- *RQ-2*: How effective is our use of *Question boosting* and *Label establishing* methods?
- *RQ-3*: How effective is our approach under different parameter settings?

4.4.1 Effectiveness Analysis

The automatic evaluation results of our proposed model and aforementioned baselines over different technical Q&A sites are summarized in Table 4.2, Table 4.3, Table 4.4 and Table 4.5 respectively. We do not report $P@5$ and $DCG@1$ in our tables, since $DCG@1$ is always equal to $P@1$ and $P@5$ will always be equal to 1. The best performing system for each column is highlighted in boldface. As can be seen, **our model outperforms all the**

other methods by a large margin in terms of $P@K$ score and $DCG@K$ score. From the table, we can observe the following points discussed below.

1. Compared to traditional classifiers, such as xgbTree and RandomForest, one can clearly see that our approach performs much better. For example, it improves over xgbTree on $P@1$ by 42% on Ask Ubuntu dataset, and 39% on Super User dataset.
2. Compared with the method proposed by [Calefato et al., 2019], which only has two kinds of labels (*positive* and *negative*), our approach constructs four kinds of labeled data (*positive*, *neutral*⁺, *neutral*⁻, *negative*) automatically via incorporating the *label establishing* process. By introducing the *neutral*⁺ and *neutral*⁻ training samples, our approach can learn how to separate the best answer from the similar ones, which may explain the obvious advantage of our model in $P@1$.
3. Our approach also outperforms the AnswerBot by a large margin. We attribute this to the following reasons. Firstly, by adding a clarifying question into our model, we can properly fuse the information between the isolated question sentences and answers, which can reduce the lexical gap between them and better pair the answer with associated questions. Secondly, we use two parallel convolutional neural network block to learn optimal vector representation of QA pairs that preserving important syntactic and semantic features. To compute the matching score, we relate the rich representation features via a weakly supervised way from the available training data.

Table 4.2: Automatic evaluation (Ask Ubuntu)

Model	P@1	P@2	P@3	P@4	DCG@2	DCG@3	DCG@4	DCG@5
RandomForest	26.6 ± 1.6%	49.2 ± 1.6%	70.8 ± 1.6%	87.1 ± 0.4%	40.9 ± 1.5%	51.7 ± 1.5%	58.8 ± 0.9%	63.7 ± 0.8%
XgbTree	28.8 ± 1.4%	53.6 ± 1.3%	73.0 ± 1.0%	87.9 ± 1.2%	44.5 ± 1.2%	54.2 ± 0.9%	60.7 ± 0.8%	65.3 ± 0.7%
LambdaMART	25.4 ± 1.1%	45.7 ± 1.0%	65.7 ± 1.2%	84.0 ± 1.0%	38.5 ± 1.0%	47.5 ± 1.1%	55.8 ± 0.9%	62.3 ± 0.6%
AdaRank	24.9 ± 1.1%	45.3 ± 1.1%	65.0 ± 1.0%	82.9 ± 0.8%	38.1 ± 1.2%	47.2 ± 1.1%	55.2 ± 1.0%	61.8 ± 0.7%
AnswerBot	27.7 ± 1.6%	52.1 ± 1.5%	73.5 ± 1.0%	89.2 ± 0.7%	43.1 ± 1.5%	53.8 ± 1.1%	60.5 ± 0.8%	64.7 ± 0.8%
DeepAns-IR	37.2 ± 2.0%	59.9 ± 2.1%	77.5 ± 1.7%	92.0 ± 1.0%	50.8 ± 1.5%	59.6 ± 1.3%	65.8 ± 1.1%	68.7 ± 0.8%
DeepAns	40.9 ± 1.5%	61.7 ± 1.9%	77.9 ± 0.9%	92.0 ± 0.9%	54.0 ± 1.7%	62.1 ± 1.1%	68.2 ± 1.1%	71.3 ± 0.9%

4. Compared to our model, the learning-to-rank based approach achieved the worst performance regarding the $P@K$ and $DCG@K$ scores with different depths. The learn-

Table 4.3: Automatic evaluation (Super-User)

Model	P@1	P@2	P@3	P@4	DCG@2	DCG@3	DCG@4	DCG@5
RandomForest	27.4 ± 1.6%	50.2 ± 1.7%	70.5 ± 1.4%	87.3 ± 0.9%	41.7 ± 1.6%	51.9 ± 1.4%	59.2 ± 1.2%	64.1 ± 0.9%
XgbTree	29.2 ± 1.6%	55.9 ± 1.3%	74.2 ± 0.9%	88.5 ± 0.7%	47.6 ± 1.2%	56.7 ± 1.1%	63.0 ± 0.9%	67.4 ± 0.8%
LambdaMART	25.9 ± 1.1%	47.1 ± 1.0%	66.1 ± 1.0%	84.5 ± 1.2%	39.8 ± 1.0%	48.8 ± 1.0%	56.4 ± 0.7%	62.9 ± 0.6%
AdaRank	25.1 ± 1.2%	46.2 ± 1.1%	65.7 ± 1.0%	84.1 ± 0.9%	38.4 ± 1.1%	47.6 ± 1.1%	55.7 ± 1.0%	62.2 ± 0.9%
AnswerBot	29.8 ± 1.4%	53.9 ± 1.2%	74.1 ± 1.3%	89.6 ± 0.8%	45.0 ± 1.2%	55.1 ± 0.9%	61.8 ± 0.7%	65.8 ± 0.6%
DeepAns-IR	38.8 ± 2.1%	63.4 ± 1.8%	80.7 ± 1.2%	92.5 ± 1.2%	54.3 ± 1.8%	63.0 ± 1.3%	68.1 ± 1.2%	70.9 ± 1.0%
DeepAns	40.7 ± 1.9%	65.8 ± 1.1%	82.2 ± 1.1%	93.9 ± 0.8%	56.5 ± 1.2%	64.7 ± 1.2%	69.8 ± 1.0%	72.1 ± 0.8%

Table 4.4: Automatic evaluation (SO-Python)

Model	P@1	P@2	P@3	P@4	DCG@2	DCG@3	DCG@4	DCG@5
RandomForest	34.0 ± 1.3%	57.2 ± 1.0%	74.8 ± 0.7%	89.7 ± 0.6%	48.6 ± 0.9%	57.4 ± 0.6%	63.9 ± 0.7%	67.8 ± 0.5%
XgbTree	35.4 ± 1.5%	58.4 ± 1.9%	74.2 ± 1.5%	88.7 ± 1.1%	49.9 ± 1.6%	57.8 ± 1.3%	64.1 ± 1.0%	68.4 ± 0.8%
LambdaMART	32.6 ± 1.7%	56.2 ± 2.2%	73.7 ± 1.7%	88.3 ± 0.8%	47.5 ± 1.9%	56.3 ± 1.7%	62.5 ± 1.2%	67.1 ± 1.0%
AdaRank	29.9 ± 1.3%	53.3 ± 1.1%	71.4 ± 0.9%	85.8 ± 0.8%	44.7 ± 1.1%	53.7 ± 0.8%	59.9 ± 0.8%	65.4 ± 0.6%
AnswerBot	31.8 ± 1.8%	52.8 ± 2.0%	71.6 ± 1.9%	88.7 ± 0.9%	44.5 ± 1.7%	54.3 ± 1.6%	62.6 ± 1.2%	68.1 ± 0.9%
DeepAns-IR	42.8 ± 1.3%	62.8 ± 1.9%	78.2 ± 2.0%	90.0 ± 0.9%	55.4 ± 1.6%	63.1 ± 1.6%	68.2 ± 1.0%	72.1 ± 0.8%
DeepAns	45.7 ± 1.6%	65.7 ± 1.6%	80.2 ± 1.9%	92.1 ± 1.2%	58.3 ± 1.4%	65.6 ± 1.3%	70.7 ± 1.1%	73.8 ± 0.8%

ing to rank approach ignores the fact that ranking is a prediction task on a list of objects. Because they require a large number of training instances with ranking labels, therefore if the ground truth ordering of input candidates is lacking, they are unable to capture the relative preference between two QA pairs. This may explain the reason why its performance is comparatively suboptimal.

5. The DeepAns-IR approach has its advantage as compared to other baselines excluding our proposed model. This is because DeepAns-IR employs the same data labeling strategy and the model structure as ours. Moreover, it also incorporates the IR-based approach to expand the query with clarifying questions. This verifies the effectiveness of our model for question and answering tasks in technical Q&A sites. The only difference between DeepAns-IR and our model is that our model generates clarifying

Table 4.5: Automatic evaluation (SO-Java)

Model	P@1	P@2	P@3	P@4	DCG@2	DCG@3	DCG@4	DCG@5
RandomForest	32.9 ± 1.0%	56.1 ± 1.1%	74.1 ± 1.1%	89.5 ± 0.7%	47.6 ± 0.9%	56.6 ± 0.8%	63.2 ± 0.6%	67.2 ± 0.5%
XgbTree	35.9 ± 1.3%	59.0 ± 1.2%	75.7 ± 0.9%	89.1 ± 1.0%	50.5 ± 1.0%	58.8 ± 0.7%	64.6 ± 0.7%	68.8 ± 0.5%
LambdaMART	31.5 ± 1.2%	54.4 ± 1.2%	72.3 ± 1.7%	87.6 ± 1.3%	46.0 ± 0.8%	54.9 ± 1.1%	61.5 ± 0.7%	66.3 ± 0.5%
AdaRank	29.2 ± 2.1%	52.1 ± 2.2%	69.9 ± 1.8%	86.2 ± 1.5%	43.6 ± 1.8%	52.5 ± 1.7%	59.6 ± 1.5%	64.9 ± 1.1%
AnswerBot	34.7 ± 1.5%	58.0 ± 2.1%	77.8 ± 1.9%	90.2 ± 1.5%	49.4 ± 1.6%	59.3 ± 1.5%	64.7 ± 1.1%	68.4 ± 0.8%
DeepAns-IR	42.3 ± 2.9%	63.7 ± 2.3%	78.3 ± 2.1%	91.8 ± 1.6%	55.7 ± 2.4%	63.1 ± 2.2%	68.9 ± 1.8%	72.1 ± 1.4%
DeepAns	45.5 ± 1.6%	65.9 ± 2.2%	79.9 ± 1.6%	92.0 ± 0.9%	58.4 ± 1.9%	65.4 ± 1.5%	70.6 ± 1.2%	73.7 ± 0.9%

questions via deep sequence to sequence learning, while the DeepAns-IR retrieves the clarifying questions from the existing database according to a similarity score, which relies heavily on whether similar questions can be found and how similar the questions are. This results in our model’s superior performance as compared to the DeepAns-IR approach.

6. By comparing the evaluation results of the different technical Q&A sites, i.e., Ask Ubuntu, Super User and Stack Overflow, we can see that our proposed model is stably and substantially better than the other baselines. This suggests that our approach behaves consistently across different technical Q&A platforms, regardless of the different topic of the specific technical forums. This supports the likely generalization and robustness of our approach. We also notice that the advantage of our proposed model is much more obvious on SO (Python) and SO (Java) as compared to Ask Ubuntu and Super User. The reason for this phenomenon is likely the large number of training samples from Stack Overflow which benefits the classification performance of our model.

In summary, our model substantially outperforms the baselines under automatic evaluation.

4.4.2 Ablation Analysis

Ablation analysis is used to verify the effectiveness of the *DeepAns* using *Question boosting* and *Label establishing* methods. More specifically, we compare our approach with several of its incomplete variants:

- **Drop CQ:** removes the clarifying question part generated by *Question boosting* model.
- **Drop Labeling:** removes the training samples generated by *Label establishing*

Table 4.6: Ablation Evaluation (Ask Ubuntu)

Measure	Drop CQ	Drop Labeling	DeepAns
P@1	34.2 ± 1.3%	31.3 ± 1.2%	40.9 ± 1.5%
P@2	58.9 ± 1.8%	50.5 ± 1.1%	61.7 ± 1.9%
P@3	77.3 ± 1.5%	68.9 ± 1.3%	77.9 ± 0.9%
P@4	91.4 ± 0.8%	86.0 ± 1.1%	92.0 ± 0.9%
DCG@2	49.8 ± 1.5%	43.4 ± 0.9%	54.0 ± 1.7%
DCG@3	59.5 ± 1.2%	51.7 ± 0.8%	62.1 ± 1.1%
DCG@4	65.8 ± 0.9%	59.1 ± 0.7%	68.2 ± 1.1%
DCG@5	68.5 ± 0.7%	64.5 ± 0.5%	71.3 ± 0.9%

Table 4.7: Ablation Evaluation (Super User)

Measure	Drop CQ	Drop Labeling	DeepAns
P@1	35.8 ± 1.4%	29.7 ± 1.4%	40.7 ± 1.9%
P@2	60.2 ± 1.0%	53.9 ± 1.9%	65.8 ± 1.1%
P@3	79.6 ± 0.9%	72.5 ± 1.5%	82.2 ± 1.1%
P@4	92.1 ± 0.5%	89.5 ± 0.9%	93.9 ± 0.8%
DCG@2	51.2 ± 1.1%	45.0 ± 1.6%	56.5 ± 1.2%
DCG@3	60.9 ± 0.9%	54.0 ± 1.4%	64.7 ± 1.2%
DCG@4	66.3 ± 0.7%	61.4 ± 1.1%	69.8 ± 1.0%
DCG@5	69.3 ± 0.7%	65.6 ± 0.8%	72.1 ± 0.8%

model, to do this, we keep the best QA pairs as *positive samples*, and make other answer pairs as *negative samples*. Our model was trained as a binary classification model.

We performed the ablation analysis experiment on Ask Ubuntu and Super User respectively. The ablation analysis results are presented in the Table 4.6 and Table 4.7. We can observe the following points.

1. By comparing the results of our approach with each of the variant model, we can see that no matter which method we dropped, it does hurt the performance of our model. This verifies the importance and effectiveness of these three mechanisms.

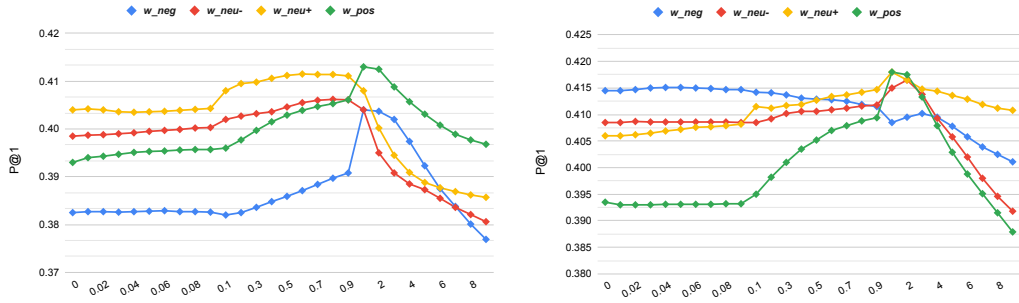


Figure 4.5: Sensitivity Analysis on Ask-ubuntu (left) and Super-user (right)

2. By comparing the results of **DeepAns** with **Drop CQ**, it is clear that incorporating a clarifying question improves the overall performance. When adding a clarifying question to our model, the $P@k$ score is improved by 19.5% and 13.9% on Ask Ubuntu and Super User dataset respectively. We attribute this to that the useful clarifying question can reduce the lexical gap between answer and questions, which can make the information properly fused between them.
3. By comparing the results of **DeepAns** with **Drop Labeling**, we can measure the performance improvements achieved due to the incorporation of “Label establishment” process. After removing the training samples constructed by *Label establishment*, there is a significant drop overall in every evaluation measure. This is because by employing our *label establishing* process, the size of the training data is largely expanded, in the meanwhile, by introducing $neutral^+$ and $neutral^-$ samples, our model can learn to better distinguish best answer from similar ones.

In summary, both the *question boosting* module and *label establishing* model are effective and helpful to enhance the performance of our approach.

4.4.3 Parameters Tuning Analysis

In this section, we tune the key parameters of our model for sensitivity analysis and robustness analysis.

Sensitivity Analysis We have four key parameters (i.e., ω_{pos} , ω_{neu+} , ω_{neu-} , ω_{neg}) in Equation 4.10. The optimal settings of these weights were carefully tuned on our dataset. We demonstrate the weights tuning on Ask Ubuntu and Super User respectively. In particular, the validation set was leveraged to validate our model and the grid search method was employed to select optimal parameters between 0 and 10 with small but adaptive step sizes. The step sizes were 0.01, 0.1, and 1 for the range of [0, 0.1], [0.1, 1] and [1, 10], respectively. The parameters tuning process was varying one weight while fixing the other three weights. For example, in order to tune the parameter ω_{neg} , we fix the other three parameters and change ω_{neg} from 0 to 10 with different step sizes. After that, we fix ω_{neg} to its optimal settings for tuning other parameters. Fig. 4.5 illustrates the performance of our model with respect to different weights on Ask Ubuntu and Super User respectively. From the figure, we have the following observations:

1. Even though the four parameters vary in a relatively wide range, the performance of our proposed model DEEPANS changes within small ranges near the optimal settings. This indicates that our model is non-sensitive to the parameters around their optimal settings, which further supports the generalization ability of our approach.
2. We notice that most parameters achieve their best performance in the range of [1, 3], we thus recommend to initialize the weights in Equation 4.10 to be around the above range, which is close to the optimal settings of our model.

Robustness Analysis In real world Q&A sites, there is no guarantee to find the exactly matched questions from the archive, especially when k is small. Therefore we have to enlarge k to improve the recall of the similar questions and hence the “matched answers”. However, a larger k may introduce more noise into the answer candidate pool with more irrelevant answers. This can then increase the difficulty of our answer recommendation task.

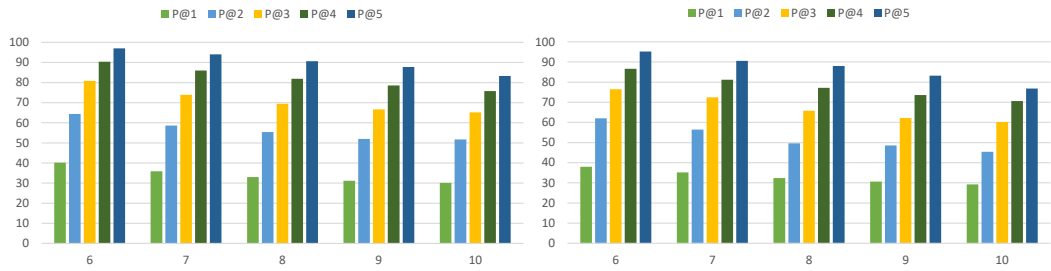


Figure 4.6: Robustness Analysis on Ask-ubuntu (left) and Super-user (right)

To verify the robustness of our proposed approach, we set different thresholds for the number of returned questions by k-NN method. More specifically, we varied the number of returned similar questions k from 6 to 10 and measured the performance of our approach, we then reported average P@1-5 over each dataset under different parameter settings of k . The results of Ask Ubuntu and Super User are shown in Fig. 4.6. We can make the following observations:

1. The trend in overall performance of our model decrease as k increases, which supports our concern that larger k settings introduce more noises and bring bigger challenges for our task. By analyzing the performance of our approach with respect to different k , we notice that our approach achieves good performance when k varies from 5 to 7, while still ensuring the “matched answer” is highly-ranked. We thus recommend setting k within the above range for real-world applications.
2. The advantage of our proposed model is more obvious on $P@1$ compared with other metrics($P@2 - 5$). Even when we set k to 10, the performance of our model on $P@1$ is still on a par with the best performance of other baselines, while k is set to 5 in these baselines (See Table 4.2 and Table 4.3). This reveals that our model can perform well under a noisy context, which shows the robustness of our model.

In summary, our model is non-sensitive and robust under different parameter settings.

4.5 User Study Setup and Results

Since automatic evaluation results do not always agree with the actual ranking preference of real-world users, we also performed a small, qualitative user study to measure how humans actually perceive the results produced by our approach. Specifically, we mainly focus on the following research questions:

- *RQ-4*: How effective is the question boosting results of our approach under human evaluation?
- *RQ-5*: How effective is the question answering results of our approach under human evaluation?

For human evaluation, we used the Ask Ubuntu and Stack Overflow (Python) platforms to perform our user study. We invited 5 evaluators to participate in our user study; all of these participants have more than three years of studying/working experience in software development process, have more than one year of experience using technical Q&A sites, and are familiar with the Ubuntu system and Python programming languages. We did not limit the amount of time for evaluators to complete the user study.

Human Evaluation on Question Boosting Results To gain a deeper understanding of how the clarifying questions impact the results in our study, we conducted human evaluation studies to measure how humans perceive the question boosting results. To do this, we consider two modalities in our user study: *Relevance* and *Usefulness*. *Relevance* measures how relevant the clarifying question is to the original question title. *Usefulness* measures how useful the clarifying question is for adding missing information for the original post. We randomly sampled 25 $\langle q, a \rangle$ pairs from Ask Ubuntu and SO (Python) respectively. For each question, we provided two clarifying questions. One was generated by our approach, the other was generated by the IR-based approach, i.e., DeepAns-IR. We also provided the accepted answer to the question as a reference. We asked the participants to manually rate

Table 4.8: Human Evaluation of Question Boosting Results

Data	Model	Score(1) _R	Score(2) _R	Score(3) _R	Avg _R	Score(1) _U	Score(2) _U	Score(3) _U	Avg _U
Ask Ubuntu	IR-based	21.6%	43.2%	35.2%	2.14	28.8%	34.4%	36.8%	2.08
	Ours	18.4%	32.8%	48.8%	2.30	22.4%	35.8%	42.4%	2.20
SO (Python)	IR-based	19.2%	36.0%	44.8%	2.26	26.4%	32.0%	41.6%	2.15
	Ours	17.6%	32.0%	50.4%	2.33	23.2%	29.6%	47.2%	2.24

the generated clarifying questions on a scale between 1 and 3 (1 = worst, 3 = best) across the above modalities. The volunteers were blinded as to which question title was generated by our approach.

Evaluation Results. We obtained 125 groups of scores from evaluators for Ask Ubuntu and SO (Python) respectively. Each group contains two pairs of scores, which were rated for clarifying questions produced by IR-based approach and ours. Each pair contains a score for the *Relevance* modality and a score for *Usefulness* modality. The score distribution and average score of *Relevance* and *Usefulness* across the two methods are presented in Table 4.8. From the table, we can observe the following points:

1. Our approach performs better than the IR-based approach on both modalities. We attribute this to the following reason: the IR-based approach relies heavily on whether similar clarifying questions can be retrieved from the existing $\langle q, cq \rangle$ dataset. Considering the complexity of the questions in technical Q&A sites, there may exist only a few questions that are very similar to the given one, hence it is difficult to retrieve relevant clarifying questions from the training set.
2. Both the IR-based approach and our approach can produce relevant and useful clarifying questions for the given question. This further verifies the clarifying question is helpful in adding missing information and reducing the gap between questions and answers. We also notice that there are still quite a few questions that received low scores for *Relevance* and *Usefulness* modalities. Even though the clarifying questions generated by our approach are still not perfect, our study is the first step on this topic and we also release our data to inspire follow-up work for utilizing the

<p>Ex1. Question: (41969 - SO Python) Standard way to open a folder window in linux?</p> <p>Answer: os.system('xdg-open "%s"' % foldername) xdg-open can be used for files/urls also</p>	<p>Ex2. Question: (2093 - Ask Ubuntu) How to install GUI desktop on a server?</p> <p>Answer: Depending on which desktop you wish to install, you install the the meta-package that installs all the necessary packages. You can use apt-get or aptitude to do this. sudo apt-get install ubuntu-desktop ninstalls the Unity desktop sudo aptitude install kubuntu-desktop would install the KDE desktop Other desktop meta-packages are xubuntu-desktop, ubuntu-gnome-desktop, lubuntu-desktop, and edubuntu-desktop .</p>	<p>Ex3. Question: (1263 - Ask Ubuntu) Unable to update thunderbird</p> <p>Answer: Nope. The only way to allow Thunderbird to update itself is to launch it as root, or to find yet another source that built Thunderbird in such a way that you can install it entirely within your own home folder. A better option, if you want to run cutting-edge apps, is to try to find someone making builds on the in a Launchpad PPA. These can update through the normal Synaptic/Upgrade system, which means you won't end up with corner cases like this. As it happens, the Mozilla team maintains a PPA of newer versions of Firefox and Thunderbird, including even nightlies. If you install Thunderbird 3.1 from there, you should be golden.</p>
<p>Clarifying Question (Ours): have you tried 'xdg-open folder()' ? Relevance: score_3(5) / score_2(0) / score_1(0) Usefulness: score_3(5) / score_2(0) / score_1(0)</p>	<p>Clarifying Questions (Ours): did you try 'sudo apt-get install -f' Relevance: score_3(5) / score_2(0) / score_1(0) Usefulness: score_3(4) / score_2(1) / score_1(0)</p>	<p>Clarifying Question (Ours): how did you install thunderbird ? Relevance: score_3(2) / score_2(1) / score_1(2) Usefulness: score_3(2) / score_2(2) / score_1(1)</p>
<p>Clarifying Question (IR-based): have you looked as 'os.walk()' ? Relevance: score_3(3) / score_2(2) / score_1(0) Usefulness: score_3(3) / score_2(1) / score_1(1)</p>	<p>Clarifying Question (IR-based): how did you try to install gnome desktop? Relevance: score_3(4) / score_2(1) / score_1(0) Usefulness: score_3(3) / score_2(1) / score_1(1)</p>	<p>Clarifying Question (IR-based): do you have any thunderbird extensions installed? Relevance: score_3(2) / score_2(2) / score_1(1) Usefulness: score_3(3) / score_2(1) / score_1(1)</p>

Figure 4.7: Evaluation Examples of Question Boosting.

clarifying questions.

Evaluation Examples. A major challenge for question answering tasks is the semantic gap between the questions and answers. This is because the questions from technical Q&A sites are, more often than not, very specific and complex, and oriented towards expert professional answers. To fill the gap between question and answers, we employ a deep encoder-decoder model to generate a clarifying question for a given post as a way of question boosting. Fig. 4.7 presents three examples of human evaluation on question boosting results (the words that appear in both clarifying questions and answers are highlighted). From these cases, we can see that:

1. The clarifying questions produced by our approach as well as the IR-based approach generally perform well across both modalities. It is clear that the clarifying question can reduce the lexical gap between the answer and the questions, which can add missing information and make the information better linked between question and answers. For example, in the first and second case, our approach generates “xdg-open” and “sudo apt-get install” for the clarifying questions which also appear in the answers. Thus, the added information can eliminate and/or reduce the isolation between questions and answers. We attribute this to the advantage of our model for

learning common patterns automatically from the $\langle q, cq \rangle$ pairs.

2. Not all the clarifying questions are appreciated by the evaluators; an example is shown in the last row of Fig. 4.7. For such cases, even though the generated clarifying question is not optimal to the participants, our approach still precisely replicates the salient tokens, i.e., “thunderbird” from the question title, which also increases the likelihood of selecting the right answer from answer candidates.

In summary, the clarifying questions generated by our approach are effective under human evaluation results.

Human Evaluation on Question Answering Results Since the final goal of our study is recommending relevant answers to developers, we also performed a human evaluation to measure the effectiveness of question answering results with respect to human developers. To be more specific, we measured how developers perceive the answers produced by our approach to solved questions, unresolved questions and unanswered questions. For solved questions, we compared our approach with the ground truth; for unresolved questions, we compared our approach with xgbTree and Answerbot methods; and for unanswered questions, we compared our approach with Stack Exchange search engine and Google search engine.

User Study on Solved Questions In order to investigate the agreement of the developers on solved questions, we randomly sampled 25 examples of solved questions from the testing set of Ask Ubuntu and SO (Python) respectively. For each solved question, we provided two answer candidates. One answer was the accepted answer – we refer to it as the ground truth in this study. The other answer was produced by our approach. After that, each evaluator was asked to manually rate on the two answer candidates from 1 to 3, according to the acceptance of the answer. Score 3 means that the evaluator strongly agrees with the acceptance of the answer, and score 0 means that the evaluator strongly disagrees with the acceptance of the answer. It is worth emphasizing that the answer selected by our approach

Table 4.9: Human Evaluation - Ask Ubuntu

Type	Approach	Score(1)	Score(2)	Score(3)	$Rank_{avg}$
Solved	Ground Truth	7.2%	18.4%	74.4%	2.67
	DeepAns	19.2%	32.8%	48.0%	2.29
Unresolved	xgbTree	15.2%	26.4%	58.4%	2.43
	AnswerBot	12.8%	28.0%	59.2%	2.46
	DeepAns	12.0%	23.2%	64.8%	2.53
Unanswered	SE Engine	51.2%	33.6%	15.2%	1.64
	Google	25.6%	32.8%	41.6%	2.16
	DeepAns	22.4%	30.4%	47.2%	2.25

Table 4.10: Human Evaluation - SO (Python)

Type	Approach	Score(1)	Score(2)	Score(3)	$Rank_{avg}$
Solved	Ground Truth	5.6%	14.4%	80.0%	2.74
	DeepAns	18.4%	31.2%	50.4%	2.32
Unresolved	xgbTree	12.0%	26.4%	61.6%	2.50
	AnswerBot	9.6%	32.0%	58.4%	2.49
	DeepAns	10.4%	21.6%	68.0%	2.58
Unanswered	SE Engine	54.4%	32.0%	13.6%	1.59
	Google	30.4%	31.2%	38.4%	2.08
	DeepAns	26.4%	28.0%	45.6%	2.19

may actually be the same with the ground truth answer, and the participants were blinded as to which answer is the ground truth.

Evaluation Results. We collected 125 groups of scores from participants for Ask Ubuntu and SO (Python) respectively. Each group contains two scores, which were rated for answers of the ground truth and ours. We count the proportion of different scores and calculate the average score for each method. The evaluation results for Ask Ubuntu and SO (Python) are presented in Table 4.9 and Table 4.10 respectively. From the table, we can observe the following points:

1. The evaluators are in agreement with acceptance of the ground truth answers for most cases. For example, around 75% of the ground truth answers in Ask Ubuntu and 80% answers in SO (Python) are appreciated by the volunteers.
2. The ground truths are better than our approach. This is reasonable because the ground truth answers are usually high-quality answers that have been accepted by the devel-

opers. Even though our approach is not as good as the ground truth at the current stage, we observe that a small number of answers produced by our approach are marked with score 1. This indicates that the answers selected by our approach are meaningful and acceptable for the majority of questions.

Evaluation Examples. Fig. 4.8 shows three examples of the user study on solved questions. It can be seen that:

1. In general, our approach can produce acceptable answers. Sometimes, the answers chosen by our approach are actually more accepted by the volunteers than the ground truth answers. For example, in the first sample, three evaluators gave a score of 3 to the ground truth answer, while four evaluators gave a score of 3 to ours. However, our answer does not belong to the current question thread and is selected from answer candidates of other questions (e.g., *Python: duplicating each element in a list*). This further justifies the feasibility of addressing *answer hungry* problem by selecting answers from the historical QA dataset.
2. Outputs from our model are not always “correct”. For example, in the last sample, the information seeker asks a question of “*Can I download Ubuntu 12.04 on a notebook/laptop?*”, while the answer provided by our approach is about how to download a file from the packages. This example reveals that considering the complexity of the questions in technical Q&A sites, the gap between the ground truth answers and ours is still large, and hence there is still a large room for our question answering system to be further improved.

User Study on Unresolved Questions To investigate how developers perceive our approach to solve the unresolved questions, we sampled 25 unresolved questions for Ask Ubuntu and SO (Python) respectively. Each question has multiple answer candidates that have not been selected as *Accept*. By computing the matching score between question and

Ex1. Solved Question (493367 - SO Python): Python: For each list element apply a function across the list ?	Ex2. Solved Question (742371 - SO Python) Why does Python skip elements when I modify a list while iterating over it?	Ex3. Solved Question (230712 - Ask Ubuntu): Can I download Ubuntu 12.04 on a notebook/laptop?
<p>Answer(Ground Truth) : score_3(3) / score_2(2) / score_1(0)</p> <p>You can do this using list comprehensions and min() (Python 3.0 code):</p> <pre>>>> nums = [1,2,3,4,5] >>> [(x,y) for x in nums for y in nums] >>> min(_, key=lambda pair: pair[0]/pair[1])</pre> <p>Note that to run this on Python 2.5 you'll need to either make one of the arguments a float, or do from __future__ import division so that 1/5 correctly equals 0.2 instead of 0.</p>	<p>Answer(Ground Truth) : score_3(4) / score_2(1) / score_1(0)</p> <p>This is a well-documented behaviour in Python, that you aren't supposed to modify the list being iterated through. Try this instead:</p> <pre>for i in x[:]: x.remove(i)</pre> <p>The [] returns a "slice" of x, which happens to contain all its elements, and is thus effectively a copy of x.</p>	<p>Answer(Ground Truth) : score_3(5) / score_2(0) / score_1(0)</p> <p>Ubuntu-desktop runs on notebooks and laptops. If you are concerned about the performance of your laptop running the desktop version of Ubuntu you should try Lubuntu which is a lightweight distro based on Ubuntu but using the LXDE desktop environment. Lubuntu 12.10 is the latest version.</p>
<p>Answer(Ours) : score_3(4) / score_2(1) / score_1(0)</p> <p>Using list comprehension:</p> <pre>>>> oldList = [1, 2, 3, 4] >>> newList = [x for x in oldList for _ in range(2)] >>> newList</pre> <p>Above list comprehension is similar to following nested for loop.</p> <pre>newList = [] for x in oldList: for _ in range(2): newList.append(x)</pre>	<p>Answer(Ours) : score_3(3) / score_2(2) / score_1(0)</p> <p>You edit the list while loop over it. Since you only keep pointers for looping you remember positions, but they change since elements vanish. Try remembering the elements you want to delete by something like:</p> <pre>dt_list = [3600, 2700, 1800, 900] dt_list = [x for x in dt_list if (3600/x).is_integer()]</pre>	<p>Answer(Ours) : score_3(2) / score_2(2) / score_1(1)</p> <p>Yes, you can download the .deb file from packages.ubuntu.com and then open it like an archive (using file-roller). There are two more archives inside, you'll need to open "data.tar.gz".</p>

Figure 4.8: Examples of Solved Questions

each answer candidate, we can identify a best answer via our approach, xgbTree and Answerbot respectively (note that different approaches may choose the same answer as the best answer). Following that, we ask each evaluator to rank three answer candidates produced by our approach, xgbTree and Answerbot from 1 to 3 (3 is the best) according to the acceptance of the answer. It is worth emphasizing that the answers identified by our approach and others could be the same, and the order of the answers is randomly decided.

Evaluation Results. The human evaluation results of unresolved questions for Ask Ubuntu and SO (Python) are presented in Table 4.9 and Table 4.10 respectively. From the table, we can see that:

1. Our model performs better than xgbTree and Answerbot baselines. This further indicates that the answers selected by our approach are more appreciated by evaluators. The results of human evaluation on unresolved questions are consistent with large-scale automatic evaluation results, which reconfirms the effectiveness of our approach for identifying the best answer in unresolved questions.
2. Compared with the evaluation results of ground truth, the average scores between the answers of unresolved questions and solved questions are close, which supports our previous assumption that users forget to mark the accepted answer is not uncommon

Ex1. Unresolved Question (12109648 - SO Python): Python: how to adjust x axis in matplotlib ?	Ex2. Unresolved Question (12035394 - SO Python) Python - replace random items in column	Ex3. Solved Question (1169424 - Ask Ubuntu): How do I put two Ubuntu OSes on the same hard drive??
Answer(xgbTree) : score_3(3) / score_2(2) / score_1(0) Look at the docs : xlocs, xlabs = plt.xticks() put in xlocs your range, and in xlabs what you want to display, then: plt.xticks(xlocs, xlabs)	Answer(xgbTree & Ours) : score_3(4) / score_2(1) / score_1(0) I would strongly suggest reading a python primer, the way your problem solved can be done in two steps 1. read items from file - reference 2. use math.random() to change random string- reference by know how to do these points you can easily achieve what you intend to do.	Answer(xgbTree) : score_3(3) / score_2(2) / score_1(0) You don't put two OS in the same partition. Each instance of Ubuntu or an Ubuntu family OS requires its own partition. You can run the installer for the second version you wish to install, and choose 'Something Else' when you get to Installation Type, and let it share the same drive; just not the same partition. See this step-by-step workflow illustration for Ubuntu 18.04 LTS.
Answer(Answerbot) : score_3(3) / score_2(1) / score_1(1) The size of the plot can be changed by setting the dynamic rc settings of Matplotlib. These are stored in a dictionary named rcParams. The size of the plot figure is stored with the key figure.figsize.	Answer(Answerbot) : score_3(3) / score_2(1) / score_1(1) Use this to generate a random string <pre>import os random_string = os.urandom(string_length)</pre> To loop over a file line by line, do <pre>with open('file') as fd: for line in fd: # do stuff</pre> No need to close the file handle, use split to well, split on whitespace and place the result in an array (indexing starts at 0) Read more at docs.python.org . Please update your question with some code when you have gotten that far... Good luck	Answer(Ours & Answerbot) : score_3(3) / score_2(2) / score_1(0) Assuming there is nothing you want to preserve on your hard drive, you can do the following: 1. When you install the first instance of Ubuntu, at some point during installation there will be a window called "Installation type". Choose "Something else". 2. On the next window you can erase and create partitions. Make sure you create at least two partitions large enough for Ubuntu installations. 3. Select the partition on which you want to install the first instance of Ubuntu and press Change. Select Ext4 under "Use as" and "/" under "Mount point". 4. Go ahead, finish the installation. 5. Boot from the installation USB again and repeat step 1. 6. No need to repartition, as that was done in step 2. already. 7. Select the second partition and repeat Step 3. for it. 8. Go ahead, and finish the installation of the second instance. The next time you boot the computer Grub will allow you to boot to whichever instance you want.
Answer(Ours) : score_3(4) / score_2(1) / score_1(0) It sounds like you want to changes the limits of the plotting display - for that use xlim (and ylim for the other axis). To change the xticks themselves is provided in the answer by @fp. Show below is an example using without/with xlim: <pre>import pylab as plt plt.subplot(2,1,1) plt.hist(X, bins=300) plt.subplot(2,1,2) plt.hist(X, bins=300) plt.xlim(0,100) plt.show()</pre>		

Figure 4.9: Examples of Unresolved Questions

in technical Q&A sites.

Evaluation Examples. Fig. 4.9 shows the examples of the user study on unresolved questions. It can be seen that:

1. The overall answer quality for the unresolved questions is good. This is because these answers are directly related to the specific problems of the questions, which are more suitable to the needs of information seekers.
2. Even all the answer candidates of an unresolved question aim at solving the same problem. As can be seen, some answers identified by our approach stand out from the rest and are more appreciated by evaluators, such as samples 1-2. This further verifies the ability of our approach to select the most relevant answer from a set of answer candidates.

User Study on Unanswered Questions Similar to unresolved questions, We also randomly sampled 25 examples of unanswered questions for Ask Ubuntu and SO(Python) respectively. For each unanswered question, considering that developers usually search for technical help using Google search engine and/or the Q&A site search engine itself, we compare our approach against two baselines built based on the above search engines re-

spectively. We used the question title of the post as the search query. For Google search engine, we add “site:stackoverflow.com” and “site:askubuntu.com” to the end of the search query so that it searches only posts on Stack Overflow and Ask Ubuntu respectively. We use the first ranked question returned by Google search engine as the most relevant question, we extracted the accept answer or the answer with the highest vote if there is no accepted answer of the relevant question. For technical Q&A site search engine, we refer to the first ranked related question recommended by the technical Q&A site search engine as the most relevant question, and extracted the associated accepted answer or the highest-vote answer. After constructing the evaluation set for unanswered questions, for each unanswered question, we asked the evaluators to rank on the 3 answer candidates from 1 to 3 (3 for the best answer), The higher grade indicates that the answer is more suitable to the given question. Please note that the participants do not know which answer is generated by which approach.

Evaluation Results. The expert evaluation results of unanswered questions for Ask Ubuntu and SO (Python) are presented in Table 4.9 and Table 4.10. We can observe the following points:

1. Compared with baselines, our model outperforms SE (Stack Exchange search engine) and Google (Google search engine). This suggests that the answers produced by our approach are considered to be more suitable to the given question by the evaluators. We attribute this to the reason that Google search engine identifies the answer via searching from similar questions, thus it is unable to judge the matching degree between the questions and answers. In contrast, our approach estimates the matching score using the context information of the qa pair, which fills the gap between questions and answers. The superior performance of our approach in terms of average score further supports the effectiveness of our approach in identifying the best answer.

<p>Ex1. Unanswered Question (57147927 - SO Python): Python: How to import package correctly in python?</p> <p>Answer(Stack Overflow) : score_3(0) / score_2(2) / score_1(3)</p> <p>For dictionaries x and y, z becomes a shallowly merged dictionary with values from y replacing those from x. In Python 3.5 or greater: z = {"x": **y} Resources on Dictionaries.</p>	<p>Ex2. Unresolved Question (57152181- SO Python) How to read a bit field integer in python?</p> <p>Answer(Stack Overflow) : score_3(0) / score_2(1) / score_1(4)</p> <p>For dictionaries x and y, z becomes a shallowly merged dictionary with values from y replacing those from x. In Python 3.5 or greater: z = {"x": **y} ... Resources on Dictionaries.</p>	<p>Ex3. Solved Question (1179515 - Ask Ubuntu): Ubuntu 18.04 reboots randomly ?</p> <p>Answer(Stack Overflow) : score_3(2) / score_2(2) / score_1(1)</p> <p>Your OEM system vendor Asus probably placed the 24GB SSD drive for caching purposes, e.g. Intel Smart Response. This could explain why you can't set it as a boot drive. I suffer from the same issue in my HP laptop with an mSATA SSD slot and HP's response to this is "this is intentional".</p> <p>One downside of this approach is that your system will fail to boot even if just one of the drives fails.</p>
<p>Answer(Google) : score_3(2) / score_2(2) / score_1(1)</p> <p>"I have a medium size Python application with modules files in various subdirectories." Good. Make absolutely sure that each directory include a <code>__init__.py</code> file, so that it's a package. "I have created modules that append these subdirectories to <code>sys.path</code>" Bad. Use <code>PYTHONPATH</code> or install the whole structure Lib/site-packages. Don't update <code>sys.path</code> dynamically. It's a bad thing. Hard to manage and maintain. Bad. Use <code>PYTHONPATH</code> or install the whole structure Lib/site-packages. Don't update <code>sys.path</code> dynamically. It's a bad thing. Hard to manage and maintain. My current project has 100's of modules, a dozen or so packages. Each module imports just what it needs. No magic.</p>	<p>Answer(Google) : score_3(1) / score_2(3) / score_1(1)</p> <p>The following code will load the requested portions of the binary number into the fields: class Register(object): def __init__(self,x): self.fieldwidths = [6,12,6,4,12,8,16] Which will result in: BitField1 = 0b111011 BitField2 = 0b111011101110 The results may not be what you wanted because of the fact that the data you provided is not 64 bits but rather 128 bits, which would mean that the 64 most significant bits of the input data will be ignored by the program.</p>	<p>Answer(Google) : score_3(3) / score_2(2) / score_1(0)</p> <p>This sounds like a combination of issues.</p> <p>In the case of an individual system rebooting randomly I would want to replace the power supply in the chassis with one that provided more than adequate amperage for the connected components (as you want it to keep running during periods of peak power draw).</p> <p>In the case where the entire rack reboot simultaneously I would look at an inadequate UPS as the root cause or possibly an overheat condition due to AC failure in the server location.</p> <p>An intermittent short in the feed cord to the multi-tap could also result in the multiple reboot result that you describe.</p>
<p>Answer(Ours) : score_3(3) / score_2(2) / score_1(0)</p> <p>Since there are already many answers on SO for this*, I will focus on question (2). About what is a better code organization: The (relative) import is done as follows, from inside module_2: from ..pkg1 import module1 as m1 Alternatively, you can use absolute imports, which refer to the top package name: from top_pkg_name.pkg1 import module1 as m1 In such an organization, when you want to run any module as a script, you have to use the <code>-m</code> flag: python -m top_pkg_name.pkg1.module1</p>	<p>Answer(Ours) : score_3(4) / score_2(1) / score_1(0)</p> <p>In Python 3 you can use something like this: int.from_bytes(byte_string, byteorder='little')</p>	<p>Answer(Ours) : score_3(3) / score_2(2) / score_1(0)</p> <p>I'm guessing hardware problems.</p> <p>Your Lubuntu problem "Press S to skip mounting or Press M for Manual recovery." could have been from corrupted filesystems. And random reboots could be an indication of bad RAM too.</p> <p>I had a laptop with bad RAM that would reboot just like that. after 20-40 minutes, the RAM was physically dirty & I think even had corroded contacts, after a while it wouldn't work at all with that RAM</p>

Figure 4.10: Examples of Unanswered Questions

2. For the unanswered questions, a gap for the answer quality between unanswered questions and solved/unresolved questions still exists. We also notice that our approach received more low scores ($score = 1$) with unanswered questions as compared to solved/unresolved questions. This is because in technical Q&A sites, some questions are rather complicated and sophisticated and it is hard to find suitable question-specific answers for these questions.

Evaluation Examples. Fig. 4.10 shows three examples of the user study on unanswered questions. we can observe the following points:

1. The search engine of the technical Q&A site achieves worst performance. For example, in sample 1 and sample 2, the SE search engine recommends the same answer to two different questions. This is why the evaluators give comparatively low scores to the answers identified by SE search engine.
2. Our approach has its advantage as compared to the Google search engine (e.g., sample 1-2). This is because the Google search engine does not consider the contextual

information between the questions and answers, but instead only identifies the answers based solely by searching for similar questions. By contrast, our approach takes the question as well as the candidate answers and calculates the matching score between the question and the answers, which results in its superior performance compared to the other baselines.

3. In technical Q&A sites, some question titles are relatively abstract and uninformative. For example, in sample 3, even the answer selected by our approach is relevant and meaningful, we can not make sure if the answer solves the actual problem or not. For such cases, more detailed information, such as the description in the question body, could be considered when searching for appropriate answers.

In summary, our model is comparatively effective under human evaluation for question answering tasks in technical Q&A sites.

4.6 Discussion

In this section, we first discuss the strength of our approach as well as the threats to validity of our work, after that we analyze the outlier cases involving in our data creation process.

4.6.1 Strength of Our Approach

To address the answer hungry problem in technical Q&A sites, we propose a deep learning based approach DEEPANS to search relevant answers from historical QA pairs. We summarized the strength of our approach as follows:

Neural Language Model for Question Boosting One advantage of our approach is training an attentional sequence-to-sequence model for generating clarifying questions as a way of question boosting. Instead of searching similar clarifying questions, our approach builds a neural language model for linking semantics of question and clarifying questions. The neural language model is able to handle the uncertainty in the correspondence between the

questions and clarifying questions. Our approach automatically learns common patterns automatically from the $\langle q, cq \rangle$ pairs. The encoder itself is a neural language model which is able to remember the likelihood of different kinds of questions. Following that, the decoder learns the context of the questions fills the gap between the questions and clarifying questions.

Label Establishment for Data Augmentation Due to the reason of the professional questions in technical Q&A sites, it is thus very hard, if not possible, to find experts and annotators for manual labeling the QA pairs. In this paper, we present a novel labeling scheme to automatically construct *positive*, *neutral⁺*, *neutral⁻*, and *negative* training samples. Guided by our four heuristic rules, this label establishment process can collect large amounts of labeled QA pairs, which greatly saves the time-consuming and labor-intensive labeling process.

Deep Neural Network for Answer Recommendation We present a weakly supervised neural network for the answer recommendation task in technical Q&A sites. Our model architecture is able to incorporate the aforementioned four types of training samples for ranking QA pairs. Our work first uses the deep neural network to solve the problem of best answer selection in technical Q&A sites, which is able to alleviate the answer hungry phenomenon that widely exists in technical Q&A forums.

4.6.2 Threats to Validity

We have identified the following threats to validity among our study:

Internal Validity Threats to internal validity are concerned with potential errors in our code implementation and study settings. For the automatic evaluation, in order to reduce errors, we have double-checked and fully tested our source code. We have carefully tuned the parameters of the baseline approaches and used them in their highest performing settings for comparison, but there may still exist errors that we did not note. Considering such cases,

we have published our source code and dataset to facilitate other researchers to replicate and extend our work.

External Validity The external validity relates to the quality and generalizability of our dataset. Our dataset is constructed from the official StackExchange data dump. We focus on three technical Q&A sites, i.e., Ask Ubuntu, Super User and Stack Overflow for our experiment. These three technical Q&A sites are commonly used by software developers and each one focuses on a specific area. However, there are still many other technical Q&A sites in StackExchange which are not considered in our study (e.g., Server Fault). We believe that our results will generalize to other technical Q&A sites as well, due to the ability of our approach to identify the best answer from a set of answer candidates. We will try to extend our approach to other technical Q&A sites to benefit more users in future studies.

Construct Validity The construct validity concerns the relation between theory and observation. In this study, such threats are mainly due to the suitability of our evaluation measures. For human evaluation, the subjectiveness of the evaluators, the evaluators' degree of carefulness, and the human errors may affect the validity of judgements. We minimized such threats by choosing experienced participants who have at least three years of studying/working experience in the software development process, and are familiar with Ubuntu system and Python programming languages. We also gave the participants enough time to complete the evaluation tasks.

Model Validity The model validity relates to model structure that could affect the learning performance of our approach. In this study, for the answer recommendation task, we choose a CNN-based model due to the optimum results achieved by [Kim, 2014]. Recent studies [Lai et al., 2015, Zhou et al., 2016] have shown that the RNN-based model can also achieve promising performance on the text classification task, which is similar to ours. For the question boosting task, we use the vanilla sequence-to-sequence model. Recent

Outlier Examples	Q (Ask Ubuntu): How to install SiS 671/771 Video Drivers in ubuntu?	A(Positive): This is an easy how-to: http://sites.google.com/site/easylinuxtipsproject/sis
		A(Neutral+): The display driver for sis would be already installed in Ubuntu. xserver-xorg-video-sis is the display driver for all Sis and XGI video driver. ...
	Q (SO Python): How can I create a regular expression in Python?	A(Positive): Try something like this: <code>r'[a-zA-Z0-9]+_[^_]+_[a-zA-Z0-9]+\.[a-zA-Z0-9]+'.</code>
		A(Neutral-): https://www.debuggex.com is also pretty good. It's an online Python (and a couple more languages) debugger, which has a pretty neat visualization of what does and what doesn't match. A pretty good resource if you need to draft a regexp quickly.
	Q (Ask Ubuntu): How to share hotspot through SSH tunnel?	A(Neutral+): Oh, it works just as that. I didn't notice I have to check the rule specifying the table explicit: ...
		A(Neutral-): ... Here's two options that do work, though: 1. use sshuttle ... 2. set up OpenVPN on the remote system and your local system

Figure 4.11: Outlier Examples in Label Establishment

research has proposed new models, such as the pointer-generator [See et al., 2017], transformer [Vaswani et al., 2017] and bert [Devlin et al., 2018]. However, our results do not shed light on the effectiveness of employing other deep learning models with respect to different structures and new advanced features. We will try to use other deep learning models for our tasks in future work and compare them to those we report in this paper.

4.6.3 Outlier Cases Study

As detailed in Section 4.2.2, we build our training samples via four heuristic rules, we thus can not ensure that there are no outlier cases distant from our heuristic rules. The outlier cases will produce a series of wrong preference pairs and hinder the learning performance of our model. Fig. 4.11 shows three outlier examples for label establishment. From the figure we can see that:

1. From the first example, it can be seen that, the quality of its non-accept answer in terms of informativeness and relevance are better than the accepted ones, not to mention that the link provided within the *Positive* sample has been not available. This shows the outlier case that the non-accept answers may be better than the accepted

answers.

2. From the second example, it can be seen that, for a given question, the answers from its similar questions are more descriptive than its own. This shows the outlier cases that the answers of other questions may be better than its own.
3. From the last example, it can be seen that, the answers from its similar questions may provide more information cues than its non-accept answers.

Detecting and removing these outlier cases before building the training samples will benefit the learning performance of our proposed DEEPANS model, we will focus on this research direction in the future.

4.7 Related Work

In this section, we describe the related studies on best answer retrieval, query expansion in software engineering, and deep learning in software engineering.

4.7.1 Best Answer Retrieval

Great effort has been dedicated to addressing the question answering tasks on Q&A sites [Adamic et al., 2008, Tian et al., 2013a, Calefato et al., 2019, Zhang et al., 2007, Jenders et al., 2016, Calefato et al., 2016, Sahu et al., 2016, Nie et al., 2017]. Conventional techniques for retrieving answers primarily focus on complementary features of the Q&A sites. For example, Adamic et al. [Adamic et al., 2008] reported the first study on best answer prediction in Yahoo! Answers using user-related features. Following Adamic et al.'s study, Tian et al. [Tian et al., 2013a] trained a classifier on a dataset from Stack Overflow without relying on user-related features. Recently, Calefato et al. [Calefato et al., 2019] modelled the answer prediction task as a binary classification problem, they assessed 26 best answer prediction model in Stack Overflow. Different from these works, we present

a novel weakly supervised neural network architecture for ranking answers for a given question. To the best of our knowledge, our work is the first to apply deep neural network to the specific problem of best answer selection in Q&A sites. Our approach can not only identify best answers from a list of candidate answers, but also recommend the most relevant answers for these unanswered posts. Besides, we also compare with Calefato et al.'s [Calefato et al., 2019] approach, and the experimental results have shown that the improvement is substantial.

4.7.2 Query Expansion in SE

Query expansion has long been investigated as a way to improve the results returned by a search engine [Haiduc et al., 2013, Hill et al., 2014, Lu et al., 2015, Xu et al., 2017, Rao and Daumé III, 2018, Nie et al., 2016, Li et al., 2016, Azizan and Bakar, 2015, Huang et al., 2018]. Some software engineering researchers have employed query expansion to improve the performance of tasks such as code search, answer summary, and similar question recommendation. For example, Haiduc et al. [Haiduc et al., 2013] proposed an approach that can recommend a good query reformulation strategy by performing machine learning on a set of historical queries and relevant results. Hill et al. [Hill et al., 2014] proposed a query expansion tool named Conquer, which introduces a novel natural language based approach to organize and present search results and suggest alternative query words. More recently, Lu et al. [Lu et al., 2015] presented an approach to expand the original query with synonyms from WordNet, which can help developers to quickly reformulate a better query. Xu et al. [Xu et al., 2017] proposed a novel framework to reformulate the answer in Stack Overflow to reduce the lexical gap between question and answer sentences. Inspired by these studies we also leverage the idea of query expansion to recommend the relevant answers. Our *DeepAns* tool generates useful clarifying questions as a way of query boosting, which can substantially reduce the lexical gap between the question and answer sentences.

In contrast, all of the aforementioned studies ignore the interactions between the asker and the potential helper.

4.7.3 Deep Learning in SE

Recently, an interesting direction in software engineering is to use deep learning to solve many diverse software engineering tasks [White et al., 2019, Gu et al., 2018, Hu et al., 2018, Gao et al., 2020, Li et al., 2017, Huang et al., 2020, Yang et al., 2015, Gu et al., 2016b, Sun et al., 2019, Alahmadi et al., 2018, Kim et al., 2019, Kim et al., 2018, Dong et al., 2019, Chen and Zhou, 2018, Liu et al., 2017, Gao et al., 2019b, Gao et al., 2020c]. For example, White et al. [White et al., 2019] leverage a deep learning approach, DeepRepair, for automatic program repairing. Gu et al. [Gu et al., 2018] propose a novel deep neural network named DeepCS for code search tasks, where code snippets semantically related to a query can be effectively retrieved. Hu et al. [Hu et al., 2018] develop a new sequence-to-sequence model named DeepCom to automatically generate code comments for Java methods. Li et al. [Li et al., 2017] present CCleaner which is a deep-learning based approach for clone detection.

Although the aforementioned studies have utilized deep learning techniques for different kinds of software engineering tasks, to our best knowledge, no one has yet considered the relevant answer recommendation task in technical Q&A sites. We proposed in this paper a novel neural network architecture to address the answer hungry problems in technical Q&A forums.

4.8 Summary

To alleviate the *answer hungry* problem in technical Q&A sites, we have presented a novel neural network-based tool, DEEPANS, to identify the most relevant answer among a set of answer candidates. Our model follows a three-stage process: *question boosting*, *label estab-*

lishing and *answer recommendation*. Given a post, we first generate a clarifying question as a way of question boosting, we then automatically generate *positive*, *neutral*⁺, *neutral*⁻ and *negative* training samples via label establishing. Finally based on the four kinds of training samples we generated, we trained a weakly-supervised neural network to compute the matching score between the question and candidate answers. Extensive experiments on the real-world technical Q&A sites have comparatively demonstrated the promising performance and the robustness of our approach in solving unanswered/unresolved questions.

Chapter 5

Que2Code: Searching Code Solutions from Stack Overflow Posts

Gao, Z., Xia, X., Lo, D., Grundy, J.C., Zhang, X., Xing, Z., I Know What You Are Searching For: Code Snippet Recommendation from Stack Overflow Posts, submitted to ACM Transactions on Software Engineering and Methodology, under major revision.

5.1 Introduction

To deliver high-quality software more effectively and efficiently, many developers frequently use community Question and Answer (Q&A) sites, such as Stack Overflow, for solutions to their programming problems and tasks [Xu et al., 2017]. Code search is one such task that plays an important role in software development. Software developers spend about 19% of their development time searching for relevant code snippets on the web [Brandt et al., 2009]. To help in programming problem solving, searching for useful code snippets from Stack Overflow has become a common part of developer's daily work [Xu et al., 2017]. Typically, when developers encounter a technical problem, they formulate the problem as a query and use a search engine to obtain a list of possible relevant posts that may contain useful solutions to their problem. After that, developers have to read answers with various levels of quality included in the returned posts to identify the possible solutions.

This kind of solution-seeking experience can be difficult and painful with respect to the

following two kinds of phenomena:

- *Query Mismatch*. Although search engines (e.g., Google) are widely used for searching required information on the web, there are certain user needs that can not be satisfied. Considering the question queries formulated by developers are highly technical, context-specific and/or subjective, these queries may be semantically related but not share lexical units. This *query mismatch* phenomena can lead to the “*discovery deficit*” problem when using general-purpose search engines. This is because no obvious answers relate to the specific problem, which decreases success rate when using standard information retrieval tools for code search tasks.
- *Information Overload*. Due to the huge amount of posts and content of most online Q&A systems software developers commonly experience *information overload*. This phenomenon can lead to the “*filter failure*” problem, that is developers become flooded with too many search results. Xu et al. [Xu et al., 2017] conducted a survey with 72 developers, according to their interview and survey-based study, there is too much noisy and redundant information online, and developers often wasted their time on reading the irrelevant posts, which was really time-consuming. Moreover, developers may fail to filter out irrelevant results and tend to get lost in the massive amount of information they encounter. Therefore, there is a need by developers to only receive the most suitable code snippets to their current programming tasks.

Stack Overflow contains a large number of code snippets embedded in posts, as both questions and solution examples. Due to this large volume, there is a high chance that code snippets related to a developer’s query already existed in the Stack Overflow posts. A major challenge is whether these code snippets can be retrieved successfully and effectively based on developer queries relating to their current programming task. In this work, we aim to help developers who want to quickly identify useful code snippets for their tech-

nical queries, without spending too much time on browsing through and/or understanding irrelevant examples. To do this, we have developed an automated technique to search for the best code fragments in Stack Overflow that most closely match the developer’s intent. This automated approach needs to be able to prioritize or rank code snippets according to developers’ specific technical questions, so more relevant and useful code snippets are ranked higher than the less useful and irrelevant ones.

We formulate this task as a *query-driven code recommendation* task for a given input question to Stack Overflow. Given an input question, instead of naively choosing the answers from relevant questions, we present a novel model and tool, named QUE2CODE, to achieve this goal of searching for the best code snippet to answer a user query. We use a two-stage model: in the first stage, we use a query rewriter to tackle the *query mismatch* challenge. The idea is to use rewritten version of a query question to cover different forms of semantically equivalent expressions. In the second stage, we use a code selector to tackle the *information overload* challenge. We extract all the code snippets from the collected answers to construct a candidate pool, and then train a Pairwise Learning to Rank neural network by automatically establishing positive and negative training samples. We then select the best code snippet from the code snippet candidates via pairwise comparisons. We conduct extensive experiments to evaluate our QUE2CODE model. To evaluate the first stage of semantically-equivalent question retrieval, we collect duplicate question pairs of Python and Java from Stack Overflow and verify the effectiveness of our approach for identifying the semantically-equivalent question for a given user query question. To evaluate the second stage of best code snippet recommendation, we collect more than 218K QC (question-code snippet) pairs for Python and more than 270K QC pairs for Java. We then evaluate the effectiveness of our approach for choosing the right code snippet in the code snippet candidate pool. The automatic experimental results show that our proposed QUE2CODE model outperforms several state-of-the-art baselines in both stages, and this

demonstrates the superiority of our model.

This paper makes the following three main contributions:

- All previous studies of question routing in CQA systems [Wang et al., 2009, Cao et al., 2010, Ganguly and Jones, 2015, Ye et al., 2014, Zou et al., 2015, Xu et al., 2018] work on finding similar questions. However, it is hard to measure the relevance between different questions automatically and experts are often asked to manually rate the relevance score [Xu et al., 2017, Xu et al., 2018]. In our study, we propose a new task of semantically-equivalent question retrieval. By utilizing duplicate question pairs archived in Stack Overflow, we present a novel model and an evaluation method to automatically evaluate the semantically-equivalent questions without a labor-intensive labeling process.
- All current studies that have investigated code snippet searching [Gu et al., 2018, Cambronero et al., 2019, Sachdev et al., 2018, Ye et al., 2016] rely on calculating a matching score between a query and a code snippet. We argue that code snippet recommendation is more about predicting relative orders rather than precise relevance scores. Hence, we propose a novel pairwise learning to rank model to recommend code snippet from Stack Overflow posts, and we first use the BERT model for searching for code snippets in Stack Overflow.
- Our experimental results show that our QUE2CODE is more effective for code snippet recommendation than several state-of-the-art baselines. We have released the source code of our QUE2CODE and our dataset¹ to help other researchers replicate and extend our study.

The rest of the paper is organized as follows. Section 5.2 presents a motivating example and user scenario of our approach. Section 5.3 presents details of our approach

¹<https://github.com/beyondacm/Que2Code>

▲
▼
2

This question already has answers here:
Closed 11 years ago.

So I'm using python to do some parsing of web pages and I want to split the full web address into two parts. Say I have the address <http://www.stackoverflow.com/questions/ask>. I would need the protocol and domain (e.g. <http://www.stackoverflow.com>) and the path (e.g. </questions/ask>). I figured this might be solved by some regex, however I'm not so handy with that. Any suggestions?

python string url split

▲
▼
14

Use the [urlparse](#) module. Check this function:

```
import urlparse

def process_url(url, keep_params=('CONTENT_ITEM_ID=',)):
    parsed= urlparse.urlsplit(url)
    filtered_query= '&'.join(
        qry_item
        for qry_item in parsed.query.split('&')
        if qry_item.startswith(keep_params))
    return urlparse.urlunsplit(parsed[:3] + (filtered_query,) + parsed[4:])
```

Figure 5.1: Motivating Example of Query Mismatch

for semantically-equivalent question retrieval and best code snippet recommendation. Section 5.4 presents the automatic experimental results of our approach with respect to the two stages separately. Section 5.5 presents the results of our approach on human evaluation. Section 5.7 discusses the strengths and of our approach and threads to validity. Section 5.8 presents key related work associated with our study. Section 5.9 concludes the paper.

5.2 Motivation

In this section, we first show two motivating examples from Stack Overflow of the sorts of problems mentioned above (i.e., *query mismatch* and *information overload*), we then present the user scenarios of employing our approach which can help developers to address these problems.

Consider Fig. 5.1 and Fig. 5.2 which illustrates the *query mismatch* and *information overload* problem in Stack Overflow respectively. For example, as shown in Fig. 5.1, the

D
How do I calculate the derivative of an input-function?
 [duplicate] (44390259)

M
How do I compute derivative using Numpy?

<https://stackoverflow.com/questions/how-do-i-calcul...>

[How do I calculate the derivative of an input-function? - Stack ...](#)
 1 answer
 6 June 2017 — From the scipy docs: >>> from scipy.misc import derivative >>> def f(x): ...
 return x**3 + x**2 ... >>> derivative(f, 1.0, dx=1e-6) ...
 You visited this page on 13/06/21.

<https://stackoverflow.com/questions/derivative-of-n...>

[Derivative of neural network with respect to input - Stack ...](#)
 26 June 2019 — I trained a neural network to do a regression on the sine function and would
 like to compute the first and second derivative with respect to the ...
 3 answers · Top answer: One possible explanation for what you observed, could be that your f...

<https://stackoverflow.com/questions/keras-calculatin...>

[Keras: calculating derivatives of model output wrt input returns ...](#)
 16 Mar 2018 — The two parameters are Tensors. The gradients function returns None. Tensor("dense_2/Softmax0", shape=(?, 10), dtype= ...
 1 answer · Top answer: You are computing the gradients respect to X_train, which is not an in...

<https://stackoverflow.com/questions/how-do-you-ev...>

[How do you evaluate a derivative in python? - Stack Overflow](#)
 30 May 2017 — Your function fprime is not the derivative. It is a function that returns the
 derivative (as a Sympy expression). To evaluate it, you can use .subs to ...
 3 answers · Top answer: The answer to this question is pretty simple. Sure, the subs option giv...

<https://stackoverflow.com/questions/calculate-the-pa...>

[Calculate the partial derivative of the loss with respect to the ...](#)
 8 Nov 2017 — Calculate the partial derivative of the loss with respect to the input of the ... The
 formula is the backpropagation error signal through the matrix multiplication. ... Note that it
 doesn't depend on the loss function, only on the error ...
 1 answer · Top answer: It's as simple as def dense_grad_input(x_input, grad_output, W, b): ret...

<https://stackoverflow.com/questions/how-do-i-comp...>

[How do I compute derivative using Numpy? - Stack Overflow](#)
 25 Sept 2014 — SymPy is an excellent project for this that integrates well with NumPy. Look at
 the autowrap or lambdify functions or check out Jensen's blogpost ...
 8 answers · Top answer: You have four options • Finite Differences • Automatic Derivatives • S...
 Missing: input- | Must include: input-

Figure 5.2: Motivating Example of Information Overload

objective is to split a URL using Python. The code snippet associated with the target question (marked with ‘M’) can satisfy the problem of the user’s query question (marked with ‘D’). However, since the user query question and the target question do not share any lexical units, it is very difficult, if not impossible, to retrieve the target question post solely based on generic, lexically-based search engines. Another example is shown in Fig. 5.2, even though the duplicate question pairs share some common words, searching the user query, i.e., “*how do I calculate the derivative of an input-function?*”, with Google search engine still returns a large volume of low-quality and/or irrelevant posts. Each post often includes multiple answers with many not useful code examples. As shown in Fig. 5.2, Even if the search engine successfully returns a target post, if the target post is not ranked high in the results, the potential solution to the programming task can be easily buried in all this overwhelming amount of information.

We illustrate the usage scenario of our proposed tool, QUE2CODE, as follows:

Without Our Tool: Consider Bob is a developer. Daily, Bob encounters a technical problem and wants a code snippet to help solve it. He tries his best to write a query to summarize his problem and searches related questions on Stack Overflow. However, due to lack of the knowledge and terminology about the problem, the query formulated by Bob does not match any post with potential answers. Furthermore, Bob has to painstakingly browse a lot of low-quality and/or irrelevant posts to identify any possible solutions. Therefore, Bob loses interest and is unsatisfied with the overwhelming number of seemingly irrelevant posts. As a result, he posts a duplicate question on Stack Overflow.

With Our Tool: Now consider Bob adopts our QUE2CODE. When Bob types in his query question, our *QueryRewriter* first generates a list of paraphrase questions for his problem, which increases the likelihood of retrieving semantically-equivalent questions in Stack Overflow. Following that, instead of naively returning a massive amount of similar questions, our *CodeSelector* sorts the returned code snippet candidates and recommends

the most relevant ones that may contain possible solutions. With the help of our tool, Bob can quickly get answers for his problem without spending much time on reviewing and digesting the low-quality and/or irrelevant information. This time, Bob successfully identifies a useful code snippet from a Stack Overflow post for his problem by using our tool.

5.3 Approach

We present a novel query-driven code recommendation system. QUE2CODE consists of two stages: *Semantically-Equivalent Question Retrieval* and *Best Code Snippet Selection*. Our approach takes in a technical question as a query from a developer, and recommends a sorted list of code snippets.

5.3.1 Overview of Approach

Fig. 5.3 demonstrates the workflow of QUE2CODE. Our model contains two stages: (i) semantically-equivalent question retrieval and (ii) best code snippet recommendation. It has two sub-components, i.e., *QueryRewriter* and *CodeSelector*. The first can qualitatively retrieve semantically-equivalent questions, and the second can quantitatively rank the most relevant code snippets to the top of the recommendation candidates.

In the first stage, our *QueryRewriter* component tackles the *query mismatch* challenge. To bridge the gap between different expressions of semantically-equivalent questions, we introduce the idea of *query rewriting*. The idea is to use a rewritten version of a query question to cover a variety of different forms of semantically equivalent expressions. In particular, we first collect the duplicate question pairs from Stack Overflow, because duplicate questions can be considered as semantically-equivalent questions of various user descriptions. We then frame this problem as a sequence-to-sequence learning problem, which directly maps a technical question to its corresponding duplicate question. We train

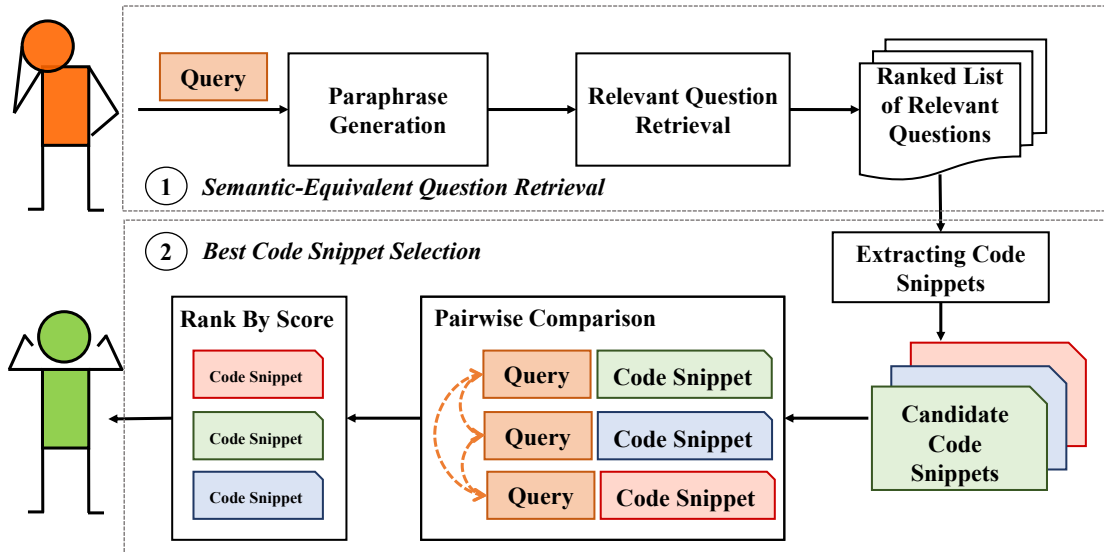


Figure 5.3: Workflow of Que2Code

a text-to-text transformer, named *QueryRewriter*, by using the collected duplicate question pairs. After the training process, for any given query question, *QueryRewriter* outputs semantically equivalent paraphrased questions of the input query. Following that, the query question and its generated paraphrased questions are encoded by *QueryRewriter* to measure their relevance with other question titles.

In the second stage, our *CodeSelector* component tackles the *information overload* challenge. To do this, we first collect all the answers of the semantic relevant questions retrieved in the first stage. We then extract all the code snippets from the collected answer posts to construct a candidate code snippets pool. For the given query question, we pair it with each of the code snippet candidates. We then fit them into the trained *CodeSelector* to estimate their matching scores and judge the preference orders. *CodeSelector* can then select the best code snippet from the code snippet candidates via pairwise comparison. Our approach is fully data-driven and does not rely on hand-crafted rules.

5.3.2 Semantically-Equivalent Question Retrieval

In this stage, given a technical problem formulated as a query, we propose a *QueryRewriter* to generate paraphrase questions and retrieve the semantically-equivalent questions in Stack Overflow. Fig. 5.4 demonstrates the workflow of *QueryRewriter*. *QueryRewriter* has three steps: paraphrase generation, question embedding and questions retrieval.

Paraphrase Generation To obtain the features of semantically-equivalent questions for a given user query, we utilize the historical archives of duplicate questions in Stack Overflow, which are manually marked by users and moderators. These duplicate question pairs can be viewed as questions of same intent but written in different ways. In this step, we first automatically generate paraphrase questions for the query question to represent different forms of user expressions. The underlying idea is that by adding these paraphrase questions, we are more likely to find the relevant question that match the intent expressed in the user query.

In this step, we model the task of paraphrase question generation as a sequence-to-sequence learning problem, where the question title is viewed as a sequence of tokens and its corresponding duplicate question as another sequence of tokens. We adopt the Seq2Seq Transformer architecture [Vaswani et al., 2017], which includes an Encoder Transformer and a Decoder Transformer. Both the Encoder and the Decoder Transformers have multiple layers and each layer contains a multi-head attentive sub-layers followed by a fully connected sub-layer with residual connections [He et al., 2016] and layer normalization [Ba et al., 2016].

More formally, given a question \mathbf{X} as a sequence of tokens (x_1, x_2, \dots, x_M) of length M , and its duplicate question \mathbf{Y} as a sequence of tokens (y_1, y_2, \dots, y_N) of length N . The encoder takes the question \mathbf{X} as input and transforms it to its contextual representations. The decoder learns to generate the corresponding duplicate question \mathbf{Y} one token at a time based on the contextual representations and all preceding tokens that have been generated

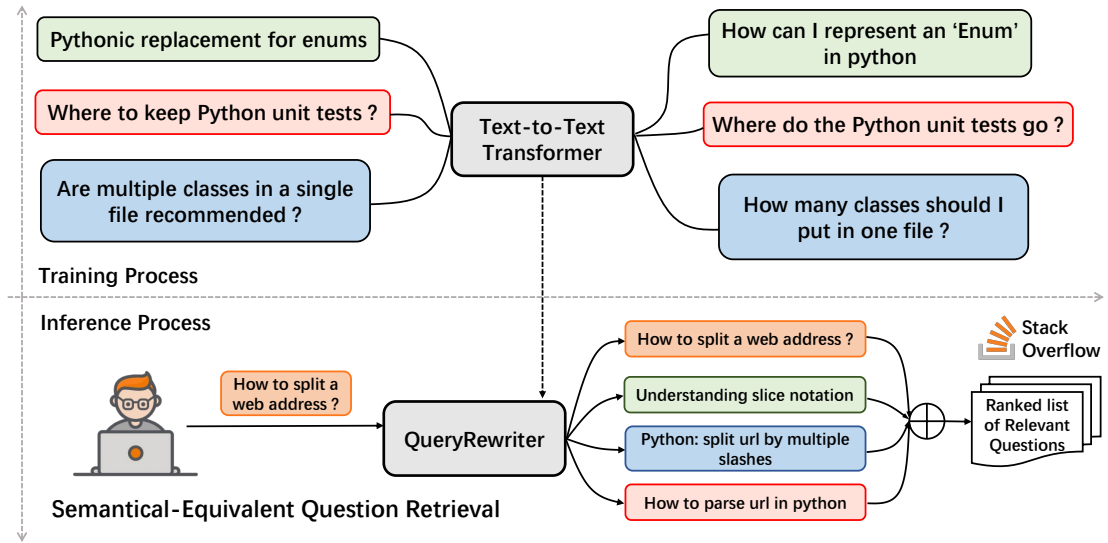


Figure 5.4: QueryRewriter Workflow

so far. Mathematically, the paraphrase generation task is defined as finding \bar{y} , such that:

$$\bar{y} = \operatorname{argmax}_{\mathbf{Y}} P_{\theta}(\mathbf{Y}|\mathbf{X}) \quad (5.1)$$

where $P_{\theta}(\mathbf{Y}|\mathbf{X})$ is defined as:

$$P_{\theta}(\mathbf{Y}|\mathbf{X}) = \prod_{i=1}^L P_{\theta}(y_i|y_1, \dots, y_{i-1}; x_1, \dots, x_M) \quad (5.2)$$

$P_{\theta}(\mathbf{Y}|\mathbf{X})$ can be seen as the conditional log-likelihood of the predicted duplicate question \mathbf{Y} given the input question \mathbf{X} . This model can be trained by minimizing the negative log-likelihood of the training question duplicate question pairs. Once the model is trained, we do inference using beam search [Koehn, 2004]. Beam Search returns a list of most likely output sequences (i.e., paraphrase questions). It searches question tokens produced at each step one by one. At each time step, it selects b tokens with the least cost, where b is the beam wise. It then prunes off the remaining branches and continues selecting the possible tokens that follow on until it meets the end-of-sequence symbol. We repeat the process and generate the top-N most likely paraphrase questions for our study.

A working example is demonstrated in Fig. 5.4, when we input the user query “*how to split a web address?*” described in the motivation example into our trained model, the

QueryRewriter automatically generates paraphrase questions of different expressions, such as “understanding slice notation”, “python: split url by multiple slashes”, “how to parse url in python”. Since the user query question (i.e., “how to split a web address”) and the target question (i.e., “slicing url with python”) do not share any lexical units, incorporating the aforementioned paraphrase questions can successfully bridge this gap and solve the *query mismatch* problem.

Question Embedding After the text-to-text transformer is trained, we can generate multiple paraphrase questions for a newly posted user query. We use these paraphrase questions to boost the user query for the downstream task of question retrieval. By incorporating the paraphrase questions, we can alleviate the *query mismatch* problem by covering the different forms of duplicate question expressions. Thus, we have a better chance to retrieve semantically-equivalent questions in Stack Overflow that are intended to solve the same problem.

For a given user query question q_u , we first construct $\mathbf{Q}_u = \{q_u, pq_k\} (1 \leq k \leq N)$, where q_u is the user query question and $pq_k (1 \leq k \leq N)$ are the top- N generated paraphrase questions. We investigated a wide range of N values (i.e., from 1 to 20) for measuring our model’s performance, setting N to 5 is close to the optimal settings for our approach. To capture the overall features and semantics of the user query q_u we embed each question q_i in \mathbf{Q}_u (including the user query question and the paraphrase questions) to a fixed dimensional vector e_{q_i} via the Encoder Transformer. We then use the average embeddings of all possible questions as the final representation for the user query question. More formally, the question embedding step is defined as follows:

$$e_{q_i} = \text{Encoder}(q_i), q_i \in \mathbf{Q}_u \quad (5.3)$$

$$e_{q_u} = \frac{1}{N} \sum e_{q_i}, q_i \in \mathbf{Q}_u \quad (5.4)$$

Question Retrieval Given a newly posted user query question q_u and a question title q

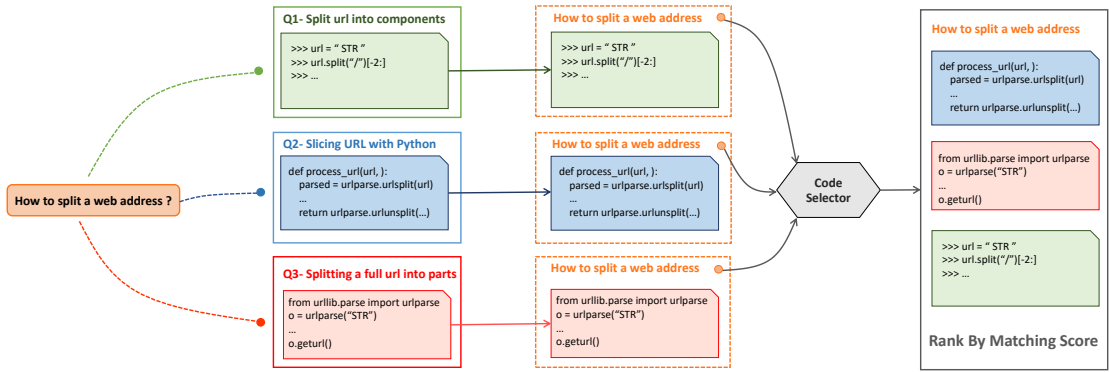


Figure 5.5: CodeSelector Workflow

in the Stack Overflow repository, we define the semantic distance as well as the relevance between two questions as below:

$$Distance(q_u, q) = \frac{Euclidean(e_{q_u}, e_q)}{|e_{q_u}| + |e_q|} \quad (5.5)$$

$$Relevance(q_u, q) = 1 - Distance(q_u, q) \quad (5.6)$$

For a given user query, we can easily compute a relevance score between the user query and any candidate question in our database. Following that, all the relevance scores are sorted and the top-K ranked questions are returned as the most semantically-relevant questions for the query. In this study, we selected different K values (i.e., from 5 to 10) to investigate the robustness of our model. In practice, we recommend setting K around the above range, which can cover a wide range of code snippets as well as maintain a good performance.

5.3.3 Best Code Snippet Recommendation

Theoretically, after retrieving the semantically-equivalent questions for a given user query, we can naively choose the code snippet from the top ranked questions and recommend the solution to the developers. However, we argue this is not sufficient regarding the following reasons: (i) The technical queries submitted by developers are complicated or sometimes

inaccurate [Gao et al., 2020a], there is no guarantee that the top ranked solution is correct and can satisfy the needs of developers. Therefore, we need to collect as many as possible relevant potential code snippet candidates. (ii) Although the search engine can return a list of relevant questions to their problems, the large number of relevant posts and the sheer amount of information in them makes it difficult for developers to find the most needed answer [Xu et al., 2017]. Therefore, how to pick the best solution from the massive amounts of information is a non-trivial task.

To address this time-consuming task of online code searching, we propose *CodeSelector* to help developers effectively select the most relevant and suitable code snippet for a specific query question. Particularly, the *CodeSelector* reranks all the code snippets by comparing the matching score among different QC (query-code) pairs. Snippets ranked in the top of the final result indicate that these code snippets are more likely and suitable for the programming task. Fig. 5.5 demonstrates the workflow of our *CodeSelector*. For a given query question, a list of relevant questions are retrieved from stage one. After that, all the code snippets associated with these questions are collected, and each code snippet is paired with the given query to make a QC pair. Then the *CodeSelector* reranks all the code snippet candidates by conducting pairwise comparison among QC pairs. To build the *CodeSelector*, two steps are performed: Preference Pairs Construction and Pairwise Comparison.

Preference Pairs Construction In this step, we use the available crowdsourced data on Stack Overflow for preference pairs construction. We propose three heuristic rules that can automatically establish the preference pairs and construct the training sets. Our approach is fully data driven and it does not need manual effort. Our three heuristic rules are as follows:

- *For a given question, its best code snippet is preferable to a non-relevant code snippet.* We define the *best code snippet* for a question as the code snippet associated with an accepted answer to the question, or the one associated with the highest-vote

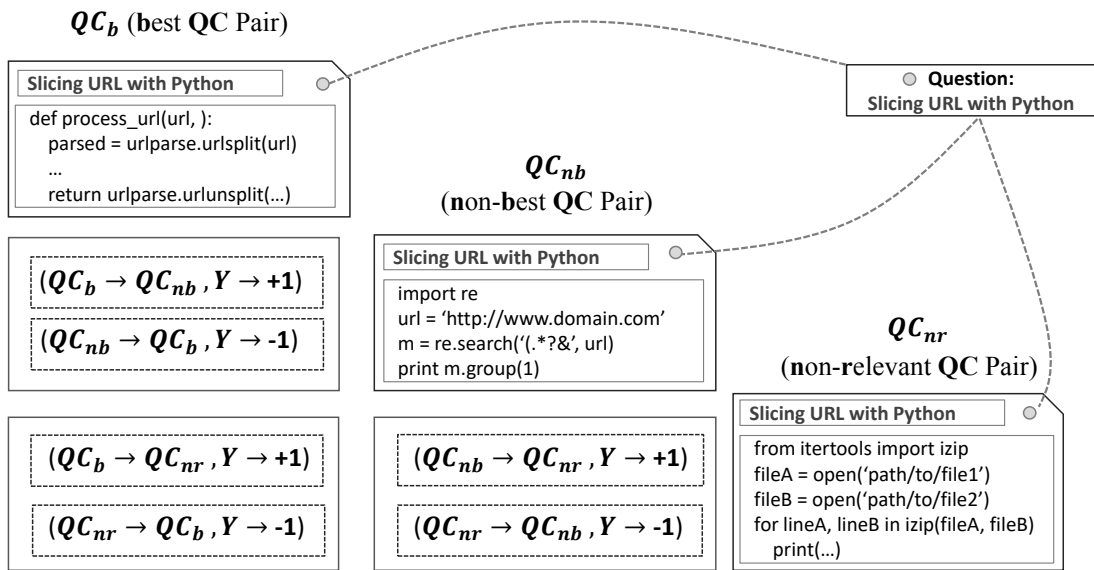


Figure 5.6: Preference-Pairs Construction

answer if there are multiple answers to the question. A non-relevant code snippet is randomly selected from the repository. This rule suggests that the quality of the best code snippet is better than the non-relevant ones.

- For a given question, its non-best code snippet is preferable to a non-relevant code snippet. We define the *non-best code snippet* as other code snippets apart from the best one within the same question thread. This rule suggests that a question prefers the code snippets associated with answers to itself to those of others.
- For a given question, its best code snippet is preferable to its non-best code snippet. Even though an individual user vote may not be very reliable, the aggregation of a great number of user votes can provide a very powerful indicator of relevance preference. We argue that the *best code snippet* from an answer post for a question is better than the *non-best ones* in different answer posts for the same question, in most cases.

According to the above heuristic rules, for each given question, three query-code (QC) pairs can be generated: we pair it with its best code snippet as the QC_b (**best QC pair**), we pair it with its non-best code snippet as the QC_{nb} (**non-best QC pair**), we pair it with a non-

relevant code snippet as the QC_{nr} (**n**on-**r**elevant **Q**C pair). We then automatically construct the training samples $\langle QC_1, QC_2, Y \rangle$ for this study. In particular, a training sample contains three parts: two QC pairs (i.e., QC_1 and QC_2) and a label (i.e., Y), the label is automatically determined by the preference relationship between the two QC pairs. Fig. 5.6 demonstrates our data labeling process. Given the query question “*Slicing URL with Python*”, we first make three QC pairs, i.e., QC_b , QC_{nb} and QC_{nr} as mentioned above. Then a Pairwise comparison between any two QC pairs can establish a label Y for this training sample. It is worth emphasizing that the comparison order of the two QC pairs matters. For example, a comparison between QC_b and QC_{nb} will be labelled as positive, while a comparison between QC_{nb} and QC_b will be labelled as negative.

There are several advantages of employing this data labeling process: (i) due to the professionalism of technical queries, only experts with domain knowledge are qualified to judge the usefulness of a code snippet to a query question. Therefore, manually labeling the relevance scores for all code snippets is very time-consuming and requires a substantial effort [Jiang et al., 2016, Gao et al., 2020b]. Our heuristic rules can automate the labeling process without any human efforts. (ii) By using these heuristic rules, we gather more training samples, which can provide enough data points for training a deep learning based model.

Pairwise Comparison

After collecting large amounts of labeled training data via preference pairs construction, we develop a learning-to-rank model to sort all the code snippets for a given query question. We design a novel semi-supervised network for ranking query-code (QC) pairs. Fig 5.7 demonstrates the architecture of our proposed model. The input to *CodeSelector* are two QC pairs. *CodeSelector* then learns to judge the preference relationship between two QC pairs based on the positive and negative training samples. In other words, we not only consider the program semantics between a query and a code snippet, but also investigate

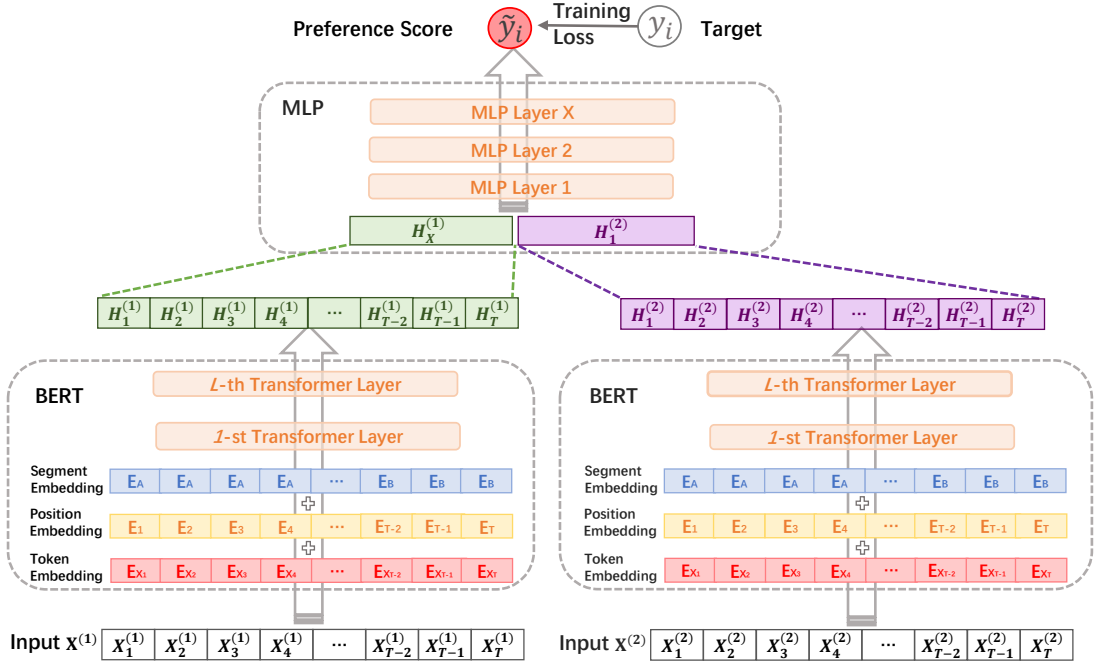


Figure 5.7: CodeSelector Workflow

the relevance preference among different QC pairs.

- BERT Embedding Layer.** We use the modeling power of BERT [Devlin et al., 2018], which is one of the most popular pre-trained models trained using Transformers [Vaswani et al., 2017]. BERT consists of 12-layer transformers, each of the transformers being composed of a self-attention sub-layer with multiple attention heads. Since BERT has been proven to be effective for capturing semantics and context information of sentences in other work, we use BERT as the feature extractor for our task. The input to the BERT embedding layer are two parallel QC pairs. Given each QC pair as a sequence of tokens $\mathbf{x} = \{x_1, \dots, x_T\}$ of length T , BERT takes the tokens as input and calculate the contextualized representations $\mathbf{H}^l = \{h_1^l, \dots, h_T^l\} \in \mathbb{R}^{T \times D}$ as output, where l denotes the l -th transformer layer and D denotes the dimension of the representation vector. The underlying sub-components work in parallel, mapping each QC pair to its distributional vectors $\mathbf{h}^{(1)}$ and $\mathbf{h}^{(2)}$ respectively, which are then used to perform the predictions for the

downstream task.

- **Multi-Layer Perceptron.** After obtaining the BERT representations, we add a Multi-Layer Perceptron (MLP) on top of BERT embedding layer to calculate the preference score between the input two QC pairs. Since *CodeSelector* adopts BERT to model two QC pairs respectively, it is intuitive to combine the features of two pathways by concatenating them. This design has been widely adopted in other deep learning work [He et al., 2017, Gao et al., 2020b]. To further capture the preference between the latent features of $\mathbf{h}^{(1)}$ and $\mathbf{h}^{(2)}$, we add a standard MLP on the concatenated vector. In this sense, we can endow the model a large level of flexibility and non-linearity to learn the interactions between the two QC pairs. The contextualized representations (i.e., $\mathbf{h}^{(1)}$ and $\mathbf{h}^{(2)}$) are fed to the MLP layer to predict the final preference $\mathbf{y} = \{0, 1\}$. More precisely, the MLP is defined as follows:

$$\begin{aligned}
 \mathbf{z}_1 &= \phi_1(\mathbf{h}^{(1)}, \mathbf{h}^{(2)}) = \begin{bmatrix} \mathbf{h}^{(1)} \\ \mathbf{h}^{(2)} \end{bmatrix} \\
 \mathbf{z}_2 &= \phi_2(\mathbf{z}_1) = \mathbf{a}_2(\mathbf{W}_2^T \mathbf{z}_1 + \mathbf{b}_2) \\
 &\dots \\
 \mathbf{z}_L &= \phi_L(\mathbf{z}_{L-1}) = \mathbf{a}_L(\mathbf{W}_L^T \mathbf{z}_{L-1} + \mathbf{b}_L) \\
 P(\mathbf{y} = j | \mathbf{x}^{(1)}, \mathbf{x}^{(2)}) &= \sigma(\mathbf{z}_L)
 \end{aligned} \tag{5.7}$$

\mathbf{W}_x , \mathbf{b}_x , and \mathbf{a}_x denote the weight matrix, bias vector, and activation function for the x -layer’s perceptron respectively. σ is the sigmoid function $\sigma(x) = 1/(1 + e^{-x})$ which will output the final preference score between 0 and 1. For the preference score, we want this score to be high if the first QC pair is preferable to the second one (i.e., $\mathbf{x}^{(1)} \succ \mathbf{x}^{(2)}$), and to be low if the second QC pair is preferable to the first one (i.e., $\mathbf{x}^{(2)} \succ \mathbf{x}^{(1)}$).

5.3.4 Experimental Settings

We implemented our system in Python using the Pytorch framework. For the *QueryWriter*, we trained the text-to-text transformer on the duplicate question pairs, we follow the param-

eter settings from [Raffel et al., 2019], which has achieved state-of-the-art results on many benchmarks covering summarization, question answering, text classification, and more. For the *CodeSelector*, we use the pre-trained BERT model released by [Devlin et al., 2018] as our feature extractor. We use the ReLu as the activation function and employ three hidden layers for MLP. The size of the first hidden layer in MLP is equal to the size of the joint vector obtained after concatenating two QC vectors from the BERT model. We fix the parameters of the BERT model and fine tune the MLP parameters for our task, and the *CodeSelector* is learnt by optimizing the log loss of Equation. 5.7.

5.4 Automatic Evaluation

To recommend the code snippets for developers in Stack Overflow, our QUE2CODE is divided into two stage: semantically-equivalent question retrieval and best code snippet recommendation. We wanted to evaluate the performance of the proposed *QueryRewriter* to address the *query mismatch* problem in the first stage, and *CodeSelector* to address the *information overload* problem in the second stage. We want to answer the following key research questions:

- RQ-1: How effective is our *QueryRewriter* for semantically-equivalent question retrieval?
- RQ-2: How effective is our *QueryRewriter* for capturing the domain-specific contextual information?
- RQ-3: How effective is the paraphrase generation added to our *QueryRewriter*?
- RQ-4: What are the optimal parameter settings for our *QueryRewriter*?
- RQ-5: How effective is our *CodeSelector* for best code snippet selection?
- RQ-6: How effective is the BERT and preference pairs added to our *CodeSelector*?

- RQ-7: How robust is our *CodeSelector* with different parameter settings?

5.4.1 Effectiveness of QueryRewriter

We want to identify the best code snippet from a list of semantically-equivalent questions for a given query question. If the retrieved questions are not relevant to the query question, it is unlikely that our tool is able to find the suitable code snippets to solve the target problem. In this study, we consider the duplicate questions in Stack Overflow as semantically-equivalent question pairs. After training with the duplicate question pairs archived in Stack Overflow, *QueryRewriter* is able to generate paraphrase questions for a given user query question. By jointly embedding the user query question and the generated paraphrase questions, *QueryRewriter* retrieves the most relevant questions in the repository. We want to investigate the effectiveness of our *QueryRewriter* for retrieving semantically-equivalent questions in Stack Overflow.

Data Preparation We first downloaded the official data dump of Stack Overflow from the StackExchange² website. The raw data dump contains timestamped information about the *Posts*, *Comments*, *Users*, *Tags*, *Postlinks* etc.,. We extracted the duplicate question pairs as follows: We first parsed the *PostLinks* and *Posts* database files. Because duplicate questions are marked with a special marker in *PostLinks* database, we can easily identify the `PostId` of the source post and the target post if they are duplicate question pairs. After that, we extracted the question title of the source post and the target post by checking the `PostId` in *Posts* database. We regard the question title of the source post as a duplicate question and the question title of the target post as a master question. We then paired each master question q_m and duplicate question q_d as $\langle q_m, q_d \rangle$ pair. After that, these collected $\langle q_m, q_d \rangle$ pairs are fed into the text-to-text Transformer to train our *QueryRewriter*. In this study, we only focused on Python and Java programming languages for our experiment. As a result, we obtained more than 47K $\langle q_m, q_d \rangle$ pairs for Python and more than 56K pairs for

²<https://archive.org/download/stackexchange>

Table 5.1: Duplicate Questions Statistics

Python	# Duplicate Questions	47,170	# Avg. Tokens (Duplicate)	8.4
	# Master Questions	20,430	# Avg. Tokens (Master)	9.0
	# $\langle q_m, q_d \rangle$ Pairs (Train)	43,170	# Avg. Tokens (Intersect)	1.8
	# $\langle q_m, q_d \rangle$ Pairs (Val)	2,000	# $\langle q_m, q_d \rangle$ Pairs (Test)	2,000
Java	# Duplicate Questions	56,938	# Avg. Tokens (Duplicate)	8.5
	# Master Questions	23,889	# Avg. Tokens (Master)	8.6
	# $\langle q_m, q_d \rangle$ Pairs (Train)	52,938	# Avg. Tokens (Intersect)	1.7
	# $\langle q_m, q_d \rangle$ Pairs (Val)	2,000	# $\langle q_m, q_d \rangle$ Pairs (Test)	2,000

the Java dataset. We further investigated the *query mismatch* problem between the master questions and its corresponding duplicate questions. Specifically, we counted the number of tokens of the master question q_m and its duplicate question q_d , and then we counted the common lexical tokens between the master question and the duplicate question. The violin plot for Python and Java is demonstrated in Fig. 5.8, we can see that the overlap tokens between the master question and duplicate question are small. For example, for the Python dataset, the average number of tokens of the master question and duplicate question are 9.0 and 8.4 respectively, while the average number of overlap tokens is 1.8. Even though the master question and its duplicate question are semantic-equivalent, they only share a few tokens in common. This also justifies our assumption that the *query mismatch* problem frequently occurs when developers submit their search queries. Therefore, it is necessary to propose a model to address the *query mismatch* problem. Table 5.1 shows the statistics of our collected datasets of Python and Java duplicate questions. We randomly sampled 2,000 $\langle q_m, q_d \rangle$ pairs for validation and 2,000 $\langle q_m, q_d \rangle$ pairs for testing, and kept the rest for training. We clarify that different duplicate questions can point to the same master question and this is the reason why the number of duplicate questions is more than the master questions.

Experimental Setup To determine the effectiveness of our *QueryRewriter* for retrieving semantic-equivalent questions, we designed the following experiment: we first build

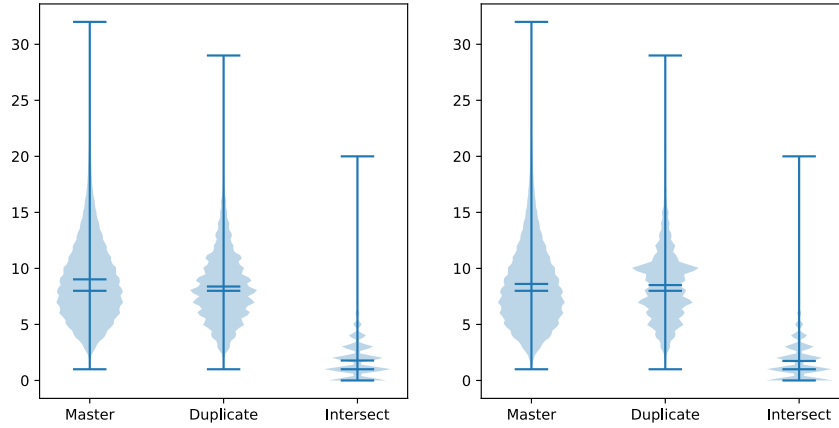


Figure 5.8: Violinplots of Question Distribution for Python(left) and Java(right) Dataset

an index with Lucene using all the question titles in testing set, the generated index is later used to retrieve the most relevant questions against a given query. In such a way, for each duplicate question q_d in the testing set, we first obtained a set of top-K similar questions $\mathcal{Q} = \{q_i\} (1 \leq i \leq k)$ via the Apache Lucene. We then added its corresponding master question q_m to \mathcal{Q} , where now $\mathcal{Q} = \{q_m, q_i\} (1 \leq i \leq k, q_m \neq q_i)$. For a given testing question q_d , we can ensure its corresponding master question q_m is in evaluation candidate pool \mathcal{Q} . Since \mathcal{Q} includes the semantic-equivalent question q_m for q_d . One way to evaluate our approach is to look at how often the master question q_m can be retrieved successfully among other members of \mathcal{Q} . Thus we adopted the metric, $P@K$ and $DCG@K$ [Manning et al., 2005] which are widely-used in previous studies [Xu et al., 2017, Jiang et al., 2016], to measure the ranking performance of the approach in our study. The evaluation metrics are defined as follows:

- $P@K$ is the precision of the master question in top-K candidate questions. Given a question, if one of the top-K ranked questions includes the master question, we consider the recommendation to be successful and set $success(q_m \in topK)$ to 1, otherwise, we consider the recommendation to be unsuccessful and set $success(q_m \in$

$topK$) to 0. The $P@K$ metric is defined as follows:

$$P@K = \frac{1}{N} \sum_{i=1}^N [success(q_m \in topK)] \quad (5.8)$$

- $DCG@K$ is another popular top-K accuracy metric that measures a recommender system performance based on the graded relevance of the recommended items and their positions in the candidate set. Different from $P@K$, the intuition of $DCG@K$ is that highly-ranked items are more important than low-ranked items. According to this metric, a recommender system gets a higher reward for ranking the correct answer at a higher position. The $success(q_m \in topK)$ is same with the previous definition, while the $rank_{q_m}$ is the ranking position of the master question q_m . The $DCG@K$ is defined as follows:

$$DCG@K = \frac{1}{N} \sum_{i=1}^N \frac{[success(q_m \in topK)]}{\log_2(1 + rank_{q_m})} \quad (5.9)$$

Experimental Baselines To demonstrate the effectiveness of our proposed method for relevant question retrieval task, we compared it to the following baselines:

- **IR** stands for the information retrieval baseline. For a given testing question q_i , it retrieves the question that is closest to q_i from the testing set. We use the traditional TF-IDF metric in our experiment, which is often used to calculate the relevance between a document and a user query in software engineering tasks, such as question retrieval [Xia et al., 2017, Cao et al., 2010] and code search [Sachdev et al., 2018].
- **Word2Vec** is a model that embeds words in a high dimensional vector space using a shallow neural network [Mikolov et al., 2013]. This word embedding technique has provided a strong baseline for information retrieval tasks. Yang et al. [Yang et al., 2016] used average word embeddings of words in a document as the vector representation for a document. The average word embeddings can be used to calculate the relevance between two question titles in our task.

- **FastText** is another word embedding model proposed by [Bojanowski et al., 2017]. Similar to the Word2Vec model, each word is represented as a high dimensional vector in such a way that similar words have similar vector representations. Same with the Word2Vec baseline, the average FastText word embeddings are used to estimate the similarity between different question titles.
- **Sent2Vec** Sent2Vec method, which is also known as para2vec or sentence embedding [Le and Mikolov, 2014]. This method modifies the Word2Vec algorithm to generate semantic embeddings of longer pieces of text (e.g., sentences or paragraphs) via unsupervised learning. The generated sentence embeddings have been applied in textual similarity tasks [Le and Mikolov, 2014]. With the help of publicly accessible tool [Řehůřek and Sojka, 2010], we train the Sent2Vec model using the duplicate question corpus and obtain the sentence embeddings for each question title.
- **AnswerBot** Xu et al. [Xu et al., 2017] proposed a three-stage framework called AnswerBot to generate an answer summary for a non-factoid technical question (i.e., non-factoid questions are defined as open-ended questions that require complex answers, like descriptions, opinions, or explanations, and technical questions are often non-factoid questions. [Song et al., 2017, Hashemi et al., 2020, Xu et al., 2017]). In their first stage, they combined the word embeddings with the traditional traditional IDF metrics for retrieving relevant questions. Their method has been proven to be effective in the task of relevant question retrieval compared with a set of baselines.
- **CROKAGE** More recently, Da et al. [da Silva et al., 2020] proposed a model named CROKAGE to recommend relevant solutions from Stack Overflow for a searching query. They aimed to address the lexical gap problem between the query and the solutions via using a multi-factor relevance mechanism. To be more specific, they

calculated the final relevance score by combining four types of scores (*lexical score*, *semantic score*, *API method score*, *API class score*). We adapt their approach for our task of retrieving semantically-equivalent questions. Particularly, we only retain the *lexical scores* and *semantic scores* for this research question since question titles usually don't contain API classes.

Given a question, we would like to investigate whether an approach can rank its duplicate question higher up among other question candidates. Intuitively, Google search engine can be considered as a baseline for searching duplicate questions. Nevertheless, we did not choose Google search engine baseline for this research question because all the duplicate question pairs have already been seen and recorded by Google search engine. If an existing question has been manually labeled as duplicate, it is more likely for Google search engine to retrieve its duplicate question. Therefore, It is not fair to compare Google search engine with our approach and other baselines, because our approach and other baseline never see the testing data set while Google search engine does.

Table 5.2: Effectiveness evaluation (Python)

Model	P@1	P@2	P@3	P@4	DCG@2	DCG@3	DCG@4	DCG@5
IR	32.1 ± 2.4%	44.6 ± 2.6%	54.0 ± 3.6%	64.8 ± 4.8%	40.0 ± 2.1%	44.7 ± 2.5%	49.3 ± 2.9%	63.0 ± 1.3%
Word2Vec	23.9 ± 2.8%	40.6 ± 1.9%	56.1 ± 1.9%	72.0 ± 2.0%	34.4 ± 1.9%	42.2 ± 1.8%	49.0 ± 1.5%	59.9 ± 1.2%
FastText	28.4 ± 3.6%	42.1 ± 2.9%	53.9 ± 3.1%	66.7 ± 2.3%	37.0 ± 2.9%	42.9 ± 3.0%	48.4 ± 2.6%	61.3 ± 1.7%
Sent2Vec	26.0 ± 2.6%	45.9 ± 2.5%	61.9 ± 3.2%	78.6 ± 3.1%	38.6 ± 2.1%	46.5 ± 2.0%	53.7 ± 2.2%	62.0 ± 1.2%
AnswerBot	33.9 ± 2.7%	54.5 ± 4.2%	68.9 ± 3.2%	81.4 ± 2.6%	46.9 ± 3.4%	54.1 ± 2.9%	59.5 ± 2.4%	66.7 ± 1.7%
CROKAGE	36.7 ± 2.6%	51.5 ± 3.1%	64.4 ± 1.9%	78.4 ± 2.5%	46.0 ± 2.7%	52.4 ± 2.0%	58.5 ± 1.6%	66.8 ± 1.3%
Ours	45.8 ± 3.2%	60.5 ± 2.5%	72.3 ± 2.7%	85.1 ± 2.6%	55.0 ± 2.6%	61.0 ± 2.5%	66.5 ± 2.1%	72.2 ± 1.6%

Table 5.3: Effectiveness evaluation (Java)

Model	P@1	P@2	P@3	P@4	DCG@2	DCG@3	DCG@4	DCG@5
IR	31.1 ± 2.0%	43.1 ± 2.5%	53.7 ± 2.4%	62.8 ± 2.5%	38.6 ± 2.1%	43.9 ± 2.1%	47.9 ± 1.8%	62.3 ± 1.2%
Word2Vec	26.4 ± 3.2%	37.7 ± 4.3%	51.7 ± 4.1%	66.3 ± 3.3%	33.6 ± 3.7%	40.5 ± 3.6%	46.8 ± 3.0%	59.9 ± 2.0%
FastText	29.8 ± 1.9%	42.7 ± 2.6%	53.0 ± 5.2%	65.8 ± 3.8%	38.0 ± 2.2%	43.1 ± 3.5%	48.6 ± 2.7%	61.9 ± 1.4%
Sent2Vec	24.8 ± 2.2%	45.8 ± 3.4%	62.0 ± 2.8%	80.8 ± 2.6%	38.0 ± 2.9%	46.1 ± 2.5%	54.2 ± 1.8%	61.7 ± 1.4%
AnswerBot	33.3 ± 3.0%	48.1 ± 2.4%	61.1 ± 2.5%	75.0 ± 2.3%	42.6 ± 2.4%	49.1 ± 2.5%	55.1 ± 2.3%	64.8 ± 1.6%
CROKAGE	38.9 ± 4.3%	53.4 ± 5.5%	65.3 ± 4.8%	78.0 ± 2.8%	48.0 ± 5.0%	54.0 ± 4.4%	59.4 ± 3.3%	68.0 ± 2.6%
Ours	58.6 ± 4.3%	73.9 ± 1.4%	84.3 ± 2.0%	92.9 ± 1.8%	68.2 ± 2.3%	73.4 ± 2.0%	77.1 ± 1.4%	79.9 ± 1.7%

Experimental Results The experimental results of our *QueryRewriter* compared to the above baselines for Python and Java are summarized in Table 5.2 and Table 5.3 respectively.

We do not report $P@5$ and $DCG@1$ in our tables, since $P@5$ is always equal to 1 and $DCG@1$ is always equal to $P@1$, both can be easily inferred from the tables. The best performing system for each column is highlighted in boldface. From the table, several points stand out:

1. It is a little surprising that the **word embedding-based approaches (e.g., Word2Vec, FastText and Sent2Vec) achieve the worst performance** regarding $P@1$. This indicates that retrieving semantically-equivalent questions from a set of similar questions is a non-trivial task. Word2Vec and Sent2Vec map each question to a fixed-length vector, so the vectors of similar questions are also close together in vector space. However, due to the reason that all candidate questions in \mathcal{Q} are similar to each other, it is thus very hard for Sent2Vec and Word2Vec approach to distinguish the duplicate questions from a list of similar questions. This is the reason why their performance is weak and ineffective for retrieving semantically-equivalent questions.
2. Regarding the $P@1$ score, the traditional **IR method performs better** than the word embedding-based methods (i.e., Word2Vec and Sent2Vec). For the IR based approach, it relies heavily on how similar the testing question and its duplicate question are. Considering that many duplicate questions may share same lexical units with the testing questions, these duplicate questions can be easily retrieved by the IR-based approach. However, there is still a large number of duplicate questions that are semantically-equivalent with only a few common words or without at all (e.g., as shown in Fig. 5.1), the IR-based approach are unable to retrieve these questions correctly solely based on the similarity between words or tokens. This may also explain its surprisingly low score as K increases.
3. AnswerBot and CROKAGE perform better than other baselines excluding our proposed model. This is because both AnswerBot and CROKAGE combines the

lexical-based model (i.e., IR) and semantic-based model (e.g, Word2Vec, FastText and Sent2Vec) for modeling the question titles, which also signals that solely based on the lexical features or semantic features is not sufficient for our task. It is also notable that the performance of CROKAGE is better than the AnswerBot approach. We attribute this to the different word embedding techniques they employed. AnswerBot combines IDF metrics with Word2Vec model, while CROKAGE combines IDF metrics with FastText model. This signals that the FastText model has its advantage as compared to Word2Vec model for modeling the duplicate question titles.

4. It is clear that **our model outperforms all the other methods by a large margin** in terms of $P@K$ and $DCG@K$ scores of different depth. We attribute this to the following reasons: Firstly, *QueryRewriter* generates multiple paraphrase questions for a given testing question. Adding these paraphrase questions can reduce the lexical gap between the testing questions and its corresponding duplicate questions and alleviate the *query mismatch* problem for different developers. Secondly, all of the baseline methods including our approach can be viewed as variants of embedding algorithm(s), which can map the questions into vectors of a high-dimensional space and then calculate the relevance score between vectors. Hence the key of retrieving semantically-equivalent questions relies on how good the embeddings are for capturing the semantics of different duplicate questions. Our approach has its advantage as compared to other baselines because our *QueryRewriter* trains the historical duplicate question pairs in Stack Overflow by using a text-to-text transformer. As a result, the embeddings generated by our approach are more suitable for identifying the semantically-equivalent questions. The superior performance of our approach also verifies the embeddings generated by our approach convey a lot of valuable information.

Answer to RQ-1: How effective is our approach for retrieving semantically-equivalent questions? – we conclude that our approach is highly effective for semantically-equivalent question retrieval in Stack Overflow.

5.4.2 Context Analysis

As the technical Q&A sites (i.e., Stack Overflow) are used by developers and professional experts, the questions in these Q&A communities are, more often than not, very professional with specific domain context. For example, these questions often include software-specific entities (e.g., software libraries/frameworks, software-specific concepts). To investigate whether the domain-specific context could influence the performance of our approach, or in other words, whether our model can learn the domain-specific context features from the training corpus, we perform a context analysis for this study. **Experimental Setup** For the context analysis, we create a training set without domain-specific context and use the same testing set for evaluation. To do this, we check each token in the training corpus if the token is a normal English word (using the NLTK package), and we only keep the English words and remove the other non proper English words. As a result, the software-specific terms within the master questions and duplicate questions are deleted. After that, We retrain our *QueryRewriter* and all the baselines on the non domain-specific context training corpus and perform the same evaluation as in RQ-1.

Experimental Results The experimental results of our *QueryRewriter* and other baselines for Python and Java dataset are presented in Table 5.4 and Table 5.5 respectively. From the tables, we can deduce the following key findings:

1. The performance of all models decreases on the training set without domain-specific context. This suggests that the domain-specific context has a major influence on the overall performance. We further counted the unique tokens of the training corpus with and without the domain-specific context, for Python dataset, only 4,756 out of 19,208 tokens are remained; for Java dataset, only 4,874 out of 22,344 tokens are

Table 5.4: Context Analysis of $P@1$ (Python)

Approach	Without Context	With Context	Δ Improve
IR	$24.9 \pm 2.2\%$	$32.1 \pm 1.2\%$	28.9%
Word2Vec	$22.2 \pm 2.9\%$	$23.9 \pm 2.8\%$	7.7%
FastText	$24.4 \pm 1.5\%$	$28.4 \pm 3.6\%$	16.4%
Sent2Vec	$24.2 \pm 2.1\%$	$26.0 \pm 2.6\%$	7.4%
AnswerBot	$29.4 \pm 3.8\%$	$33.9 \pm 2.7\%$	15.3%
CROKAGE	$26.9 \pm 2.3\%$	$36.7 \pm 2.6\%$	36.4%
Ours	$31.1 \pm 2.4\%$	$45.8 \pm 3.2\%$	47.3%

Table 5.5: Context Analysis of $P@1$ (Java)

Approach	Without Context	With Context	Δ Improve
IR	$26.9 \pm 2.1\%$	$31.1 \pm 2.0\%$	15.6%
Word2Vec	$24.5 \pm 2.4\%$	$26.4 \pm 3.2\%$	7.6%
FastText	$25.5 \pm 2.6\%$	$29.8 \pm 1.9\%$	16.7%
Sent2Vec	$23.4 \pm 2.5\%$	$24.8 \pm 2.2\%$	6.0%
AnswerBot	$26.8 \pm 3.3\%$	$33.3 \pm 3.0\%$	24.3%
CROKAGE	$28.2 \pm 2.2\%$	$38.9 \pm 4.3\%$	37.9%
Ours	$37.7 \pm 4.2\%$	$58.6 \pm 4.3\%$	55.4%

remained. The large proportion of domain-specific context in Stack Overflow may also explain the performance drop in all baseline methods.

2. It is notable that after adding the domain-specific context, our model achieves the biggest performance rise among different models. This reveals that our model is effective in learning the domain-specific features and knowledge. Moreover, the performance of our proposed model still outperforms the other baseline approaches even under the training corpus without domain-specific context, which justifies the robustness of our model.

Answer to RQ-2: How effective is our approach for capturing contextual information? – we conclude that the domain-specific context can influence the model’s performance, and our approach is highly effective for learning domain-specific context information.

Table 5.6: Ablation Analysis

Measure	Python		Java	
	Drop-PQ	Ours	Drop-PQ	Ours
P@1	36.9%	45.8%	49.5%	58.6%
P@2	50.0%	60.5%	65.3%	73.9%
P@3	62.1%	72.3%	76.9%	84.3%
P@4	74.5%	85.1%	87.5%	92.9%
DCG@2	45.1%	55.0%	59.4%	68.2%
DCG@3	51.2%	61.0%	65.3%	73.4%
DCG@4	56.5%	66.5%	69.8%	77.1%
DCG@5	66.4%	72.2%	74.7%	79.9%

5.4.3 Ablation Analysis

Ablation analysis is a common method to estimate the contribution of a component to the overall system [Gao et al., 2020a]. It studies the performance of a system by removing certain components. Our *QueryRewriter* learns to encode the duplicate question pairs from Stack Overflow, so that two semantically-equivalent questions are close in terms of vector representation. When we perform question retrieval tasks, a main novelty of our approach is adding paraphrase questions generated by *QueryRewriter*. In this research question, we perform an ablation analysis to investigate if the novel aspect that we introduce helps. To be more specific, we investigate the effectiveness of the added paraphrase question to our model.

Experimental Setup For the ablation analysis, we compare our approach with one of its incomplete variants, named **Drop-PQ**. Different from our proposed model, **Drop-PQ** removes all the generated paraphrase questions added to our model, and only keeps the embedding of the original testing question. By going through the same steps as our approach in Section 5.3.2, we can evaluate **Drop-PQ** model for the semantic-equivalent question retrieval task.

Experimental Results The comparison results between **Drop-PQ** and our approach are

displayed in Table 5.6. We observe the following points from the table:

1. By comparing the results of our approach and **Drop-PQ**, it is clear that **incorporating the paraphrase questions improves the overall performance**. When adding the paraphrase questions to our model, the $P@1$ score is improved by 24.1% in Python and 18.3% in Java dataset. We attribute this to the ability of paraphrase questions to reduce the lexical gap between the semantically-equivalent questions.
2. By comparing the results of **Drop-PQ** and our previous baselines in RQ-1, we can see that **even by dropping the paraphrase questions, Drop-PQ still achieves better or comparable results than other baselines**. This is because, even removing the paraphrase questions, the question embeddings are generated from the same Encoder of our text-to-text transformer. This further verifies the importance and necessity of the embeddings of our approach.

Answer to RQ-3: How effective is the paraphrase generation component added to our <i>QueryRewriter</i>? - We conclude that adding paraphrase questions significantly improves the overall performance of our model

5.4.4 Parameter Tuning Analysis

Considering that paraphrase questions have a major influence on the overall performance of our model, a key parameter is the number of paraphrase questions added to our approach, referred to N as shown in Equation 5.4. We perform a parameter tuning analysis to investigate the optimal parameter settings of N for our *QueryRewriter*.

Experimental Setup For the parameter tuning analysis, we fine tune N , the number of paraphrase questions added to our model, on Python and Java dataset respectively. We vary N from 0 to 20 with step size 1 to select the optimal parameter, where N equals 0 corresponds exactly to the **Drop-PQ** model in RQ-3. The parameters corresponding to the best $P@1$ were used to report the final results.

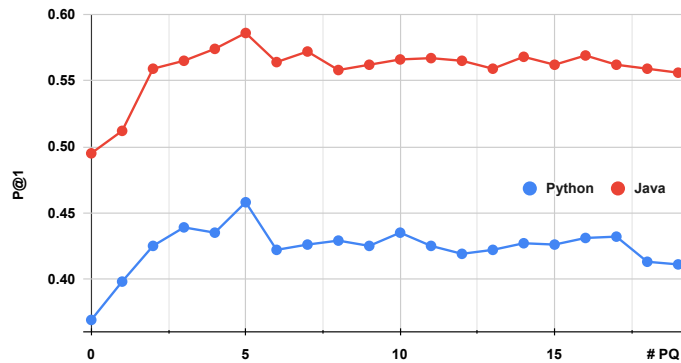


Figure 5.9: Parameter Tuning Analysis

Experimental Results Fig. 5.9 illustrates the performance of the model with respect to different values of N on Python and Java dataset. From the figure, we can make the following observation:

1. **The performance of our model rapidly increases as N is increased from 0 to 3.** For example, when N is increased from 0 to 3, $P@1$ is sharply increased from 36.9% to 43.9% on Python dataset, and from 49.5% to 56.5% on Java dataset. This further justifies our previous conclusion that paraphrasing questions makes an important contribution to the overall performance of our approach.
2. **We notice that our model achieves its best performance when N reaches around 5.** We thus recommend to initialize the N to be around the above value, which is close to the optimal settings of N in our *QueryRewriter*.
3. **The overall performance of our model reaches a plateau after achieving the best performance.** The possible reason may be that adding too many paraphrase questions brings in the higher level of noise for question embeddings, hence the embeddings of the user query question can be contaminated by adding too many paraphrase questions.

Answer to RQ-4: What is the optimal parameter settings for QueryRewriter? - We recommend to set the number of generated paraphrase questions to 5, which is close to the optimal settings of N in our approach.

5.4.5 Effectiveness of CodeSelector

When trying to solve daily coding problems, developers often formulate their problems as a question and/or a few keywords to some search engines. The search engine returns a list of potential posts which may contain useful answers. Due to the complexity of the online CQA forums and the large volume of information generated from it, software developers may encounter the *information overload* problem wherein the massive amounts of information makes it hard to be aware of the most relevant resources to meet the information needs of the developers. To alleviate this *information overload* problem, we propose a *CodeSelector* to rank the code snippets candidates via pairwise comparisons. To evaluate our approach, we conducted a large-scale automatic evaluation experiment to evaluate the effectiveness of our approach to identify the best code solution for a technical problem.

Data Preparation To train the *CodeSelector*, we first constructed positive and negative training samples in terms of preference pairs. For each Stack Overflow post, we extracted the code snippets (using $\langle code \rangle$ tags) within the post’s question body and corresponding post question title. In order to avoid being context-specific, numbers and strings within a code snippet are replaced with special tokens “NUMBER” and “STRING” respectively. We first adopted the NLTK [Bird and Loper, 2004] toolkit to tokenize the code snippets, we removed the code snippets that are too long (more than 512 tokens) or too short (less than 5 tokens). This is because for a given code snippet, it is unable to capture the code semantics if it is too short. We set the 512 as the maximum number of code snippet tokens since the maximum input sequence length of BERT [Devlin et al., 2018] is restricted to 512 tokens, and this setting is sufficient for most cases of code snippets in Stack Overflow [Gao et al., 2020a]. For question titles, we only preserved the “how” related questions in Stack Overflow. The resulting $\langle question, code snippet \rangle$ pairs are added to our corpus. Towards this end, we collected 218K QC pairs for Python dataset and 272K QC pairs for Java dataset.

Table 5.7: QC Dataset Statistics

Python	# $\langle q, cs \rangle$ Pairs	218,717
	# best code snippet	112,447
	# non-best code snippet	106,270
	# non-relevant code snippet	112,447
	# Positive Samples	141,074
	# Negative Samples	141,224
Java	# $\langle q, cs \rangle$ Pairs	272,120
	# best code snippet	134,993
	# non-best code snippet	137,127
	# non-relevant code snippet	134,993
	# Positive Samples	177,340
	# Negative Samples	176,942

For each question in the corpus, we make the code snippet associated with the accepted answer or the highest-vote answer as the *best code snippet*, the code snippet associated with the non-accepted answers as the *non-best code snippet*, and randomly select the code snippet from other questions as the *non-relevant code snippet*. According to our heuristic rules described above in Section 5.3.3, we constructed our dataset with balanced positive and negative preference pairs. We randomly sampled 5,000 samples for validation and 5,000 pairs for testing respectively, and kept the rest for training. The details of the statistics of our collected dataset are summarized in Table 5.7.

Experimental Setup To evaluate our *CodeSelector* performance for identifying the best code snippet for a technical question, for each question q in the testing set, we employ the same KNN strategy in RQ-1 to search its top-K similar questions over the whole dataset. Undoubtedly, the testing question itself can be found. We then constructed a code snippet candidates pool \mathcal{C} by gathering all the code snippets associated with the returned questions. In the light of this, we can ensure the ground truth code snippet (*best code snippet*) is in the code snippet candidates pool \mathcal{C} . Following that, we pair the given question q with each of the code snippet in \mathcal{C} to make a QC pair. Hereafter, by doing a pairwise comparison

between each two QC pairs, we can generate a ranking list of preference scores for each code snippet. All the code snippet candidates can be ranked by their preference scores. For this code selection task, we also employ the same automatic evaluation metric $P@K$ and $DCG@K$ used in RQ-1. $P@K$ and $DCG@K$ stands for the proportion of the selected code snippets in the top-K that are the ground truth.

Experimental Baselines To demonstrate the effectiveness of our proposed approach, we compare it with several competitive baseline approaches. We adapt these approaches slightly for our specific task, i.e., selecting the best code snippet from a pool of code snippet candidates. We briefly introduce these approaches and our evaluation experimental settings below. For each method below, the involved parameters are carefully tuned, and the best performance of each approach is used to report the final results.

- **Traditional Classifiers** Considering that our *CodeSelector* ranks the code snippet candidates by doing classification between QC pairs, it is hence natural to compare our approach with traditional classifiers. Recently Calefato et al. [Calefato et al., 2019] proposed an approach for best answer prediction problem by formulating it as a binary-classification task. The binary-classification methods output a score referring to a probability of relevance. They assessed 26 traditional classifiers for predicting the best answer in Stack Overflow. We choose the two most effective traditional classifiers, *xgbTree* and *RandomForest*, to apply to our code snippet recommendation task. In our experiments, we treated the pair of $\langle question, best\ code\ snippet \rangle$ as positive sample and the pair of $\langle question, non - best\ code\ snippet \rangle$ as negative sample, we then train the traditional classifiers with these training samples. Thereafter, we rank the code snippet candidates by the relevance scores generated by the above trained classifiers.
- **Answer Ranking Methods** The code snippet selection need of our task is similar to the answer ranking problem in CQA forums. Hence our task is transformed to

find an optimal ranking order of the code snippet candidates according to their relevance to the given query question. Two answer ranking methods, i.e., AnswerBot [Xu et al., 2017] and DeepAns [Gao et al., 2020b] are chosen as baselines. Regarding the AnswerBot baseline, their user study showed a promising performance for selecting salient answers in the second stage of their approach. Regarding the DeepAns baseline, they calculated a matching score via a deep neural network between each answer and the question title; the experimental results show that DeepAns is effective for selecting the most relevant answer compared with several state-of-the-art benchmarks. For both of these answer ranking methods, an overall score is computed to estimate the relevance between each answer and the question title. For our task, we can replace the answer with the code snippet and thus adapt their answer ranking methods to our task of ranking code snippets among a set of code snippet candidates.

- **DL-based Code Search Methods** Another thread of similar research that is relevant to our work is code search. A plethora of approaches have been investigated for searching code snippet in software repositories, and recent DL (deep learning) - based approaches have achieved promising results for this task. The DL-based code search methods advocate the idea of mapping and matching data in a high-dimensional vector space. Three state-of-the-art DL-based code search methods, i.e., NCS [Sachdev et al., 2018], DeepCS [Gu et al., 2018] and CROKAGE [da Silva et al., 2020] are chosen for our study. NCS is an unsupervised technique for neural code search proposed by Facebook [Sachdev et al., 2018]. They combine the word embeddings and TF-IDF weighting derived from a code corpus. In our experiment, we used all the collected QC pairs as the code corpus for training the word embeddings and TF-IDF weightings. DeepCS is a supervised technique which jointly embeds code and natural language description into a high-dimensional

vector space proposed by Gu et al [Gu et al., 2018]. The author constructed the $\langle C, D+, D- \rangle$ triples to train their model for minimizing the ranking loss, where C is the code snippet, $D+$ and $D-$ are the correct and incorrect description respectively. In our experiment, for each code snippet C , we treat the associated question title as the correct description $D+$, and treat a randomly selected question title as the incorrect description $D-$. CROKAGE [da Silva et al., 2020] is a tool to deliver a comprehensible solution for a programming task. It chooses the top quality answers related to the task. To mitigate the lexical gap problem, CROKAGE calculates the scores for candidate QA pairs using four similarity factors (e.g., *lexical score*, *semantic score*, *API method score* and *API class score*). Specifically, they use TF-IDF and fastText embedding to calculate the *lexical scores* and *semantic scores* respectively. They also reward the QA pairs which contain the top API methods and relevant API classes with *API method scores* and *API class scores*. Their final outputs contain both code examples and code explanations. Because our study mainly focuses on the code snippet recommendation task, we only retain the code examples part and remove the code explanation part.

Table 5.8: Effectiveness evaluation of CodeSelector (Python)

Model	P@1	P@2	P@3	P@4	DCG@2	DCG@3	DCG@4	DCG@5
RandomForest	26.6 ± 1.6%	49.6 ± 2.6%	69.7 ± 2.0%	86.4 ± 1.4%	41.1 ± 2.1%	51.1 ± 1.8%	58.3 ± 1.5%	63.5 ± 1.0%
xgbTree	24.3 ± 1.5%	47.3 ± 2.4%	69.1 ± 2.5%	85.8 ± 1.8%	38.8 ± 1.9%	49.7 ± 1.8%	56.9 ± 1.4%	62.4 ± 0.9%
AnswerBot	31.0 ± 1.5%	51.1 ± 2.3%	70.4 ± 2.1%	87.4 ± 1.5%	43.7 ± 1.8%	53.3 ± 1.4%	60.6 ± 1.1%	65.5 ± 0.8%
DeepAns	29.6 ± 2.2%	52.3 ± 1.9%	71.2 ± 1.2%	88.5 ± 1.2%	43.9 ± 1.9%	53.4 ± 1.5%	60.8 ± 1.3%	65.3 ± 1.1%
NCS	29.8 ± 1.6%	52.7 ± 2.8%	71.3 ± 2.0%	88.3 ± 1.9%	44.2 ± 2.2%	53.5 ± 1.6%	60.9 ± 1.6%	65.3 ± 1.0%
DeepCS	29.5 ± 2.2%	51.3 ± 2.3%	70.3 ± 1.5%	86.7 ± 1.4%	43.3 ± 2.2%	52.8 ± 1.7%	59.8 ± 1.4%	64.9 ± 1.2%
CROKAGE	33.3 ± 2.3%	55.0 ± 2.0%	71.9 ± 1.3%	86.5 ± 1.1%	47.0 ± 1.9%	55.4 ± 1.5%	61.7 ± 1.2%	66.9 ± 1.1%
Ours	42.6 ± 2.5%	64.6 ± 1.1%	80.0 ± 1.0%	92.3 ± 0.9%	56.5 ± 1.5%	64.2 ± 1.2%	69.5 ± 1.0%	72.5 ± 1.0%

Experimental Results The experimental results of our proposed model and the above baselines over our Python and Java datasets are summarized in Table 5.8 and Table 5.9 respectively. From the table, we can observe the following points:

1. The performance of **traditional classifiers are comparatively suboptimal**. This

Table 5.9: Effectiveness evaluation of CodeSelector (Java)

Model	P@1	P@2	P@3	P@4	DCG@2	DCG@3	DCG@4	DCG@5
RandomForest	24.8 ± 2.9%	47.5 ± 2.3%	67.8 ± 2.1%	85.0 ± 1.4%	39.1 ± 2.2%	49.2 ± 1.8%	57.0 ± 1.7%	62.5 ± 1.3%
xgbTree	25.8 ± 1.7%	47.5 ± 2.6%	68.1 ± 2.4%	85.0 ± 1.5%	39.5 ± 2.1%	49.8 ± 2.0%	57.1 ± 1.5%	62.8 ± 1.1%
AnswerBot	31.4 ± 2.1%	52.1 ± 1.3%	70.9 ± 1.1%	86.9 ± 1.7%	44.5 ± 1.4%	53.9 ± 1.0%	60.7 ± 0.8%	65.8 ± 0.8%
DeepAns	29.1 ± 2.3%	52.5 ± 2.3%	71.3 ± 2.3%	86.7 ± 1.4%	43.9 ± 2.2%	53.2 ± 1.9%	59.9 ± 1.6%	65.1 ± 1.2%
NCS	30.4 ± 1.8%	52.1 ± 1.7%	71.4 ± 1.5%	86.9 ± 1.4%	44.1 ± 1.6%	53.7 ± 1.3%	60.4 ± 1.2%	65.5 ± 0.9%
DeepCS	28.5 ± 3.9%	48.2 ± 3.4%	65.8 ± 4.6%	81.6 ± 3.5%	40.9 ± 3.3%	49.7 ± 3.6%	56.5 ± 2.9%	63.6 ± 2.0%
CROKAGE	33.6 ± 1.8%	54.6 ± 1.8%	72.3 ± 1.9%	86.6 ± 1.1%	46.8 ± 1.6%	55.7 ± 1.4%	61.8 ± 1.1%	67.0 ± 0.8%
Ours	42.4 ± 1.9%	66.1 ± 1.8%	81.6 ± 1.5%	92.6 ± 1.6%	57.3 ± 1.6%	65.1 ± 1.5%	69.8 ± 1.5%	72.7 ± 0.9%

indicates that traditional classifiers are unable to capture the semantics between the code snippets and the questions.

2. The **answer ranking methods and the DL-based code search methods achieve similar results**. The underlying idea of these two kinds of methods is similar, namely the application of applying the embedding technique to map the raw data (including the questions and code snippets) into a high-dimensional space and then estimate the match score between them. This may explain the reason why their performances are comparable with each other. CROKAGE has its advantage as compared to other benchmarks. This is caused by several reasons: First, it combines the lexical features and semantic features (using lexical score and semantic score) that is why it is superior to the traditional classifiers. Second, in addition to lexical features and semantic features, it also incorporates the API related features, this results in its superior to the other answer ranking methods and DL-based code search methods.
3. **Our proposed model is substantially better than all of the baseline methods**. We attribute this to the following reasons: First, all of the baseline approach (including traditional classifiers, answer ranking methods as well as the DL-based code search methods) can be viewed as pointwise approaches. Pointwise approaches transform the task of ranking into classification or regression on single QC pairs. They are thus unable to consider the relative order or preference between different code snippets. Nevertheless, ranking is more about predicting relative orders rather than precise

Table 5.10: Component-Wise Evaluation (Python)

Measure	Drop-Pairwise	Drop-Bert	Ours
P@1	36.4 ± 1.8%	33.9 ± 1.2%	42.6 ± 2.5%
P@2	59.7 ± 2.1%	57.5 ± 1.1%	64.6 ± 1.1%
P@3	78.1 ± 1.7%	76.7 ± 1.9%	80.0 ± 1.0%
P@4	91.5 ± 1.0%	90.5 ± 1.2%	92.3 ± 0.9%
DCG@2	51.1 ± 1.8%	48.8 ± 1.0%	56.5 ± 1.5%
DCG@3	60.3 ± 1.5%	58.4 ± 1.2%	64.2 ± 1.2%
DCG@4	66.1 ± 1.2%	64.4 ± 0.9%	69.5 ± 1.0%
DCG@5	69.4 ± 1.0%	68.0 ± 0.6%	72.5 ± 1.0%

relevance scores. In light of this, we propose a pairwise approach to judge the preference relationship between any two given QC pairs rather than the absolute relevance value of a single QC pair. Compared with the pointwise approaches, our model not only considers the relevance between a query and a code snippet, but also investigates the relevance preference of two QC pairs. This is why it is superior to other pointwise baselines. Second, in addition to constructing the preference pairs, our model also incorporates the BERT model to embed the QC pairs. The attention mechanism behind BERT makes it possible to express sophisticated functions beyond the simple weighted average, which results in its superior to the DL-based code search methods. This also signals that the embeddings produced by BERT convey much valuable information, which can better capture the semantics between the user query question and the code snippets.

4. By comparing the evaluation results of the different datasets (i.e., Python and Java), we can see that our proposed model behaves consistently across different programming languages. This also indicates the generalization ability of our approach.

Answer to RQ-5: How effective is our CodeSelector for best code snippet selection? - we conclude that our CodeSelector is effective for selecting the correct code snippet for a given technical question.

Table 5.11: Component-Wise Evaluation (Java)

Measure	Drop-Pairwise	Drop-Bert	Ours
P@1	36.9 ± 2.2%	34.6 ± 1.7%	42.4 ± 1.9%
P@2	60.9 ± 2.6%	59.5 ± 2.6%	66.1 ± 1.8%
P@3	78.4 ± 1.8%	78.6 ± 1.8%	81.6 ± 1.5%
P@4	91.4 ± 1.2%	91.1 ± 1.0%	92.6 ± 1.6%
DCG@2	52.0 ± 2.3%	50.3 ± 2.1%	57.3 ± 1.6%
DCG@3	60.8 ± 1.8%	59.8 ± 1.6%	65.1 ± 1.5%
DCG@4	66.4 ± 1.6%	65.3 ± 1.2%	69.8 ± 1.5%
DCG@5	69.7 ± 1.2%	68.7 ± 0.9%	72.7 ± 0.9%

5.4.6 Component-Wise Evaluation

Compared with other methods, the key advantages of our *CodeSelector* are its two sub-components: incorporating the BERT model and employing the pairwise comparisons. To verify the effectiveness of both aforementioned components, we conduct a component-wise evaluation to evaluate their performance one by one.

Experimental Setup For our component-wise evaluation, we compare our model with two of its incomplete versions:

- **Drop-Pairwise** removes the pairwise comparison component from our *CodeSelector*. In this experiment, for a given question, we drop the process of constructing the positive and negative preference pairs. To do this, we reconstruct the best QC pairs as positive samples, and make the nonbest and nonrelevant QC pairs as negative samples. The **Drop-Pairwise** model is then trained as a binary classification model same as ours.
- **Drop-BERT** removes the BERT component from our *CodeSelector*. In this experiment, we keep the preference pairs construction process but drop the BERT embedding process. To do this, we replace the BERT embedding layers (described in Section 5.3.3) with the traditional Word2Vec embedding layers. The **Drop-BERT**

can then be trained with the preference QC pairs in just the same way.

Experimental Results The evaluation results of **Drop-Pairwise** and **Drop-BERT** are displayed in Table 5.10 and Table 5.11 respectively. It can be seen that:

1. **Dropping either component does hurt the performance of *CodeSelector*.** This justifies the importance and effectiveness of both components.
2. **Drop-BERT achieves the worst performance.** This indicates that a good embedding technique has a major influence on the overall performance. For example, when adding BERT component for embedding the QC pairs, the $P@1$ score is improved by 20.4% and 18.2% on Python and Java dataset respectively. We attribute this to the advantage of BERT for capturing the intent of the query question as well as the program code. This is why our model outperforms other deep learning-based code searching methods.
3. By comparing the results of **Drop-Pairwise** and **Ours**, we can measure the performance improvement achieved due to the employment of pairwise comparison component. It is clear that by **removing the pairwise comparison component, there is a significant drop with respect to different metrics.** This shows that the pairwise comparison component has a significant contribution to the overall performance of our model. This is the reason why our model outperforms other pointwise baselines.

Answer to RQ-6: How effective is the BERT component and preference pairs added to our CodeSelector? - we conclude that both the BERT component and the preference pairs are effective and helpful to enhance the performance of our CodeSelector.

5.4.7 Robustness Analysis

Considering the complexity and diversity of the CQA sites, there is little chance to find the past solved questions that exactly match the user query questions. We thus have to enlarge K - the number of retrieved questions - to improve the recall of the relevant questions as

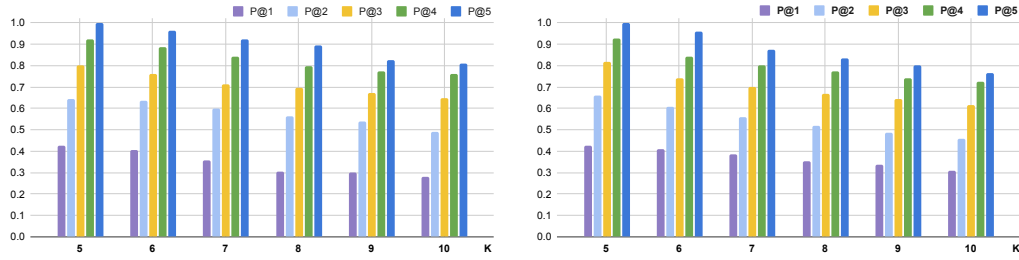


Figure 5.10: Robustness Analysis on Python (left) and Java (right)

well as the potential code snippets within these questions. However, a larger K may often bring more noise into the code snippet candidates pool. This increases the complexity and difficulty for recommending the potential code snippet. We conduct a robustness analysis to investigate our model’s performance with respect to different number of retrieved questions.

Experimental Setup To verify the robustness of our proposed model, we set different thresholds for the number of returned questions. In particular, we increase the number of returned questions k from 5 to 10 ($k=5$ corresponds to our model described in RQ-4), and then evaluate the performance of our *CodeSelector* with respect to different parameter settings of K .

Experimental Results The average $P@1-5$ over Python and Java datasets are shown in Fig. 5.10. By jointly analyzing these two figures, we can have the following observation:

1. The **overall performance trend of our model goes down as k increases**. This justifies our previous concerns that more noise are introduced when enlarging k , which incurs bigger challenges for our task.
2. The **performance of our model on the Java dataset decreases faster than that on Python dataset**. The reason for this phenomenon may be that the Java code snippets are more complex, and contain more noise compared with Python code snippets even under the same k settings.

3. The **performance drop of our model with increasing k is not very large**. For example, when we set k to 10, the performance of our model on $P@1$ is still better than or comparable with the best performance on several baselines. This further shows the robustness of our model.

Answer to RQ-7: *How robust is our CodeSelector with different parameter settings?* - we conclude that our approach is robust to noises.

5.5 Human Evaluation

The goal of our tool is to recommend the best code snippets that most closely match a developer’s intent from past solved questions in Stack Overflow. We perform a user study to measure how developers actually perceive the results produced by our approach.

5.5.1 Human Evaluation Preliminary

Participants Selection Since we are recommending code snippets for newly posted queries, we thus sampled 50 unanswered Python posts from Stack overflow for our human evaluation. We recruited 10 participants to join our human evaluation, which is larger than or comparable to the size of the user study participants in previous studies [Liu et al., 2018, Xu et al., 2017, Gao et al., 2020b]. Our user study includes 1 postdoctoral fellow and 4 Ph.D. Students majoring in Computer Science and 5 software developers from industry. All of these selected participants have working experience with Python development. The years of their working experience on Python range from 3 to 10 years. In practice, our tool aims to help different levels of practitioners, from novice to senior developers. The diversity of their background (i.e., from academic and industry) and their working years can improve the generality of our user study. Our human evaluation includes two user studies: the user study on question relevance and the user study on code usefulness. All the selected evaluators are asked to participate in the above two user studies.

Human Evaluation Baselines We use the following baselines for our human evaluation:

- **Google Search Engine.** Considering developers usually search for technical help using the Google search engine, we employ the Google search engine as one of our baselines. For the Google search engine, we use the question title of the post as the search query, we then add the “site:stackoverflow.com” to the end of the search query so that it only searches on Stack Overflow. We treat the first ranked question returned by Google search engine as the most relevant question, we treat the code fragment within its best answer (the accept answer or the highest vote answer) as the most relevant code snippet.
- **Stack Overflow Search Engine.** Similar to Google search engine, Stack Overflow also provide a service to search relevant posts. For the Stack Overflow search engine, we refer to the first ranked related question suggested by the Stack Overflow as the most relevant question, and the code snippet within its best answer as the recommended solution.
- **CROKAGE.** Since **CROKAGE** outperforms the other baselines in both stage one (semantically-equivalent question retrieval) and stage two (best code snippet recommendation). Therefore we also employ the **CROKAGE** to find the relevant questions and code snippets for a given query in our human evaluation.
- **1stRanked.** Given a user query, our approach first retrieves the semantically-equivalent questions, then it reranks all the code snippets associated with these questions. Different from our approach, the **1stRanked** method drops the second stage of the code reranking process and simply uses the code snippet from the first ranked question as the recommended solution.
- **QUE2CODE.** For our approach, we perform an end-to-end evaluation using our QUE2CODE tool. In particular, we use the *QueryRewriter* to retrieve the semantic

QUESTION RELEVANCE USER STUDY
Scoring Criterion: Score 1: The two questions have no relevance. Score 2: The two questions share some common words but are semantically irrelevant. Score 3: The two questions are semantically similar. Score 4: The two questions are semantically equivalent (they are identical in meaning).
Recommended Question: How to convert binary file into readable format on linux server
Reference Question: Binary file downloaded is unreadable (57738034)
<input type="radio"/> Score 1
<input type="radio"/> Score 2
<input type="radio"/> Score 3
<input type="radio"/> Score 4

Figure 5.11: Example of Question Relevance User Study

relevant questions in Stack Overflow, and then use the *CodeSelector* to select the best code snippet for the unanswered question. After *CodeSelector* reranks all the code snippets, we treat the first ranked code snippet and its associated question as the recommended solution.

5.5.2 User Study on Question Relevance

Experimental Setup Since we are recommending code snippets from semantically-equivalent questions, if the retrieved questions are not relevant to the user query question, it is unlikely that we can select the appropriate code snippet from the answer candidates pool. Therefore we first conduct a user study to measure how humans perceive the question retrieval results. To do this, we consider the *question relevance* modality for this user study. The *question relevance* metric measures how relevant is the retrieved question to the user query question. To be more specific, we asked participants to do a web questionnaire. For each unanswered question, the evaluator is displayed with this user query question along with 5 retrieved questions from the above baselines. For each retrieved question, the participant is asked to give a score between 1 to 4 to measure the relevance between the user query question and the retrieved question. We define the scoring criterion as follows:

- Score 1: The two questions have no relevance.

- Score 2: The two questions share some common words but are semantically irrelevant.
- Score 3: The two questions are semantically similar.
- Score 4: The two questions are semantically equivalent (they are identical in meaning).

We provide the scoring criterion in the beginning of each questionnaire to guide participants. Fig. 5.11 shows one example in our survey. It is worth mentioning that the order of the 5 retrieved questions is randomly decided, so the participants do not know which question is generated by our approach.

Experimental Results We obtained 500 groups of scores from the above user study. Each group contains 5 scores for 5 retrieved questions respectively. We regard a score of 0 as *low* quality, a score of 1 as *medium-* quality, a score of 2 as *medium+* quality, a score of 4 as *high* quality. The score distribution and the mean score of *question relevance* across baselines are presented in Table 5.12. We also evaluate whether the differences between our approach and the baselines are statistically significant by performing Wilcoxon signed-rank test [Wilcoxon, 1992]. From the table, we can see that:

1. **The Stack Overflow search engine achieved the worst performance among all the baselines.** The very large proportion of low quality questions reflects that the Stack Overflow search engine is ineffective for searching relevant questions for newly posted questions. We manually checked the 50 questions produced by Stack Overflow search engine, it recommended the same common question 15 times, which explains its weak performance in question retrieval tasks.
2. **Our approach outperforms the CROKAGE baseline regarding the *question relevance* metric.** The results of human evaluation are consistent with large-scale auto-

Table 5.12: User Study on Question Relevance

Measure	Low	Medium-	Medium+	High	Mean Score	P-value
Google	3.20%	31.60%	36.80%	28.40%	2.91	> 0.5
Stack Overflow	58.40%	28.80%	6.80%	6.00%	1.60	< 0.01
CROKAGE	12.80%	40.00%	36.40%	10.80%	2.45	< 0.01
1stRanked	8.00%	33.20%	45.60%	13.20%	2.64	< 0.01
Ours	2.80%	27.20%	47.20%	22.80%	2.90	—

matic evaluation results, which further justifies the effectiveness of our approach for retrieving relevant questions.

3. **The 1stRanked method has its advantages as compared to other baselines (i.e., Stack Overflow and CROKAGE).** This is reasonable because both the 1stRanked method and our approach employs *QueryRewriter* for embedding relevant questions. The *QueryRewriter* incorporates historical duplicate question pairs from Stack Overflow, such that two semantically-equivalent questions are close in terms of vector representations.
4. **Google search engine performs better than our approach regarding the *high quality* questions.** Considering Google’s capability, i.e., the larger searching database and accumulated user searching histories, it is not surprising that the Google search engine can identify the high quality relevant questions for the user query. **However, our approach still achieves comparable mean score regarding the *question relevance* metric, and the difference between our approach and Google search engine is also not statistically significant.** This is because the proportion of relevant questions (including the *medium+* and *high* quality questions) outnumber those of the Google search engine. This reflects that, for a given unseen question, our approach is more likely to retrieve relevant questions in general.

CODE USEFULNESS USER STUDY
Scoring Criterion: Score 1: The code snippet is irrelevant and useless for solving the target question Score 2: The code snippet is relevant to the target question but not useful for solving it Score 3: The code snippet is helpful to guide developer's further searching and learning Score 4: The code snippet can successfully solve the target question
Reference Question: Binary file downloaded is unreadable (57738034)
Try the below code, Working with Binary Data
<pre> with open("test_file.docx", "rb") as binary_file: # Read the whole file at once data = binary_file.read() print(data) # Seek position and read N bytes binary_file.seek(0) # Go to beginning couple_bytes = binary_file.read(2) print(couple_bytes) </pre>
<input type="radio"/> Score 1
<input type="radio"/> Score 2
<input type="radio"/> Score 3
<input type="radio"/> Score 4

Figure 5.12: Example of Code Usefulness User Study

5.5.3 User Study on Code Usefulness

Experimental Setup Our final goal is searching for useful code snippets to help developers solve unanswered questions. We also conduct a user study to measure the *code usefulness* of our recommended code snippets. The *code usefulness* metric refers to how useful the recommended code snippet is for solving the user query questions. Similar to our previous user study, for each unanswered question, we provide 5 code snippets candidates recommended by 5 baselines respectively. After that, each evaluator was asked to rate 5 code snippets from 1 to 4 according to its *code usefulness* with respect to the following scoring criterion:

- Score 1: the code snippet is irrelevant and useless for solving the target question.
- Score 2: the code snippet is relevant to the target question, but not useful for solving it.

- Score 3: the code snippet can guide developer’s further searching and learning, which is useful to solve the target question.
- Score 4: the code snippet can successfully solve the target question.

We provide the scoring criterion in the beginning of the questionnaire to guide the evaluators. Fig. 5.12 demonstrates one example of our survey. The evaluators were blinded to which code snippet is generated by our approach. **Experimental Results** Same as for the user study on *question relevance*, after obtaining the evaluator’s feedback, we regard a score of 1 as *low* quality, a score of 2 as *medium–* quality, a score of 3 as *medium+* quality, and a score of 4 as *high* quality. The score distribution and the mean score of *code usefulness* across different baselines are presented in Table 5.13. The Wilcoxon signed-rank test [Wilcoxon, 1992] is also performed between our approach and each baseline method, the results are displayed in the last column of Table 5.13. From the table, we can see that:

1. **Our model significantly outperforms all the baselines (including the Google search engine) regarding the *code usefulness* metric.** This suggests that the code snippets recommended by our approach are considered to be more useful to the given query question compared with baselines. The reason may be due to the two stages of our approach, i.e., *semantically-equivalent question retrieval* and *best code snippet recommendation*. The first stage focuses on retrieving as many as possible relevant questions to construct the candidate set. The more relevant the retrieved question is to the query, the more likely the code snippet associated with the question is helpful to solve the problem. The second stage tries to rank the useful code snippet to the top of the recommendation result.
2. **Compared with the 1stRanked method, our approach has its advantages.** Even though the 1st-ranked method retrieves the same relevant question candidates as ours, it naively picks the code snippet from the most relevant question. However, consid-

ering the complexity of the technical queries, it is very hard, if not possible, to find identical questions to the given query. Therefore it is necessary to consider the correlation between the code snippet and the user query. Different from the 1stRanked method which is solely based on *question relevance*, our *CodeSelector* to rerank all the code snippet candidates by performing pairwise comparisons. The *CodeSelector* not only considers the program semantics between the code snippet and the query question, but also investigates the relevance preference between different QC pairs. Thus a useful code snippet can be ranked higher up among other candidates. The superior performance of our approach regarding the mean score of *code usefulness* further supports the ability of our *CodeSelector* model for recommending useful code snippet.

3. **By comparing the mean score of *code usefulness* and *question relevance*, there is a significant drop overall in every baseline method.** This indicates that compared with relevant question retrieval tasks, identifying useful code snippets is more challenging. Technical questions in Stack Overflow are rather complicated and specific, even though two technical questions are semantically relevant, the code snippet is not applicable or reusable for the programming task. **We also observe that the performance drop of our approach is much smaller than other baseline models.** We attribute this to the effectiveness of *CodeSelector* for putting relevant snippet on higher positions than the irrelevant ones, this also verifies the importance and necessity of our *CodeSelector* in the second stage.

5.5.4 Qualitative Analysis

Experimental Setup In this work, we aim to alleviate the *query mismatch* and *information overload* problems by using the *QueryRewriter* and *CodeSelector* respectively. To vividly demonstrate the workflow details of our model (i.e., from generating paraphrase

Table 5.13: User Study on Code Usefulness

Measure	Low	Medium-	Medium+	High	Mean Score	P-value
Google	17.60%	23.60%	32.40%	26.40%	2.68	< 0.01
Stack Overflow	73.60%	18.00%	6.40%	2.00%	1.37	< 0.01
CROKAGE	30.80%	26.00%	28.40%	14.80%	2.27	< 0.01
1stRanked	28.40%	31.60%	23.20%	16.80%	2.28	< 0.01
Ours	12.00%	22.00%	34.00%	32.00%	2.86	—

questions, semantically-equivalent question retrieval and best code snippet selection), we further conduct a case study to manually investigate some questions used in the previous user study. Fig. 5.13 shows 5 examples from the previous user studies to demonstrate the detailed results. For each unanswered question, we present the intermediary results of the generated paraphrase questions, the top-5 ranked retrieved question, as well as the recommended code snippets generated by our approach. The words that do not appear in the original query question, but include both in the generated paraphrase questions and the target retrieved questions are highlighted in yellow color. We also highlight the question in boldface which provides the recommended code snippet.

Experimental Results From the cases demonstrated in Fig 5.13, we can see that:

1. **The paraphrase questions generated by our *QueryRewriter* are meaningful for the given user query question.** Note that the *QueryRewriter* automatically transforms the user query question into different forms of semantically-equivalent user expressions. For example, in the third case, the original user query question is about “*Diagnose python memory usage*”, and our approach generates multiple paraphrase questions such as “*python memory management*”, “*using a python module to monitor memory usage*”, “*memory leak on python object*”. These generated paraphrase questions can be viewed as meaningful outputs to capture the developer’s intent and used as a way of question boosting for the original user query question.

Query Question	Generated Paraphrase Question	Retrieved Questions	Recommend Code Snippet
Ex1. (12702617) automatic deletion of directory in linux (centos)	how to quickly remove directory in python	(27700114) How to delete a python directory effectively?	import shutil shutil.rmtree("path_to_dir")
	how to delete files in python with subdirectory	(43756284) How to remove a directory including all its files in python?	
	how to rename a directory with ipython script	(51080451) How do I delete a specified number of files in a directory in Python?	
	setting a directory for deletion in python	(36156426) How rename files in a directory using python?	
	how to delete a directory in python2.7 on centos?	(31223312) how to rename file in a directory using python	
Ex2. (12745563) use of plot3d in python	plotting 3d vector using matplotlib	(47806784) how to plot a 3d graph in python using matplotlib ?	import matplotlib.pyplot as plt from sklearn.datasets import make_s_curve from mpl_toolkits.mplot3d import Axes3D # make and plot 3d X, y = make_s_curve(n_samples=1000) ax = plt.axes(projection='3d') ax.scatter3D(X[:, 0], X[:, 1], X[:, 2], c=y) ax.view_init(10, -60) plt.show()
	how to plot 3d on a computer	(4802157) How to make 3D plots in Python?	
	use the 3d symbol when plotting with matplotlib	(49561740) How to make a 3D data surface plot using matplotlib in python	
	use of 3d plots in python	(25286811) How to plot a 3D density map in python with matplotlib	
	pyplot 3d plots, variables	(36811960) How do I plot a 2D array graph in Python using matplotlib	
Ex3. (13090635) Diagnose python memory usage	python memory management	(552744) How do I profile memory usage in Python?	# heaps is quite simple to use from guppy import hpy h = hpy() print(h.heap()) # This gives you some output like this: ... Index Count % Size % Cumulative % Kind 0 25773 53 1612820 49 1612820 49 str 1 11699 24 483960 15 2096780 64 tuple ...
	using a python module to monitor memory usage	(43737948) How do I determine the memory usage of a python type?	
	memory leak on python object	(21701434) How to manage memory error in python?	
	python memory error	(11596371) How Does Python Memory Management Work?	
	how can i easily determine memory usage of python packages?	(55392166) How to detect memory leak in python code?	
Ex4. (3201964) merging 2 columns in a csv file through python	how to merge dataframe columns?	(38049548) How to merge two columns into one in csv format (Python Pandas)?	A, B = [], [] with open(your_file) as f: for line in f: if ...: A.append(line.split(your_seperator)) else: B.append(line.split(your_seperator)) A = pd.DataFrame(A, columns = list_of_columns) B = pd.DataFrame(B, columns = list_of_columns_2) .drop(columns_to_drop, 1) df = pd.concat([A, B]).reset_index(drop = True)
	merging dataframe rows of different lengths	(31384177) How to merge two pandas DataFrames in Python?	
	using pandas to merge two column in a file	(37894654) How to merge dataframes using pandas python?	
	merge text data in csv file using python	(37697195) how to merge two data frames based on particular column in pandas python?	
	pandas merging 101	(50708959) How to merge two columns from a dataframe	
Ex5. (5839210) is python faster than php?	speed of python vs .php	(5497540) How to call a Python Script from PHP?	SactivateScript = \$_GET['activeScript']; exec("python / path / to / file.py SactivateScript");
	python vs php: which is faster ?	(36859666) How run python with html or php?	
	python over php	(1686192) How fast is Python?	
	how php compares to python's model of program	(1060436) How do I include a PHP script in Python?	
	are python two times faster than php	(16288021) How to combine php & python code in Python	

Figure 5.13: Qualitative Analysis

2. **It is clear that adding the paraphrase questions can reduce the lexical gap between different user expressions**, which increases the likelihood of retrieving the semantic relevant questions in Stack Overflow. As shown in the second case, the developer formulate his/her problem as a user query “*use of plot3d in python*”, the generated paraphrase question “*plotting 3d vector using matplotlib*” can add missing information for the user query question and better link to the target semantically-equivalent question in Stack Overflow (i.e., “*how to plot a 3d graph in python using matplotlib*”), which verify the ability of our QueryRewriter to alleviate the *query mismatch* problem.
3. **A large number of code snippets recommended by our system are relevant and useful with respect to the user query question.** Some code snippets can well satisfy the developer’s programming tasks directly. For example, as shown in the first case of Fig. 5.13, the developer’s query question “*automatic deletion of directory in linux (centos)*” can be successfully solved by the code snippets within a relevant question “*How to remove a directory including all its files in python?*”. This verifies the effectiveness and possibility of our approach for recommending code solutions from the existing historical answers.
4. **We also notice that the recommended code snippets are not always selected from the first ranked question candidate.** Instead of naively choosing the first ranked code snippet like the Google search engine and Stack Overflow search engine, our *CodeSelector* selects best code snippet among a set of code snippet candidates via pairwise comparisons, which justify the ability of our *CodeSelector* to alleviate the *information overload* problem.
5. However, **the recommended code snippets from our system are not always useful.** For example, in the second case, the developer would like to inquire about “*use of*

plot3d in python”, our recommended code snippet is about “plot 3d graph using matplotlib”, which can be viewed as helpful by looking at the intent of the developer. In the fourth sample, even though the recommended code snippet can not be applied directly, it can be easily adapted to the user query question with minor modification.

6. Also, **the recommended code snippets from our system are not always relevant.**

For example, in the last sample, even though the generated paraphrase questions are meaningful and relevant to the user query question, the final recommended code snippet is still irrelevant to the problem described in the query question. This is because some user queries posted by developers are often complex and sophisticated; there may not exist semantically-equivalent questions that to the given one. It is thus very hard to search the query-specific code snippet to solve the corresponding problem.

Overall, our approach is more effective for retrieving relevant questions and searching useful code snippets compared with other baselines under human evaluation.

5.6 Practical Usage

The experiment was conducted on an Nvidia GeForce GTX 2080 GPU with 12GB memory. The time cost of our approach is mostly for the training process which takes approximately 20 to 24 hours for training Python and Java datasets respectively. However, after finishing the training process, each question title and code snippet in our data base can be converted into vector representations, which is highly efficient for later computing and searching processes. For example, the searching process on 5,000 examples takes five to eight minutes, while searching a single code snippet only costs 60 to 80ms.

Considering that searching code snippets in Stack Overflow with our approach is efficient, we have implemented QUE2CODE as a prototype web-based tool, which can facilitate developers in using our approach and inspire follow up research. Fig. 5.14 shows

the web interface of QUE2CODE. Developers can type or paste their query question into our web application, after that QUE2CODE goes through the question retrieval and code snippet reranking process, and recommends the top ranked questions and code snippets to the developers. We below describe the details of the input and output of our tool.

- **Input:** the input to the QUE2CODE is a user query question, which is a sequence of tokens. The input box in Fig. 5.14 shows an example of the user query question, i.e., “*how to find the most frequent item in array*”. After inputting the user query question, the developer can click the “Search” button to submit their query.
- **Output:** the output of the QUE2CODE is two folds: relevant questions and code snippets. After the developer submits his/her query to the server, the QUE2CODE searches through the codebase and returns top 5 relevant code snippets with their associated question titles. The link to these question posts on Stack Overflow are also provided for user references. For example, the relevant question post “*how to find the most frequent string element in numpy array*” and its code snippet are retrieved from our database to guide developers for solving their problems. Developers can use our tool to quickly locate the potential solutions to their query programming tasks and have a better understanding of their problems.

The goal of our tool is helping developers effectively search code snippets from Stack Overflow to their programming tasks and saving their time to do so. This is by no means these code snippets can perfectly solve the developers’ problem. After browsing these code snippets, developers still need to manually modify these code snippets for further refactoring and testing.

Overall, our approach is efficient enough for practical use and we have implemented a web service tool, QUE2CODE, to apply our approach for practical use.

Code Search in StackOverflow

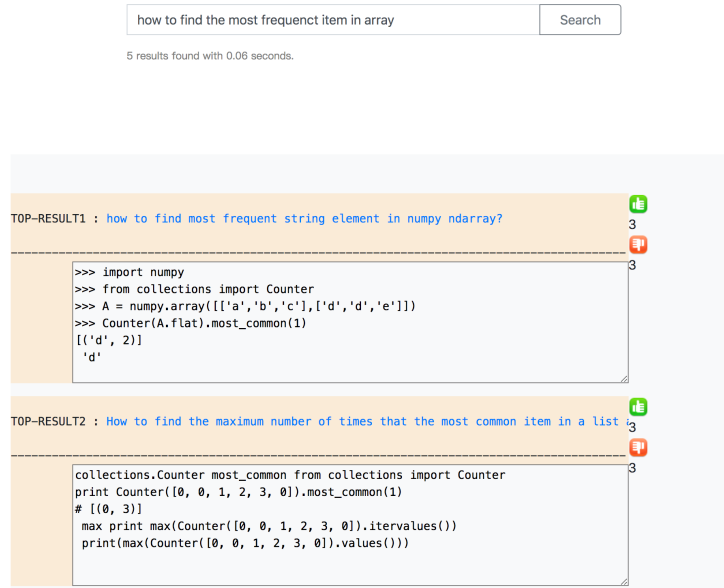


Figure 5.14: Prototype of Que2Code

5.7 Discussion

We discuss the strengths and weaknesses of our approach as well as the threats to validity for our experiments.

5.7.1 Strengths of Our Approach

To address the *query mismatch* and *information overload* problem in the Stack Overflow community, we proposed a novel query-driven code snippet recommendation model. Its key strengths are summarised below.

Paraphrase Question Generation A key advantage of our model is training a text-to-text transformer, named *QueryRewriter*, for generating paraphrase questions as a way of question boosting. This greatly improves the likelihood of our approach of retrieving semantically-equivalent questions for a user query question. By training on the duplicate

question pairs in Stack Overflow, for a user query question, *QueryRewriter* is able to generate different user descriptions for the same problem, which is helpful for our semantically-equivalent question retrieval tasks. The experimental results in Section 5.4.3 verify the effectiveness of adding paraphrased questions to our model.

Pairwise Learning to Rank To recommend the most relevant code snippets in Stack Overflow, we propose a novel pairwise learning to rank model in our work. Guided by our three heuristic rules, we can automatically construct the preference QC pairs and transform the code snippet recommendation task to a binary classification task. Rather than calculating a precise relevance score for a single QC pair, we estimate the preference relationship between two QC pairs. Ranking code snippets by pairwise comparison is more suitable for the code recommendation task in our study.

BERT model for embeddings BERT is designed to pre-train deep bidirectional representations from unlabeled text. It is conceptually simple and empirically powerful, which obtains new state-of-the-art results on eleven natural language processing tasks, such as question answering, language inference, etc. In our research, we investigated the BERT model for embedding query and code snippet pairs, and the ablation analysis in Section 5.4.6 demonstrates that it greatly enhanced the performance of our proposed model.

5.7.2 Threats to Validity

We have identified the following threats to validity: **Construct Validity** Internal validity relate to potential errors in our code implementation and experimental settings. To reduce errors in automatic evaluation, we have carefully tuned the parameters of the baseline approaches and used them in their highest performing settings for comparison, but there may still exist errors that we did not notice. Considering such cases, we have released the code and data of our research to facilitate other researchers to repeat our work and verify their ideas.

External Validity Threats to external validity are concerned with the generalizability of our dataset. Our dataset is collected from the official Stack Overflow data dump. We focus on two popular programming languages, i.e., Python and Java for our experiment. However, there are still many other programming languages which are not considered in our study. We believe that our model will generalize to other programming languages due to the effectiveness and robustness of our approach. We will try to extend our approach to other programming languages to benefit more developers in future studies. Besides, the data in our work is up to the release date of the official data dump, we can't automatically update the newly added data after the release date. However, this constraint can be supported by adding the engineering work. We will try to expand our approach to automatically handle the newly added data in our future work.

Model Validity The model validity relates to model structure that could affect the learning performance of our approach. In the first stage, we choose an encoder-decoder architecture for our *QueryRewriter*. Such an encoder-decoder architecture targets the sequence-to-sequence learning problem, which requires a large amount of manually labeled duplicate question pairs. However, our model may not generalize to other technical Q&A sites if the training set is limited. In the second stage, we choose the basic BERT model as our embedding layer due to its promising results on a wide range of NLP tasks [Devlin et al., 2018]. Recent research has proposed new models, such as GPT [Radford et al., 2018], RoBERTa [Liu et al., 2019], DistilBERT [Sanh et al., 2019], ALBERT [Lan et al., 2019] that can achieve better performance than BERT and/or similar performance with much less parameters. However, our results do not shed light on the effectiveness of employing other deep learning models with respect to different structures and new advanced features. We will try to use other deep learning models for our tasks in future work and compare them to the results that we report in this paper.

5.8 Related Work

5.8.1 Code Search in Software Engineering

The goal of code search is to find code fragments from a large code repository that most closely match a developer’s intent. Many code search methods have been proposed in the literature [Gu et al., 2018, Cambronero et al., 2019, Sachdev et al., 2018, Ye et al., 2016, Lu et al., 2015, Lv et al., 2015, Moreno et al., 2015, Ponzanelli et al., 2016]. Existing code search methods can be classified into two mainstreams: Information Retrieval-based methods and Deep Learning-based methods.

Ye et al. [Ye et al., 2016] proposed a model to fill the gap between natural-language queries and code snippets by projecting them into the same high dimensional vector space. Sachdev et al. [Sachdev et al., 2018] proposed a neural code search method which combined the of token-level embeddings and conventional information retrieval techniques TF-IDF. They found that the basic word embedding techniques can achieve good performance on code search task. Gu et al. [Gu et al., 2018] proposed a supervised technique, named DeepCS, for code searching using deep neural networks. They used multiple sequence-to-sequence-based networks to capture the features of the natural language queries and the code snippets.

The above code search methods are designed to measure the relevance degree between an individual QC (natural language query-code snippet) pair. However, in our study, rather than modeling a single QC pair to predict the precise relevance score, we model the preference relationship between two QC pairs. In other words, our model not only considers the relevance between a query and a code fragment, but also investigates the preference relationship between different QC pairs.

5.8.2 Duplicated Questions in Stack Overflow

The quality of the user-generated content is a key factor to attract users to visit the CQA sites, such as Stack Overflow. Prior work suggests that a quality decay problem occurs in these CQA community due to the growth in the number of duplicate questions [Srba and Bielikova, 2016]. This makes finding answers to a question harder and may dilute quality of answers. To maintain the quality of posts in Stack overflow, many studies have investigated the duplicate questions in Stack Overflow [Zhang et al., 2015, Silva et al., 2018, Ahasanuzzaman et al., 2016, Wang et al., 2020a, Mizobuchi and Takayama, 2017, Zhang et al., 2017].

Zhang et al. [Zhang et al., 2015] proposed an approach, named DupPredictor, to predict whether a question is duplicate question in Stack Overflow. They considered multiple factors, such as similarity scores of topics, titles, descriptions and tags for each question pair and calculated an overall similarity score by combining these features. Followed by their research, Ahasanuzzaman et al. [Ahasanuzzaman et al., 2016] first manually investigated why duplicate questions are asked by users in Stack Overflow, then proposed Dupe, which extracted features from question corpus and then built a binary classifier to judge if a question pair is duplicated or not. More recently, Wang et al. [Wang et al., 2020a] presented a deep-learning based approach to detect duplicate questions in Stack Overflow, which can capture the document-level and word-level semantic information respectively.

In our work, instead of considering the negative aspect of the duplicated questions in Stack Overflow, we consider duplicated question as semantically-equivalent questions pairs. We then train a query rewriting model for retrieving relevant questions in Stack Overflow.

5.8.3 Query Reformulation in Software Engineering

The effectiveness of code search heavily relies on the quality of the search query. If a query performs poorly, searching useful code snippets become increasingly difficult. Therefore it is necessary to reformulate and/or improve the user query when the query is poorly expressed. This need has motivated researchers to investigate the query reformulation (or expansion) approaches for software engineering tasks [Sirres et al., 2018, Rahman and Roy, 2018, Rahman et al., 2019, Cao et al., 2021, Shepherd et al., 2007, Haiduc et al., 2013, Hill et al., 2014, Lu et al., 2015, Nie et al., 2016, Jiang et al., 2016].

Shepherd et al. [Shepherd et al., 2007] presented an approach, V-DO, that automatically extracts verbs and objects from source code comments for misspelled query terms. Haiduc et al. [Haiduc et al., 2013] developed a query reformulation strategy by performing machine learning on a set of historical queries and relevant results. Following that, Hill et al. [Hill et al., 2014] proposed a query expansion tool, named *Conquer*, which combines the V-DO and contextual searching technique to suggest alternative query words. Lu et al. [Lu et al., 2015] implemented an approach to expand a query by using synonyms with the help of Wordnet. Nie et al. [Nie et al., 2016] proposed a model, named *QECK*, to identify the software-specific expansion words from the high quality pseudo feedback on Stack Overflow and generate expansion queries. After that, Rahman et al. [Rahman et al., 2019] proposed a query reformulation approach that suggests a list of relevant API classes for code search. Most recently, Cao et al. [Cao et al., 2021] proposed an automated deep-learning based query reformulation approach by using the query logs in Stack Overflow.

Different from the existing query expansion approaches, in this study, we first investigate the possibility of using duplicate question pairs from Stack Overflow for query rewriting. Our *QueryRewriter* can capture features between semantically equivalent questions and address the query mismatch problem.

5.8.4 Question Answering in CQA Sites

Finding similar questions and/or appropriate answers from historical archives has been applied in CQA sites. Great effort has been dedicated to various tasks such as question retrieval [Wang et al., 2009, Cao et al., 2010, Ganguly and Jones, 2015, Ye et al., 2014, Zou et al., 2015, Xu et al., 2018], answer selection [Xu et al., 2017, Gao et al., 2020b, Singh and Simperl, 2016], tagging [Zhou et al., 2019, Wang et al., 2015, González et al., 2015], and expert identification [Pal et al., 2012, Tian et al., 2013b, Kumar and Pedanekar, 2016].

Conventional techniques for retrieving answers primarily focus on complementary features of the CQA sites. Calefato et al. [Calefato et al., 2019] transform the answer selection task to a binary classification problem, they empirically evaluated 26 answer prediction model in Stack Overflow. Xu et al. [Xu et al., 2017] proposed a novel framework for generating relevant, useful and diverse answer summary for technical questions in Stack Overflow. Rather than directly ranking community answers, Tian et al. [Tian et al., 2013b] predicted the best answerer for a technical question in Stack Overflow by assuming that good respondents will give better answers. More recently, Gao et al. [Gao et al., 2020a] proposed a model for generating good question titles for developers by mining the code snippets in Stack Overflow.

Different from the aforementioned studies, we aim to search the best code fragment from the historical data in CQA database. We frame this task as a query-driven code recommendation task, and we propose a two stage framework to address the semantic-equivalent question retrieval and best code recommendation task respectively.

5.9 Summary

We have presented a fully data-driven approach, named `QUE2CODE`, for recommending the best code snippet in Stack Overflow for a user query question. We formulate this task as a query-driven code recommendation problem. Our proposed `QUE2CODE` model

contains two components: *QueryRewriter* and *CodeSelector*. In particular, we proposed a *QueryRewriter* for retrieving semantically-equivalent questions (as the first stage) and a *CodeSelector* for selecting the best code fragment in Stack Overflow (as the second stage). We have conducted extensive experiments to evaluate our approach on Stack Overflow dataset. Compared with several existing baselines, experimental results have comparatively demonstrated the effectiveness and superiority of our proposed model in both evaluation and human evaluation.

Chapter 6

Conclusion and Future Work

6.1 Key Contributions

This PhD project aimed to develop neural network methods for alleviating the *answer hungry* problem in SQA sites. Particularly, we present three different deep learning models to solve this problem from three different aspects:

First, we proposed a **sequence-to-sequence learning approach**, **CODE2QUE**, which can help developers in writing higher quality questions for a given code snippet. Stack Overflow has been heavily used by software developers as a popular way to seek programming-related information from peers via the internet. The Stack Overflow community recommends users to provide the related code snippet when they are creating a question to help others better understand it and offer their help. Previous studies have shown that a significant number of these questions are of low-quality and not attractive to other potential experts in Stack Overflow. These poorly asked questions are less likely to receive useful answers and hinder the overall knowledge generation and sharing process. Considering one of the reasons for introducing low-quality questions in SO is that many developers may not be able to clarify and summarize the key problems behind their presented code snippets due to their lack of knowledge and terminology related to the problem, and/or their poor writing skills, in this study we propose an approach to assist developers in writing high-quality questions by automatically generating question titles for a code snippet

using a deep sequence-to-sequence learning approach. Our approach is fully data-driven and uses an *attention* mechanism to perform better content selection, a *copy* mechanism to handle the rare-words problem and a *coverage* mechanism to eliminate word repetition problem. We evaluate our approach on Stack Overflow datasets over a variety of programming languages (e.g., Python, Java, Javascript, C# and SQL) and our experimental results show that our approach significantly outperforms several state-of-the-art baselines in both automatic and human evaluation. We summarise the following key contributions for this work: (i) We first proposed the a novel task to improve the low-quality question titles in Stack Overflow. (ii) We developed a tool, named CODE2QUE, to assist developers in writing high-quality question titles from a given code snippet. The human evaluation results show that our model can improve the Stack Overflow low-quality question titles in terms of *Clearness*, *Fitness* and *Willingness*.

Second, we proposed a **semi-supervised neural network**, DEEPANS, which can help developers identify the most relevant answers among a set of answer candidates. Our approach follows a three-stage process: question boosting, label establishment, and answer recommendation. Given a post, we first generate a clarifying question as a way of question boosting. We automatically establish the *positive*, *neutral*⁺, *neutral*⁻ and *negative* training samples via label establishment. When it comes to answer recommendation, we sort answer candidates by the matching scores calculated by our neural network-based model. To evaluate the performance of our proposed model, we conducted a large scale evaluation on four datasets, collected from the real world technical Q&A sites (i.e., Ask Ubuntu, Super User, Stack Overflow Python and Stack Overflow Java). Our experimental results show that our approach significantly outperforms several state-of-the-art baselines in automatic evaluation. We also conducted a user study with 50 solved/unanswered/unresolved questions. The user study results demonstrate that our approach is effective in solving the answer hungry problem by recommending the most relevant answers from historical

archives. We summarise the key contributions for this work as follows: (i) We developed a novel weakly supervised neural network, named DEEPANS, to find the potential solutions for unanswered/unresolved questions in SQA sites. (ii) We first use the clarifying questions in SQA sites to fill the gap between questions and answers as a way of question boosting.

Third, **we proposed a query-driven code search engine, QUE2CODE**, which can help developers effectively identify the code solutions to their programming tasks. More and more developers use SQA forums, such as Stack Overflow, to search for code examples of how to accomplish a certain coding task. This is often considered to be more efficient than working from source documentation, tutorials or full worked examples. However, due to the complexity of these online Question and Answer forums and the very large volume of information they contain, developers can be overwhelmed by the sheer volume of available information. This makes it hard to find and/or even be aware of the most relevant code examples to meet their needs. To alleviate this issue, in this work we present a query-driven code recommendation tool, named QUE2CODE, that identifies the best code snippets for a user query from Stack Overflow posts. Our approach has two main stages: (i) semantically-equivalent question retrieval and (ii) best code snippet recommendation. During the first stage, for a given query question formulated by a developer, we first generate paraphrase questions for the input query as a way of query boosting, and then retrieve the relevant Stack Overflow posted questions based on these generated questions. In the second stage, we collect all of the code snippets within questions retrieved in the first stage and develop a novel scheme to rank code snippet candidates from Stack Overflow posts via pairwise comparisons. To evaluate the performance of our proposed model, we conduct a large scale experiment to evaluate the effectiveness of the semantically-equivalent question retrieval task and best code snippet recommendation task separately on Python and Java datasets in Stack Overflow. We also perform a human study to measure how real-world developers perceive the results generated by our model. Both the automatic and human evaluation

results demonstrate the promising performance of our model over a set of state-of-the-art baselines. In this work, we make the following key contributions: (i) We first use the duplicate question pairs to retrieve the the semantically-equivalent questions in Stack Overflow. (ii) We developed a neural network based learning to rank model to search code snippets from Stack Overflow posts, which is able to calculate the matching score between a query and a code snippet.

6.2 Future Work

As for the future work, we have identified a few key research directions:

For our first task of generating high-quality question titles, a key challenge for our current work is that the question titles generated by our approach suffered from semantic drift. This is because it is difficult to judge a questioner’s intent by solely looking at the code snippet. For example, the developers may provide the same code snippet but ask different questions from different perspectives. For such cases, more relevant information, e.g., question description, question tags could further be incorporated with our model, which can help our model to generate a question title that is more accurate and precise. One of our future research direction is to **generate more meaningful question titles** by considering extra context information.

For our second task of finding potential answers for unanswered/unresolved questions, a key challenge for our current work is that we build our training set via four heuristic rules, there is no guarantee that all the training samples are correct. There still exists outlier cases distant from our heuristic rules. For example, we label all the accept answers as *Positive* samples and all the non-accept answers as *Neutral+* samples, which means we assume the all the accept answers are better than the non-accept answers. According to our observation, non-accept answers may be better than the accept answers which may conflict our heuristic rules. These outlier cases will produce a series of wrong training samples and hinder the

learning performance of our model. **Detecting and removing these outlier cases** before building our training samples will benefit the learning performance of our model, we will focus on this research direction in the future.

For our third research of searching code solutions for a user query, a key challenge for our current work is that we can only find code solutions if the user query has semantically-equivalent posts in Stack Overflow. If there are no posts relevant to the user query, a better way is to identify suitable experts for providing solutions for this problem. Therefore, how to route a user query to relevant experts is one of our future research direction. We **plan to design models for recommending appropriate domain experts** by considering the semantic of the user query as well as the user profiles of the potential experts.

6.3 Summary

In summary, this Ph.D. thesis focuses on the open issue of “*answer hungry*” problem in SQA sites. We developed three models, i.e., CODE2QUE, DEEPANS, QUE2CODE for improving the quality of question titles, recommending appropriate answers for unanswered/unresolved questions, and searching for useful code solutions for newly posted user queries respectively. We plan to enhance the performance of our existing models and design new models for identifying appropriate domain experts in our future work.

References

- [Adamic et al., 2008] Adamic, L. A., Zhang, J., Bakshy, E., and Ackerman, M. S. (2008). Knowledge sharing and yahoo answers: everyone knows something. In *Proceedings of the 17th international conference on World Wide Web*, pages 665–674. ACM.
- [Agarwal et al., 2012] Agarwal, A., Raghavan, H., Subbian, K., Melville, P., Lawrence, R. D., Gondek, D. C., and Fan, J. (2012). Learning to rank for robust question answering. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 833–842. ACM.
- [Ahasanuzzaman et al., 2016] Ahasanuzzaman, M., Asaduzzaman, M., Roy, C. K., and Schneider, K. A. (2016). Mining duplicate questions of stack overflow. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 402–412. IEEE.
- [Alahmadi et al., 2018] Alahmadi, M., Hassel, J., Parajuli, B., Haiduc, S., and Kumar, P. (2018). Accurately predicting the location of code fragments in programming video tutorials using deep learning. In *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*, pages 2–11. ACM.
- [Allamanis and Sutton, 2013] Allamanis, M. and Sutton, C. (2013). Why, when, and what: analyzing stack overflow questions by topic, type, and code. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 53–56. IEEE Press.

- [Allamanis et al., 2015] Allamanis, M., Tarlow, D., Gordon, A., and Wei, Y. (2015). Bimodal modelling of source code and natural language. In *International Conference on Machine Learning*, pages 2123–2132.
- [Anderson et al., 2012] Anderson, A., Huttenlocher, D., Kleinberg, J., and Leskovec, J. (2012). Discovering value from community activity on focused question answering sites: a case study of stack overflow. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 850–858.
- [Arora et al., 2015] Arora, P., Ganguly, D., and Jones, G. J. (2015). The good, the bad and their kins: Identifying questions with negative scores in stackoverflow. In *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 1232–1239. IEEE.
- [Asaduzzaman et al., 2013] Asaduzzaman, M., Mashiyat, A. S., Roy, C. K., and Schneider, K. A. (2013). Answering questions about unanswered questions of stack overflow. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 97–100. IEEE.
- [Azizan and Bakar, 2015] Azizan, A. and Bakar, Z. A. (2015). Query reformulation using crop characteristic in specific domain search. In *2015 IEEE European Modelling Symposium (EMS)*, pages 374–379. IEEE.
- [Ba et al., 2016] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.
- [Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

- [Baltadzhieva and Chrupała, 2015] Baltadzhieva, A. and Chrupała, G. (2015). Question quality in community question answering forums: a survey. *Acm Sigkdd Explorations Newsletter*, 17(1):8–13.
- [Bazelli et al., 2013] Bazelli, B., Hindle, A., and Stroulia, E. (2013). On the personality traits of stackoverflow users. In *2013 IEEE international conference on software maintenance*, pages 460–463. IEEE.
- [Bird and Loper, 2004] Bird, S. and Loper, E. (2004). Nltk: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics.
- [Bojanowski et al., 2017] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.
- [Bosu et al., 2013] Bosu, A., Corley, C. S., Heaton, D., Chatterji, D., Carver, J. C., and Kraft, N. A. (2013). Building reputation in stackoverflow: an empirical investigation. In *2013 10th working conference on mining software repositories (MSR)*, pages 89–92. IEEE.
- [Brandt et al., 2009] Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., and Klemmer, S. R. (2009). Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1589–1598.
- [Büch and Andrzejak, 2019] Büch, L. and Andrzejak, A. (2019). Learning-based recursive aggregation of abstract syntax trees for code clone detection. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 95–104. IEEE.

- [Bunel et al., 2018] Bunel, R., Hausknecht, M., Devlin, J., Singh, R., and Kohli, P. (2018). Leveraging grammar and reinforcement learning for neural program synthesis. *arXiv preprint arXiv:1805.04276*.
- [Burges, 2010] Burges, C. J. (2010). From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81.
- [Calefato et al., 2015] Calefato, F., Lanubile, F., Marasciulo, M. C., and Novielli, N. (2015). Mining successful answers in stack overflow. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 430–433. IEEE.
- [Calefato et al., 2016] Calefato, F., Lanubile, F., and Novielli, N. (2016). Moving to stack overflow: Best-answer prediction in legacy developer forums. In *Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement*, page 13. ACM.
- [Calefato et al., 2018] Calefato, F., Lanubile, F., and Novielli, N. (2018). How to ask for technical help? evidence-based guidelines for writing questions on stack overflow. *Information and Software Technology*, 94:186–207.
- [Calefato et al., 2019] Calefato, F., Lanubile, F., and Novielli, N. (2019). An empirical assessment of best-answer prediction models in technical q&a sites. *Empirical Software Engineering*, 24(2):854–901.
- [Cambronero et al., 2019] Cambronero, J., Li, H., Kim, S., Sen, K., and Chandra, S. (2019). When deep learning met code search. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 964–974.
- [Cao et al., 2021] Cao, K., Chen, C., Baltes, S., Treude, C., and Chen, X. (2021). Automated query reformulation for efficient search based on query logs from stack overflow.

In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1273–1285. IEEE.

[Cao et al., 2010] Cao, X., Cong, G., Cui, B., and Jensen, C. S. (2010). A generalized framework of exploring category information for question retrieval in community question answer archives. In *Proceedings of the 19th international conference on World wide web*, pages 201–210.

[Chang and Pal, 2013] Chang, S. and Pal, A. (2013). Routing questions for collaborative answering in community question answering. In *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013)*, pages 494–501. IEEE.

[Chen et al., 2018] Chen, C., Su, T., Meng, G., Xing, Z., and Liu, Y. (2018). From ui design image to gui skeleton: a neural machine translator to bootstrap mobile gui implementation. In *Proceedings of the 40th International Conference on Software Engineering*, pages 665–676.

[Chen et al., 2016] Chen, G., Chen, C., Xing, Z., and Xu, B. (2016). Learning a dual-language vector space for domain-specific cross-lingual question retrieval. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 744–755. IEEE.

[Chen and Zhou, 2018] Chen, Q. and Zhou, M. (2018). A neural framework for retrieval and summarization of source code. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 826–831. ACM.

[Chen et al., 2019] Chen, Z., Komrusch, S. J., Tufano, M., Pouchet, L.-N., Poshyanyk, D., and Monperrus, M. (2019). Sequencer: Sequence-to-sequence learning for end-to-end program repair. *IEEE Transactions on Software Engineering*.

- [Collobert et al., 2011] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537.
- [Correa and Sureka, 2013] Correa, D. and Sureka, A. (2013). Fit or unfit: analysis and prediction of ‘closed questions’ on stack overflow. In *Proceedings of the first ACM conference on Online social networks*, pages 201–212. ACM.
- [Correa and Sureka, 2014] Correa, D. and Sureka, A. (2014). Chaff from the wheat: Characterization and modeling of deleted questions on stack overflow. In *Proceedings of the 23rd international conference on World wide web*, pages 631–642.
- [da Silva et al., 2020] da Silva, R. F. G., Roy, C. K., Rahman, M. M., Schneider, K. A., Paixão, K., de Carvalho Dantas, C. E., and de Almeida Maia, M. (2020). Crokage: effective solution recommendation for programming tasks by leveraging crowd knowledge. *Empirical Software Engineering*, 25(6):4707–4758.
- [Dauphin et al., 2017] Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2017). Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941.
- [Desai et al., 2016] Desai, A., Gulwani, S., Hingorani, V., Jain, N., Karkare, A., Marron, M., Roy, S., et al. (2016). Program synthesis using natural language. In *Proceedings of the 38th International Conference on Software Engineering*, pages 345–356. ACM.
- [Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- [Dong et al., 2019] Dong, H., Liu, S., Han, S., Fu, Z., and Zhang, D. (2019). Tablesense: Spreadsheet table detection with convolutional neural networks.

- [Duijn et al., 2015] Duijn, M., Kucera, A., and Bacchelli, A. (2015). Quality questions need quality code: Classifying code fragments on stack overflow. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 410–413. IEEE.
- [Franks et al., 2015] Franks, C., Tu, Z., Devanbu, P., and Hellendoorn, V. (2015). Cacheca: A cache language model based code suggestion tool. In *Proceedings of the 37th International Conference on Software Engineering-Volume 2*, pages 705–708. IEEE Press.
- [Ganguly and Jones, 2015] Ganguly, D. and Jones, G. J. (2015). Partially labeled supervised topic models for retrieving similar questions in cqa forums. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, pages 161–170. ACM.
- [Gantayat et al., 2015] Gantayat, N., Dhoolia, P., Padhye, R., Mani, S., and Sinha, V. S. (2015). The synergy between voting and acceptance of answers on stackoverflow-or the lack thereof. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 406–409. IEEE.
- [Gao et al., 2019a] Gao, Y., Wang, Z., Liu, S., Yang, L., Sang, W., and Cai, Y. (2019a). Teccd: A tree embedding approach for code clone detection. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 145–156. IEEE.
- [Gao, 2020] Gao, Z. (2020). Dataset for the paper: Generating Question Titles for Stack Overflow from Mined Code Snippets.
- [Gao et al., 2019b] Gao, Z., Jayasundara, V., Jiang, L., Xia, X., Lo, D., and Grundy, J. (2019b). Smartembed: A tool for clone and bug detection in smart contracts through structural code embedding. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 394–397. IEEE.

- [Gao et al., 2020] Gao, Z., Jiang, L., Xia, X., Lo, D., and Grundy, J. (2020). Checking smart contracts with structural code embedding. *IEEE Transactions on Software Engineering*, pages 1–1.
- [Gao et al., 2020a] Gao, Z., Xia, X., Grundy, J., Lo, D., and Li, Y.-F. (2020a). Generating question titles for stack overflow from mined code snippets. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 29(4):1–37.
- [Gao et al., 2020b] Gao, Z., Xia, X., Lo, D., and Grundy, J. (2020b). Technical q&a site answer recommendation via question boosting. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(1):1–34.
- [Gao et al., 2020c] Gao, Z., Xia, X., Lo, D., Grundy, J., and Li, Y.-F. (2020c). Code2que: A tool for improving question titles from mined code snippets in stack overflow. *arXiv preprint arXiv:2007.10851*.
- [Ginsca and Popescu, 2013] Ginsca, A. L. and Popescu, A. (2013). User profiling for answer quality assessment in q&a communities. In *Proceedings of the 2013 workshop on Data-driven user behavioral modelling and mining from social media*, pages 25–28.
- [Giordani and Moschitti, 2009] Giordani, A. and Moschitti, A. (2009). Semantic mapping between natural language questions and sql queries via syntactic pairing. In *International Conference on Application of Natural Language to Information Systems*, pages 207–221. Springer.
- [Gkotsis et al., 2014] Gkotsis, G., Stepanyan, K., Pedrinaci, C., Domingue, J., and Liakata, M. (2014). It’s all in the content: state of the art best answer prediction based on discretisation of shallow linguistic features. In *Proceedings of the 2014 ACM conference on Web science*, pages 202–210. ACM.

- [González et al., 2015] González, J. R. C., Romero, J. J. F., Guerrero, M. G., and Calderón, F. (2015). Multi-class multi-tag classifier system for stackoverflow questions. In *2015 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*, pages 1–6. IEEE.
- [Grant and Betts, 2013] Grant, S. and Betts, B. (2013). Encouraging user behaviour with achievements: an empirical study. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 65–68. IEEE.
- [Graves, 2012] Graves, A. (2012). Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*.
- [Gu et al., 2016a] Gu, J., Lu, Z., Li, H., and Li, V. O. (2016a). Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*.
- [Gu et al., 2018] Gu, X., Zhang, H., and Kim, S. (2018). Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 933–944. IEEE.
- [Gu et al., 2016b] Gu, X., Zhang, H., Zhang, D., and Kim, S. (2016b). Deep api learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 631–642.
- [Gu et al., 2017] Gu, X., Zhang, H., Zhang, D., and Kim, S. (2017). Deepam: Migrate apis with multi-modal sequence to sequence learning. *arXiv preprint arXiv:1704.07734*.
- [Haiduc et al., 2013] Haiduc, S., Bavota, G., Marcus, A., Oliveto, R., De Lucia, A., and Menzies, T. (2013). Automatic query reformulations for text retrieval in software engineering. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 842–851. IEEE.

- [Halavais et al., 2014] Halavais, A., Kwon, K. H., Havener, S., and Striker, J. (2014). Badges of friendship: Social influence and badge acquisition on stack overflow. In *2014 47th Hawaii international conference on System Sciences*, pages 1607–1615. IEEE.
- [Hanrahan et al., 2012] Hanrahan, B. V., Convertino, G., and Nelson, L. (2012). Modeling problem difficulty and expertise in stackoverflow. In *Proceedings of the ACM 2012 conference on computer supported cooperative work companion*, pages 91–94.
- [Hart and Sarma, 2014] Hart, K. and Sarma, A. (2014). Perceptions of answer quality in an online technical question and answer forum. In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 103–106.
- [Hashemi et al., 2020] Hashemi, H., Aliannejadi, M., Zamani, H., and Croft, W. B. (2020). Antique: A non-factoid question answering benchmark. In *European Conference on Information Retrieval*, pages 166–173. Springer.
- [He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- [He et al., 2017] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. (2017). Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182.
- [Heafield, 2011] Heafield, K. (2011). Kenlm: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197. Association for Computational Linguistics.
- [Henß et al., 2012] Henß, S., Monperrus, M., and Mezini, M. (2012). Semi-automatically extracting faqs to improve accessibility of software development knowledge. In *Pro-*

- ceedings of the 34th International Conference on Software Engineering*, pages 793–803. IEEE Press.
- [Hill et al., 2014] Hill, E., Roldan-Vega, M., Fails, J. A., and Mallet, G. (2014). NI-based query refinement and contextualized code search results: A user study. In *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 34–43. IEEE.
- [Hoang et al., 2019] Hoang, T., Dam, H. K., Kamei, Y., Lo, D., and Ubayashi, N. (2019). Deepjit: an end-to-end deep learning framework for just-in-time defect prediction. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 34–45. IEEE.
- [Hu et al., 2018] Hu, X., Li, G., Xia, X., Lo, D., and Jin, Z. (2018). Deep code comment generation. In *Proceedings of the 26th Conference on Program Comprehension*, pages 200–210. ACM.
- [Huang et al., 2020] Huang, Q., Xia, X., Lo, D., and Murphy, G. C. (2020). Automating intention mining. *IEEE Transactions on Software Engineering*, 46(10):1098–1119.
- [Huang et al., 2018] Huang, Q., Yang, Y., Zhan, X., Wan, H., and Wu, G. (2018). Query expansion based on statistical learning from code changes. *Software: Practice and Experience*, 48(7):1333–1351.
- [Husain et al., 2019] Husain, H., Wu, H.-H., Gazit, T., Allamanis, M., and Brockschmidt, M. (2019). Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.
- [Iyer et al., 2016] Iyer, S., Konstas, I., Cheung, A., and Zettlemoyer, L. (2016). Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual*

Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), volume 1, pages 2073–2083.

[Jenders et al., 2016] Jenders, M., Krestel, R., and Naumann, F. (2016). Which answer is best?: Predicting accepted answers in mooc forums. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 679–684. International World Wide Web Conferences Steering Committee.

[Jiang et al., 2016] Jiang, H., Nie, L., Sun, Z., Ren, Z., Kong, W., Zhang, T., and Luo, X. (2016). Rosf: Leveraging information retrieval and supervised learning for recommending code snippets. *IEEE Transactions on Services Computing*, 12(1):34–46.

[Jiang et al., 2017] Jiang, S., Armaly, A., and McMillan, C. (2017). Automatically generating commit messages from diffs using neural machine translation. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pages 135–146. IEEE Press.

[Jiarpakdee et al., 2016] Jiarpakdee, J., Ihara, A., and Matsumoto, K.-i. (2016). Understanding question quality through affective aspect in q&a site. In *2016 IEEE/ACM 1st International Workshop on Emotional Awareness in Software Engineering (SEmotion)*, pages 12–17. IEEE.

[Jin and Servant, 2019] Jin, X. and Servant, F. (2019). What edits are done on the highly answered questions in stack overflow? an empirical study. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 225–229. IEEE.

[Kalchbrenner et al., 2014] Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

- [Keivanloo et al., 2014] Keivanloo, I., Rilling, J., and Zou, Y. (2014). Spotting working code examples. In *Proceedings of the 36th International Conference on Software Engineering*, pages 664–675. ACM.
- [Kim et al., 2019] Kim, J., Feldt, R., and Yoo, S. (2019). Guiding deep learning system testing using surprise adequacy. In *Proceedings of the 41st International Conference on Software Engineering*, pages 1039–1049. IEEE Press.
- [Kim et al., 2018] Kim, J., Kwon, M., and Yoo, S. (2018). Generating test input with deep reinforcement learning. In *2018 IEEE/ACM 11th International Workshop on Search-Based Software Testing (SBST)*, pages 51–58. IEEE.
- [Kim, 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.
- [Koehn, 2004] Koehn, P. (2004). Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *Conference of the Association for Machine Translation in the Americas*, pages 115–124. Springer.
- [Koehn et al., 2007] Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., et al. (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics.
- [Kumar and Pedanekar, 2016] Kumar, V. and Pedanekar, N. (2016). Mining shapes of expertise in online social q&a communities. In *Proceedings of the 19th ACM conference on computer supported cooperative work and social computing companion*, pages 317–320.

- [Lai et al., 2015] Lai, S., Xu, L., Liu, K., and Zhao, J. (2015). Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*.
- [Lam et al., 2017] Lam, A. N., Nguyen, A. T., Nguyen, H. A., and Nguyen, T. N. (2017). Bug localization with combination of deep learning and information retrieval. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 218–229. IEEE.
- [Lan et al., 2019] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.
- [Le and Mikolov, 2014] Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196.
- [LeClair et al., 2019] LeClair, A., Jiang, S., and McMillan, C. (2019). A neural model for generating natural language summaries of program subroutines. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 795–806. IEEE.
- [Lezina and Kuznetsov, 2013] Lezina, C. G. E. and Kuznetsov, A. M. (2013). Predict closed questions on stackoverflow.
- [Li et al., 2012] Li, B., Jin, T., Lyu, M. R., King, I., and Mak, B. (2012). Analyzing and predicting question quality in community question answering services. In *Proceedings of the 21st International Conference on World Wide Web*, pages 775–782.
- [Li et al., 2019a] Li, H., Kim, S., and Chandra, S. (2019a). Neural code search evaluation dataset. *arXiv preprint arXiv:1908.09804*.

- [Li et al., 2017] Li, L., Feng, H., Zhuang, W., Meng, N., and Ryder, B. (2017). Cclearner: A deep learning-based clone detection approach. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 249–260. IEEE.
- [Li et al., 2019b] Li, X., Li, W., Zhang, Y., and Zhang, L. (2019b). Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 169–180.
- [Li et al., 2016] Li, Z., Wang, T., Zhang, Y., Zhan, Y., and Yin, G. (2016). Query reformulation by leveraging crowd wisdom for scenario-based software search. In *Proceedings of the 8th Asia-Pacific Symposium on Internetware*, pages 36–44. ACM.
- [Lin, 2004] Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*.
- [Ling et al., 2016] Ling, W., Grefenstette, E., Hermann, K. M., Kočiskỳ, T., Senior, A., Wang, F., and Blunsom, P. (2016). Latent predictor networks for code generation. *arXiv preprint arXiv:1603.06744*.
- [Liu et al., 2013] Liu, J., Wang, Q., Lin, C.-Y., and Hon, H.-W. (2013). Question difficulty estimation in community question answering services. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 85–90.
- [Liu et al., 2017] Liu, P., Zhang, X., Pistoia, M., Zheng, Y., Marques, M., and Zeng, L. (2017). Automatic text input generation for mobile testing. In *Proceedings of the 39th International Conference on Software Engineering*, pages 643–653. IEEE Press.
- [Liu et al., 2019] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pre-training approach. *arXiv preprint arXiv:1907.11692*.

- [Liu et al., 2018] Liu, Z., Xia, X., Hassan, A. E., Lo, D., Xing, Z., and Wang, X. (2018). Neural-machine-translation-based commit message generation: how far are we? In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 373–384.
- [Locascio et al., 2016] Locascio, N., Narasimhan, K., DeLeon, E., Kushman, N., and Barzilay, R. (2016). Neural generation of regular expressions from natural language with minimal domain knowledge. *arXiv preprint arXiv:1608.03000*.
- [Lu et al., 2015] Lu, M., Sun, X., Wang, S., Lo, D., and Duan, Y. (2015). Query expansion via wordnet for effective code search. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 545–549. IEEE.
- [Lv et al., 2015] Lv, F., Zhang, H., Lou, J.-g., Wang, S., Zhang, D., and Zhao, J. (2015). Codehow: Effective code search based on api understanding and extended boolean model (e). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 260–270. IEEE.
- [Mamykina et al., 2011] Mamykina, L., Manoim, B., Mittal, M., Hripcsak, G., and Hartmann, B. (2011). Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 2857–2866.
- [Manning et al., 2005] Manning, C. D., Raghavan, P., and Schütze, H. (2005). Introduction to information retrieval.
- [Mesbah et al., 2019] Mesbah, A., Rice, A., Johnston, E., Glorioso, N., and Aftandilian, E. (2019). Deepdelta: learning to repair compilation errors. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 925–936.

- [Mikolov et al., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.
- [Mizobuchi and Takayama, 2017] Mizobuchi, Y. and Takayama, K. (2017). Two improvements to detect duplicates in stack overflow. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 563–564. IEEE.
- [Moran et al., 2018] Moran, K., Bernal-Cárdenas, C., Curcio, M., Bonett, R., and Poshyvanyk, D. (2018). Machine learning-based prototyping of graphical user interfaces for mobile apps. *IEEE Transactions on Software Engineering*, 46(2):196–221.
- [Moreno et al., 2015] Moreno, L., Bavota, G., Di Penta, M., Oliveto, R., and Marcus, A. (2015). How can i use this method? In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 880–890. IEEE.
- [Murgia et al., 2016] Murgia, A., Janssens, D., Demeyer, S., and Vasilescu, B. (2016). Among the machines: Human-bot interaction on social q&a websites. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 1272–1279.
- [Nasehi et al., 2012] Nasehi, S. M., Sillito, J., Maurer, F., and Burns, C. (2012). What makes a good code example?: A study of programming q&a in stackoverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 25–34. IEEE.
- [Nie et al., 2016] Nie, L., Jiang, H., Ren, Z., Sun, Z., and Li, X. (2016). Query expansion based on crowd knowledge for code search. *IEEE Transactions on Services Computing*, 9(5):771–783.

- [Nie et al., 2017] Nie, L., Wei, X., Zhang, D., Wang, X., Gao, Z., and Yang, Y. (2017). Data-driven answer selection in community qa systems. *IEEE transactions on knowledge and data engineering*, 29(6):1186–1198.
- [Novielli et al., 2014] Novielli, N., Calefato, F., and Lanubile, F. (2014). Towards discovering the role of emotions in stack overflow. In *Proceedings of the 6th international workshop on social software engineering*, pages 33–36.
- [Oda et al., 2015] Oda, Y., Fudaba, H., Neubig, G., Hata, H., Sakti, S., Toda, T., and Nakamura, S. (2015). Learning to generate pseudo-code from source code using statistical machine translation (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 574–584. IEEE.
- [Pal et al., 2012] Pal, A., Harper, F. M., and Konstan, J. A. (2012). Exploring question selection bias to identify experts and potential experts in community question answering. *ACM Transactions on Information Systems (TOIS)*, 30(2):1–28.
- [Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.
- [Ponzanelli et al., 2016] Ponzanelli, L., Bavota, G., Mocci, A., Di Penta, M., Oliveto, R., Russo, B., Haiduc, S., and Lanza, M. (2016). Codetube: extracting relevant fragments from software development video tutorials. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pages 645–648. IEEE.
- [Ponzanelli et al., 2014] Ponzanelli, L., Mocci, A., Bacchelli, A., and Lanza, M. (2014). Understanding and classifying the quality of technical forum questions. In *Quality Software (QSIC), 2014 14th International Conference on*, pages 343–352. IEEE.

- [Posnett et al., 2012] Posnett, D., Warburg, E., Devanbu, P., and Filkov, V. (2012). Mining stack exchange: Expertise is evident from initial contributions. In *2012 International Conference on Social Informatics*, pages 199–204. IEEE.
- [Potdar and Shihab, 2014] Potdar, A. and Shihab, E. (2014). An exploratory study on self-admitted technical debt. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 91–100. IEEE.
- [Radford et al., 2018] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.
- [Raffel et al., 2019] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.
- [Rahman and Roy, 2018] Rahman, M. M. and Roy, C. (2018). Effective reformulation of query for code search using crowdsourced knowledge and extra-large data analytics. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 473–484. IEEE.
- [Rahman and Roy, 2015] Rahman, M. M. and Roy, C. K. (2015). An insight into the unresolved questions at stack overflow. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 426–429. IEEE.
- [Rahman et al., 2019] Rahman, M. M., Roy, C. K., and Lo, D. (2019). Automatic query reformulation for code search using crowdsourced knowledge. *Empirical Software Engineering*, 24(4):1869–1924.
- [Rao and Daumé III, 2018] Rao, S. and Daumé III, H. (2018). Learning to ask good questions: Ranking clarification questions using neural expected value of perfect information. In *Proceedings of the 56th Annual Meeting of the Association for Computational*

Linguistics (Volume 1: Long Papers), pages 2737–2746, Melbourne, Australia. Association for Computational Linguistics.

[Ravi et al., 2014] Ravi, S., Pang, B., Rastogi, V., and Kumar, R. (2014). Great question! question quality in community q&a. In *Eighth International AAAI Conference on Weblogs and Social Media*.

[Řehůřek and Sojka, 2010] Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. <http://is.muni.cz/publication/884893/en>.

[Rekha et al., 2014] Rekha, V. S., Divya, N., and Bagavathi, P. S. (2014). A hybrid auto-tagging system for stackoverflow forum questions. In *Proceedings of the 2014 International Conference on Interdisciplinary Advances in Applied Computing*, pages 1–5.

[Ren et al., 2019] Ren, X., Xing, Z., Xia, X., Lo, D., Wang, X., and Grundy, J. (2019). Neural network-based detection of self-admitted technical debt: From performance to explainability. *ACM transactions on software engineering and methodology (TOSEM)*, 28(3):1–45.

[Riahi et al., 2012] Riahi, F., Zolaktaf, Z., Shafiei, M., and Milios, E. (2012). Finding expert users in community question answering. In *Proceedings of the 21st international conference on world wide web*, pages 791–798.

[Robertson and Walker, 1994] Robertson, S. E. and Walker, S. (1994). Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 232–241. Springer-Verlag New York, Inc.

- [Sachdev et al., 2018] Sachdev, S., Li, H., Luan, S., Kim, S., Sen, K., and Chandra, S. (2018). Retrieval on source code: a neural code search. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 31–41.
- [Saha et al., 2013] Saha, A. K., Saha, R. K., and Schneider, K. A. (2013). A discriminative model approach for suggesting tags automatically for stack overflow questions. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 73–76. IEEE.
- [Sahu et al., 2016] Sahu, T. P., Nagwani, N. K., and Verma, S. (2016). Selecting best answer: An empirical analysis on community question answering sites. *IEEE Access*, 4:4797–4808.
- [Sanh et al., 2019] Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.
- [Sankaran et al., 2016] Sankaran, B., Mi, H., Al-Onaizan, Y., and Ittycheriah, A. (2016). Temporal attention model for neural machine translation. *arXiv preprint arXiv:1608.02927*.
- [Savenkov, 2015] Savenkov, D. (2015). Ranking answers and web passages for non-factoid question answering: Emory university at trec liveqa. In *TREC*.
- [See et al., 2017] See, A., Liu, P. J., and Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*.
- [Seljan et al., 2012] Seljan, S., Brkic, M., and Vicic, T. (2012). Bleu evaluation of machine-translated english-croatian legislation. In *LREC*, pages 2143–2148.

- [Sennrich et al., 2016] Sennrich, R., Haddow, B., and Birch, A. (2016). Edinburgh neural machine translation systems for wmt 16. *arXiv preprint arXiv:1606.02891*.
- [Shepherd et al., 2007] Shepherd, D., Fry, Z. P., Hill, E., Pollock, L., and Vijay-Shanker, K. (2007). Using natural language program analysis to locate and understand action-oriented concerns. In *Proceedings of the 6th international conference on Aspect-oriented software development*, pages 212–224.
- [Silva et al., 2018] Silva, R. F., Paixão, K., and de Almeida Maia, M. (2018). Duplicate question detection in stack overflow: A reproducibility study. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 572–581. IEEE.
- [Singh and Simperl, 2016] Singh, P. and Simperl, E. (2016). Using semantics to search answers for unanswered questions in q&a forums. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 699–706. International World Wide Web Conferences Steering Committee.
- [Sinha et al., 2015] Sinha, T., Wei, W., and Carley, K. (2015). Modeling similarity in incentivized interaction: A longitudinal case study of stackoverflow. In *NIPS 2015 Workshop on Social and Information Networks, 29th Annual International Conference on Neural Information and Processing Systems*.
- [Sirres et al., 2018] Sirres, R., Bissyandé, T. F., Kim, D., Lo, D., Klein, J., Kim, K., and Le Traon, Y. (2018). Augmenting and structuring user queries to support efficient free-form code search. *Empirical Software Engineering*, 23(5):2622–2654.
- [Song et al., 2017] Song, H., Ren, Z., Liang, S., Li, P., Ma, J., and de Rijke, M. (2017). Summarizing answers in non-factoid community question-answering. In *Proceedings*

- of the Tenth ACM International Conference on Web Search and Data Mining*, pages 405–414.
- [Squire and Funkhouser, 2014] Squire, M. and Funkhouser, C. (2014). " a bit of code": How the stack overflow community creates quality postings. In *2014 47th Hawaii International Conference on System Sciences (HICSS)*, pages 1425–1434. IEEE.
- [Srba and Bielikova, 2016] Srba, I. and Bielikova, M. (2016). Why is stack overflow failing? preserving sustainability in community question answering. *IEEE Software*, 33(4):80–89.
- [Sun et al., 2019] Sun, Z., Zhu, Q., Mou, L., Xiong, Y., Li, G., and Zhang, L. (2019). A grammar-based structural cnn decoder for code generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7055–7062.
- [Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.
- [Suzuki and Nagata, 2016] Suzuki, J. and Nagata, M. (2016). Rnn-based encoder-decoder approach with word frequency estimation. *arXiv preprint arXiv:1701.00138*.
- [Svyatkovskiy et al., 2020] Svyatkovskiy, A., Deng, S. K., Fu, S., and Sundaresan, N. (2020). Intellicode compose: Code generation using transformer. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1433–1443.
- [Thaller et al., 2019] Thaller, H., Linsbauer, L., and Egyed, A. (2019). Feature maps: A comprehensible software representation for design pattern detection. In *2019 IEEE 26th international conference on software analysis, evolution and reengineering (SANER)*, pages 207–217. IEEE.

- [Tian et al., 2013a] Tian, Q., Zhang, P., and Li, B. (2013a). Towards predicting the best answers in community-based question-answering services. In *Seventh International AAAI Conference on Weblogs and Social Media*.
- [Tian et al., 2013b] Tian, Y., Kochhar, P. S., Lim, E.-P., Zhu, F., and Lo, D. (2013b). Predicting best answerers for new questions: An approach leveraging topic modeling and collaborative voting. In *International Conference on Social Informatics*, pages 55–68. Springer.
- [Trienes and Balog, 2019] Trienes, J. and Balog, K. (2019). Identifying unclear questions in community question answering websites. In *ECIR*.
- [Tu et al., 2016] Tu, Z., Lu, Z., Liu, Y., Liu, X., and Li, H. (2016). Modeling coverage for neural machine translation. *arXiv preprint arXiv:1601.04811*.
- [Vasic et al., 2019] Vasic, M., Kanade, A., Maniatis, P., Bieber, D., and Singh, R. (2019). Neural program repair by jointly learning to localize and repair. *arXiv preprint arXiv:1904.01720*.
- [Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.
- [Wan et al., 2018] Wan, Y., Zhao, Z., Yang, M., Xu, G., Ying, H., Wu, J., and Yu, P. S. (2018). Improving automatic source code summarization via deep reinforcement learning. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 397–407. ACM.
- [Wang et al., 2009] Wang, K., Ming, Z., and Chua, T.-S. (2009). A syntactic tree matching approach to finding similar questions in community-based qa services. In *Proceedings*

of the 32nd international ACM SIGIR conference on Research and development in information retrieval, pages 187–194.

[Wang et al., 2020a] Wang, L., Zhang, L., and Jiang, J. (2020a). Duplicate question detection with deep learning in stack overflow. *IEEE Access*, 8:25964–25975.

[Wang et al., 2018] Wang, S., Lo, D., Vasilescu, B., and Serebrenik, A. (2018). Entagrec++: An enhanced tag recommendation system for software information sites. *Empirical Software Engineering*, 23(2):800–832.

[Wang et al., 2020b] Wang, W., Li, G., Ma, B., Xia, X., and Jin, Z. (2020b). Detecting code clones with graph neural network and flow-augmented abstract syntax tree. *arXiv preprint arXiv:2002.08653*.

[Wang et al., 2015] Wang, X.-Y., Xia, X., and Lo, D. (2015). Tagcombine: Recommending tags to contents in software information sites. *Journal of Computer Science and Technology*, 30(5):1017–1035.

[Wang et al., 2020c] Wang, Y., Wang, K., Gao, F., and Wang, L. (2020c). Learning semantic program embeddings with graph interval neural network. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):1–27.

[Wen et al., 2018] Wen, M., Wu, R., and Cheung, S.-C. (2018). How well do change sequences predict defects? sequence learning from software changes. *IEEE Transactions on Software Engineering*, 46(11):1155–1175.

[White et al., 2019] White, M., Tufano, M., Martinez, M., Monperrus, M., and Poshyanyk, D. (2019). Sorting and transforming program repair ingredients via deep learning code similarities. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 479–490. IEEE.

- [White et al., 2016] White, M., Tufano, M., Vendome, C., and Poshyvanyk, D. (2016). Deep learning code fragments for code clone detection. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 87–98. IEEE.
- [Wilcoxon, 1992] Wilcoxon, F. (1992). Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer.
- [Wong et al., 2013] Wong, E., Yang, J., and Tan, L. (2013). Autocomment: Mining question and answer sites for automatic comment generation. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 562–567. IEEE.
- [Xia et al., 2017] Xia, X., Bao, L., Lo, D., Kochhar, P. S., Hassan, A. E., and Xing, Z. (2017). What do developers search for on the web? *Empirical Software Engineering*, 22(6):3149–3185.
- [Xia et al., 2016] Xia, X., Lo, D., Correa, D., Sureka, A., and Shihab, E. (2016). It takes two to tango: Deleted stack overflow question prediction with text and meta features. In *2016 IEEE 40th annual computer software and applications conference (COMPSAC)*, volume 1, pages 73–82. IEEE.
- [Xia et al., 2013] Xia, X., Lo, D., Wang, X., and Zhou, B. (2013). Tag recommendation in software information sites. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 287–296. IEEE.
- [Xu et al., 2017] Xu, B., Xing, Z., Xia, X., and Lo, D. (2017). Answerbot: Automated generation of answer summary to developers’ technical questions. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 706–716. IEEE.

- [Xu et al., 2018] Xu, B., Xing, Z., Xia, X., Lo, D., and Li, S. (2018). Domain-specific cross-language relevant question retrieval. *Empirical Software Engineering*, 23(2):1084–1122.
- [Xu and Li, 2007] Xu, J. and Li, H. (2007). Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 391–398. ACM.
- [Xu et al., 2019] Xu, Z., Li, S., Xu, J., Liu, J., Luo, X., Zhang, Y., Zhang, T., Keung, J., and Tang, Y. (2019). Ldfr: Learning deep feature representation for software defect prediction. *Journal of Systems and Software*, 158:110402.
- [Yang et al., 2014a] Yang, J., Hauff, C., Bozzon, A., and Houben, G.-J. (2014a). Asking the right question in collaborative q&a systems. In *Proceedings of the 25th ACM conference on Hypertext and social media*, pages 179–189. ACM.
- [Yang et al., 2014b] Yang, J., Tao, K., Bozzon, A., and Houben, G.-J. (2014b). Sparrows and owls: Characterisation of expert behaviour in stackoverflow. In *International conference on user modeling, adaptation, and personalization*, pages 266–277. Springer.
- [Yang et al., 2013] Yang, L., Qiu, M., Gottipati, S., Zhu, F., Jiang, J., Sun, H., and Chen, Z. (2013). Cqarank: jointly model topics and expertise in community question answering. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 99–108.
- [Yang et al., 2016] Yang, X., Lo, D., Xia, X., Bao, L., and Sun, J. (2016). Combining word embedding with information retrieval to recommend similar bug reports. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*, pages 127–137. IEEE.

- [Yang et al., 2015] Yang, X., Lo, D., Xia, X., Zhang, Y., and Sun, J. (2015). Deep learning for just-in-time defect prediction. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 17–26. IEEE.
- [Yao et al., 2013] Yao, Y., Tong, H., Xie, T., Akoglu, L., Xu, F., and Lu, J. (2013). Want a good answer? ask a good question first! *arXiv preprint arXiv:1311.6876*.
- [Yao et al., 2015] Yao, Y., Tong, H., Xie, T., Akoglu, L., Xu, F., and Lu, J. (2015). Detecting high-quality posts in community question answering sites. *Information Sciences*, 302:70–82.
- [Ye et al., 2014] Ye, T., Xie, B., Zou, Y., and Chen, X. (2014). Interrogative-guided re-ranking for question-oriented software text retrieval. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 115–120.
- [Ye et al., 2016] Ye, X., Shen, H., Ma, X., Bunescu, R., and Liu, C. (2016). From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th international conference on software engineering*, pages 404–415.
- [Yin and Neubig, 2017] Yin, P. and Neubig, G. (2017). A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*.
- [Zampetti et al., 2020] Zampetti, F., Serebrenik, A., and Di Penta, M. (2020). Automatically learning patterns for self-admitted technical debt removal. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 355–366. IEEE.

- [Zhang et al., 2007] Zhang, J., Ackerman, M. S., and Adamic, L. (2007). Expertise networks in online communities: structure and algorithms. In *Proceedings of the 16th international conference on World Wide Web*, pages 221–230. ACM.
- [Zhang et al., 2019] Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K., and Liu, X. (2019). A novel neural source code representation based on abstract syntax tree. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 783–794. IEEE.
- [Zhang et al., 2018] Zhang, T., Upadhyaya, G., Reinhardt, A., Rajan, H., and Kim, M. (2018). Are code examples on an online q&a forum reliable?: a study of api misuse on stack overflow. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 886–896. IEEE.
- [Zhang et al., 2017] Zhang, W. E., Sheng, Q. Z., Lau, J. H., and Abebe, E. (2017). Detecting duplicate posts in programming qa communities via latent semantics and association rules. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1221–1229.
- [Zhang et al., 2015] Zhang, Y., Lo, D., Xia, X., and Sun, J.-L. (2015). Multi-factor duplicate question detection in stack overflow. *Journal of Computer Science and Technology*, 30(5):981–997.
- [Zheng and Li, 2017] Zheng, W. and Li, M. (2017). The best answer prediction by exploiting heterogeneous data on software development q&a forum. *Neurocomputing*, 269:212–219.
- [Zhou et al., 2019] Zhou, P., Liu, J., Liu, X., Yang, Z., and Grundy, J. C. (2019). Is deep learning better than traditional approaches in tag recommendation for software information sites? *Information and Software Technology*, 109:1–13.

- [Zhou et al., 2016] Zhou, P., Qi, Z., Zheng, S., Xu, J., Bao, H., and Xu, B. (2016). Text classification improved by integrating bidirectional lstm with two-dimensional max pooling. *arXiv preprint arXiv:1611.06639*.
- [Zhu et al., 2015] Zhu, J., Shen, B., Cai, X., and Wang, H. (2015). Building a large-scale software programming taxonomy from stackoverflow. In *SEKE*, pages 391–396.
- [Zou et al., 2015] Zou, Y., Ye, T., Lu, Y., Mylopoulos, J., and Zhang, L. (2015). Learning to rank for question-oriented software text retrieval (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1–11. IEEE.