Research Collection School Of Computing and Information Systems

School of Computing and Information Systems

1-2022

On the reproducibility and replicability of deep learning in software engineering

Chao LIU

Cuiyun GAO

Xin XIA

David LO
Singapore Management University, davidlo@smu.edu.sg

John C. GRUNDY

See next page for additional authors

Author

Chao LIU, Cuiyun GAO, Xin XIA, David LO, John C. GRUNDY, and Xiaohu YANG

On the Replicability and Reproducibility of Deep Learning in Software Engineering

CHAO LIU, College of Computer Science and Technology, Zhejiang University, China, and PengCheng Laboratory, China

CUIYUN GAO, Harbin Institute of Technology (Shenzhen), China

XIN XIA*, Monash University, Australia

DAVID LO, Singapore Management University, Singapore

JOHN GRUNDY, Monash University, Australia

XIAOHU YANG, College of Computer Science and Technology, Zhejiang University, China

Deep learning (DL) techniques have gained significant popularity among software engineering (SE) researchers in recent years. This is because they can often solve many SE challenges without enormous manual feature engineering effort and complex domain knowledge. Although many DL studies have reported substantial advantages over other state-of-the-art models on effectiveness, they often ignore two factors: *(1) replicability* – whether the reported experimental result can be approximately reproduced in high probability with the same DL model and the same data; and *(2) reproducibility* – whether one reported experimental findings can be reproduced by new experiments with the same experimental protocol and DL model, but different sampled real-world data. Unlike traditional machine learning (ML) models, DL studies commonly overlook these two factors and declare them as minor threats or leave them for future work. This is mainly due to high model complexity with many manually set parameters and the time-consuming optimization process. In this study, we conducted a literature review on 93 DL studies recently published in twenty SE journals or conferences. Our statistics show the urgency of investigating these two factors in SE, where only 10.8% of the studies discussed any research questions affecting replicability and/or reproducibility. More than 74.2% of the studies do not even share source code and data to support the replicability of their complex models. Moreover, we re-ran four representative DL models in SE. Experimental results show the importance of replicability and reproducibility, where the reported performance of a DL model could not be replicated for an unstable optimization process. Reproducibility could be substantially compromised if the model training is not convergent, or if performance is sensitive to the size of vocabulary and testing data. It is therefore urgent for the SE community to provide a long-lasting link to a replication package, enhance DL-based solution stability and convergence, and avoid performance sensitivity on different sampled data.

CCS Concepts: • **Software and its engineering → Software maintenance tools**.

Additional Key Words and Phrases: Deep Learning, Replicability, Reproducibility, Software Engineering

1   INTRODUCTION

Deep learning (DL) has become a key branch of Machine Learning (ML) [51, 76, 122], and now is a core component in systems for many aspects of modern society, such as autonomous cars [128], medical image diagnosis [83], financial market prediction [32], etc. The popularity of DL technologies mainly derive from their success in computer vision [33, 126], natural language processing [24, 104], machine translation [28, 133], etc. Generally, DL models aim to facilitate representation learning by leveraging big data and a specially designed neural network [76], such as the Recurrent Neural Network (RNN) [106, 107] and the Convolutional Neural Network (CNN) [65, 73].

In recent years, DL models have been increasingly used in the Software Engineering (SE) domain for building things such as code search engines [44, 124, 134], code summarization tools [75, 135], vulnerability identification tools [38, 52], and so forth. Most studies have shown that a DL algorithm can achieve higher *effectiveness* over the state-of-the-art approaches that employ alternative solutions [37, 87, 134, 135, 161]. The main advantage of these neural network based models over other machine learning models is easier multi-level representation learning from raw data (e.g., code or text) for a specific task (e.g., code classification), substantially mitigating the hard work involved in manual feature engineering [5, 76, 86, 161].

Apart from effectiveness, *replicability* and *reproducibility* are also widely accepted as important considerations in scientific research [12, 13, 63, 101, 114]. Commonly, replicability refers to whether a reported experimental result can be exactly reproduced with the same model and the same data. This is usually achieved by sharing a replication package including source code and data [13, 63, 97]. This is a key way to help researchers build confidence in the scientific merit of a reported result [12, 13, 79]. However, most DL models are likely fail when exact reproduction is attempted due to the strong randomness in model initialization and optimization, unlike many other machine learning (ML) models [79]. Thus, for DL studies, we need to consider replicability under a broader definition: whether a reported experimental result can be *approximately* reproduced to a high probability with the same model and the same data.

Nonetheless, high replicability is just a requirement for the repeated experiment and it does not guarantee that the reported result represents the reality under new experiments [4, 12, 63]. Therefore, reproducibility is required, which refers to whether one reported experimental finding can be achieved by another new experiment with the same experimental protocol, the same model, but different sampled real-world data [63, 101].

Despite the scientific merit of replicability and reproducibility, many DL-based studies in SE often ignore these two issues. It is uncertain the degree to which their reported experimental results can be replicated or reproduced with various manually set model parameters, such as the number of training iterations, the size of vocabulary that converts words into numbers, etc. Although some studies realized the importance of these factors, they regard them as mere threats and left solving them to future work [10, 44, 75, 130].

To understand the prevalence and importance of replicability and reproducibility issues for DL studies in SE, we conducted this study. We wanted to investigate the following seven key research questions (RQs) in two parts.

Part I: Literature Review on DL Replicability and Reproducibility. The first three RQs explore the prevalence of replicability and reproducibility issues via a detailed literature review.

- **RQ1. How are DL models used and evaluated in SE studies?** We performed a meta-analysis on 93 DL models published in the last five years in twenty SE journals or conferences. Our statistics show that 77.4%

of them are published in the last two years, indicating DL's rising popularity in SE. 76.3% of these DL studies address the challenges in code/text representation learning via RNN or CNN based models.

- **RQ2. How often do SE studies provide replication packages to support the replicability of DL models?** We observed that *only 25.8%* of the reviewed DL studies provide publicly accessible replication packages. We thus highly recommend that DL all studies must share source code and data to strengthen their replicability, considering the complexity of the DL technique.

- **RQ3. How often do SE studies investigate RQs affecting DL replicability/reproducibility?** We found that *only 10.8%* of the DL studies discuss at least one RQ related to DL replicability/reproducibility. Therefore, the replicability and reproducibility of most DL studies in SE left unknown and future DL studies in SE must pay more attention to these two factors.

Part II: Experiments on DL Replicability and Reproducibility. To analyze the importance of replicability and reproducibility, the following four RQs re-run four typical DL models and investigate four representative elements affecting replicability/reproducibility. These four DL models include an information retrieval model DeepCS [44], an RNN based generation model RRGen [34], a generation model based on reinforcement learning plus neural network (RLNN) [95], and a classification model ASTNN [154]. They were selected because they cover four different representative SE tasks that now often use ML, and they were published within the last two years in top SE venues with accessible replication packages.

- **RQ4. How does model stability affect replicability?** Model stability is usually influenced by randomly initialized network weights and randomly selected training data in model optimization. After running each DL model multiple times, our experimental results show that *the performance of two of the DL models is substantially overestimated by 12.2% and 12.5%*, in terms of the average compared to their reported results. Therefore, overestimation caused by instability will strongly influence model replicability, as the reported results would be reproduced with a low probability.

- **RQ5. How does model convergence affect reproducibility?** Model convergence is determined by many aspects, such as quality of training data, optimization stop conditions, etc. When training each DL model with more iterations, we observe that *two of the DL models show strongly turbulent performance* when compared with the ones in RQ4. Here, performance may be improved by over 8% or reduced by less than -8%. Thus, the high level of performance turbulence derived from low convergence would strongly affect its reproducibility, due to the low probability of reproducing a good reported result.

- **RQ6. How does the out-of-vocabulary issue affect reproducibility?** The out-of-vocabulary (OOV) issue widely occurs when applying a DL model for natural language processing. This is because the vocabulary built for model training cannot cover new words in testing data. To test how the OOV issue affects reproducibility, we train a model with different vocabulary sizes. We find that *the performance of one DL model is highly sensitive to the vocabulary size*, and whose performance increased by 3.4% to 104.2% for larger sizes. The results imply a low reproducibility of this model as its reported performance can not be reproduced under the new testing data with many out-of-vocabulary words. We also observed that the pointer generator [123] in RLNN [95] can largely avoid the OOV issue by copying related existing word from the training data. The abstract syntax tree (AST) that captures the code structural information in ASTNN [154] can substantially mitigate the influence of OOV.

- **RQ7. How does testing data size affect reproducibility?** DL models are commonly verified on a subset of real-world data, e.g., GitHub[1] and Stack Overflow[2]. It is often implicitly assumed that experimental findings could be reproduced for larger-scale testing data. To investigate the relationship between testing

---

[1]https://github.com/
[2]https://stackoverflow.com/

data size and DL reproducibility, we test each pre-trained DL model under different sizes of testing data. We notice that *two of the DL models show considerable sensitivity to the testing data size*, where performance may be improved by 7% or decreased by -9%. In this case, results for a model whose performance is highly sensitive to testing data size can not be satisfactorily reproduced.

In summary, ignoring the importance of DL replicability and reproducibility can substantially threaten the validity of a DL model used in SE. Hence, in order to strengthen DL replicability/reproducibility, it is essential and urgent for the SE community to consider the following experimental guidelines:

- *To ensure DL replicability*, it is recommended for all SE DL-based studies to provide a long-lasting link to a replication package with an automated evaluation approach to enhance the DL stability and convergence.
- *To strengthen DL reproducibility*, SE studies using DL should mitigate issues brought by newly sampled data from other reproduced experiments, such as the OOV issue and sensitivity to the testing data scale.
- *To promote DL replicability/reproducibility studies*, SE studies using DL should reduce model training time so that the burden of computation resources will not be an obstacle for replicability/reproducibility studies

The main contributions of this study are:

- Conducting a detailed literature review on many recent DL studies in SE pubished in top SE venues to analyze the prevalence of DL replicability and reproducibility issues.
- Performing large-scale in-depth experiments on four representative SE studies using DL-based models to investigate the importance of DL replicability and reproducibility.
- Providing some guidelines for the SE community to mitigate the replicability and reproducibility issues in many DL-based SE studies.

The remainder of this paper is organized as follows. Section 2 provides a background and surveys key related work, followed by the detailed investigated RQs in Section 3. Section 5 and Section 6 present four studied DL models and their experimental setup. Section 7 and Section 9 show the results of our replication studies and discuss implications of these. Section 10 summarizes this study.

2 BACKGROUND AND RELATED WORKS

2.1 DL technology in SE

The popularity of DL models in SE is mainly due to the advantages of representation learning from raw data [79, 95, 134]. For example, in many recent SE studies, a large number of challenges derive from the semantic comprehension of code in programming languages [84, 85, 148, 149, 154], text in natural languages [34, 150], or their mutual transformation [44]. As code and text involves some form of natural language processing (NLP), it commonly starts with encoding words by a fixed size of vocabulary [44]. Afterwards, a DL model is used for embedding digitalized code/text into a vector space. However, the vocabulary built from training data usually suffers from the OOV issue on dynamic and growing size of testing data [8, 52, 54, 95, 146].

The DL models used in SE are commonly based on three different types [49, 154, 156]. The first type is the traditional artificial neural network (ANN) that consists of fully connected neural networks, such as the multilayer perceptron (MLP) [22, 31, 39], deep belief network (DBN) [39], etc. To enhance representation learning, researchers developed two popular DL types. One is the convolutional neural network (CNN) [65, 73] that can sense regional characteristics of a matrix data via special convolution functions. The other is the recurrent neural network (RNN) [106, 107] that can capture the features of sequential data. The long short term memory (LSTM) structure is a popular variation of RNN, which can not only capture the long-term pattern of sequential data as RNN but also focus on the short-term features [56, 121].

By leveraging a DL model to learn the representation from the vectorized code/text, the model can be optimized for a specific SE task [10, 48]. To train a DL model, the parameters within the model are usually initialized by a

pseudo-random generator and optimized by randomly selected subsets of training data. The optimization stops when values of a loss function converge or after a fixed number of training iterations. Many DL studies show their effectiveness over traditional state-of-the-art models [161].

2.2 Replicability and Reproducibility in SE

Replicability and reproducibility are considered to be key aspects of the scientific method [61, 61, 97]. This is because they help researchers confirm the merit of previous experimental findings and promote scientific innovations [97]. In the SE domain, replicability and reproducibility have been widely claimed as essential for all empirical studies due to the unsatisfactory quality of previous research findings [3, 16, 26, 40, 41, 64, 99]. However, they are rarely investigated [42, 68, 125, 153].

Based on the findings of empirical studies, a large proportion of recent SE studies leveraged ML techniques to solve SE tasks [2, 102]. When verifying the validity of newly proposed ML models, the replication of previous state-of-the-art baselines are required. However, the replicability and reproducibility of these ML models are barely investigated in SE. Even for the most active SE research, defect prediction, a survey shows that very few ML models are replicated and model reliability is unclear [101]. To improve the replicability of ML models, it is suggested using replication infrastructures (e.g., OpenML[3]) to mitigate the replication efforts [101], leveraging the Docker container to save the experimental environment [23], and providing standard replication guidelines [101].

In recent years, many DL models have been successfully applied for diverse SE tasks [87, 134]. Although DL models often show substantial outperformance over traditional ML models, we noticed that they often overlooked replicability and reproducibility, as done for many SE studies using traditional ML models. However, DL models have many differences with the traditional ML models. These include high complexity with many manually set parameters and the time-consuming optimization processes [43, 76]. Therefore, the requirement for DL replicability and reproducibility may have many differences. Due to the lack of study of DL replicability and reproducibility, we conducted this study.

3 RESEARCH QUESTIONS

We present our study's seven key RQs on the replicability and reproducibility of DL models in SE, divided into two parts – literature review and experiments. We summarise how we went about answering each RQ in this study.

Part I: Literature Review on DL Replicability and Reproducibility. The first part studies the prevalence of replicability and reproducibility issues via a literature review.

> **RQ1.** How are DL models used and evaluated in SE studies?

Before investigating the replicability/reproducibility issues, we need to analyze how SE studies using DL models are reported in the SE literature and what are the characteristics of these studies. To do this we performed a literature review on studies published in twenty top SE journals/conferences in the last five years.

> **RQ2.** How often do SE studies provide replication packages to support the replicability of their DL models?

---

[3]https://www.openml.org/

To help others replicate work, replication packages are often released. We count the percentage of the reviewed studies in RQ1 that share accessible replication packages. A small percentage value implies a high prevalence of replicability issues in DL studies.

RQ3. How often do SE studies investigate RQs affecting DL replicability/reproducibility?

To further analyze the prevalence of replicability/reproducibility issues, we inspect the percentage of reviewed studies in RQ1 that discuss any RQ related to DL replicability or reproducibility. A lower ratio indicates low attention on DL replicability and reproducibility from the SE community.

Part II: Experiments on DL Replicability and Reproducibility.

The second part of our study evaluates the importance of replicability/reproducibility in SE studies using DL models by a suite of experiments on four representative DL models. These are DeepCS [44], RRGen [34], RLNN [95], and ASTNN [154]. These four models were selected because they cover four different SE tasks and are published in the last two years in top SE venues with accessible replication packages.

RQ4. How does DL model stability affect replicability?

DL model stability is affected by the random initialization and the iterative optimization with randomly selected batches of training data. However, DL studies in SE often report one experimental result. It is unknown whether the reported result can be replicated with high probability. Therefore, we run the above four DL models multiple times and estimate their replicability.

RQ5. How does DL model convergence affect reproducibility?

Ideally, the random optimization of DL models stops when the iterative process converges or all the training data have been used. However, DL models in SE usually stop optimization after a limited number of iterations, leading to an uncertainty over their convergence level. To investigate the relationship between convergence and reproducibility, we analyze how DL model performance changes by training with more iterations. A low convergent model would generate highly turbulent performance, as the model is trained for a higher number of iterations. High turbulence indicates low reproducibility, because a good reported performance would be more difficult to be replicated.

RQ6. How does the Out-of-Vocabulary Issue Affect Reproducibility?

Although DL draws high popularity in code/text representation learning in SE, the OOV issue is rarely investigated. If a model suffers from the OOV issue, its performance would be sensitive to new words in testing data. Then the reported result would be hardly reproduced for different testing data, i.e., low reproducibility. To analyze whether the OOV issue is a high potential threat to reproducibility, we train the four DL models with different sizes of vocabulary and estimate the sensitivity degree of model performance.

RQ7. How does testing data size affect reproducibility?

The DL models in SE are commonly verified by testing data collected from a real-world environment. However, this data collection process is usually random, and the testing data size only covers a small fraction of the real

Table 1. The selected SE conferences and journals for literature review.

No.	Venue	Full Name
1	FSE	ACM SIGSOFT Symposium on the Foundation of Software Engineering/ European Software Engineering Conference
2	ICSE	International Conference on Software Engineering
3	ASE	International Conference on Automated Software Engineering
4	ISSTA	International Symposium on Software Testing and Analysis
5	ESEM	International Symposium on Empirical Software Engineering and Measurement
6	SANER	International Conference on Software Analysis, Evolution, and Reengineering
7	ICSME	International Conference on Software Maintenance and Evolution
8	ICPC	IEEE International Conference on Program Comprehension
9	ICST	IEEE International Conference on Software Testing, Verification and Validation
10	ISSRE	International Symposium on Software Reliability Engineering
11	TOSEM	ACM Transactions on Software Engineering and Methodology
12	TSE	IEEE Transactions on Software Engineering
13	JSS	Journal of Systems and Software
14	IST	Information and Software Systems
15	ASEJ	Automated Software Engineering
16	ESE	Empirical Software Engineering
17	IETS	IET Software
18	STVR	Software Testing, Verification and Reliability
19	JSEP	Journal of Software: Evolution and Process
20	SQJ	Software Quality Journal

world data that exists. It is also implicitly assumed that the reported good performance achieved from the testing data can be generalised and reproduced with larger test data. To investigate the relationship between testing data size and reproducibility, we analyze how a pre-trained DL model performs on different sizes of testing data. If the model performance varies largely at different scales of testing data, its reproducibility will be compromised.

4 LITERATURE REVIEW

We present a literature review on DL models in SE to answer the first three RQs raised in Section 3.

4.1 RQ1. How are DL models used and evaluated in SE studies?

Motivation. Recently, DL techniques are frequently applied in the SE domain, therefore it is important to investigate how the DL models are used and evaluated for SE tasks. In this study we also investigate which SE tasks DL models work for, and which DL techniques are applied in SE tasks specifically. By answering these questions, we can perceive the overview of DL studies in SE and further analyze their replicability/reproducibility issues.

Method. To find DL studies in SE, we conducted a literature review using twenty widely read SE journals and conferences listed in Table 1. We performed an automated search on three digital library portals, limiting to the past five years (2015-2019), including IEEE Xplore[4], ACM digital library[5], and Web of Science[6] with terms "deep OR neural OR network" for literature title, abstract, and keywords. Afterward, we inspected the searched 676 papers one by one and excluded the ones that do not propose a DL model or are not a full research paper. We finally obtained 93 papers after this filtering.

---

[4]https://ieeexplore.ieee.org
[5]https://dl.acm.org
[6]http://apps.webofknowledge.com

Table 2. Number of papers published in the recent five years.

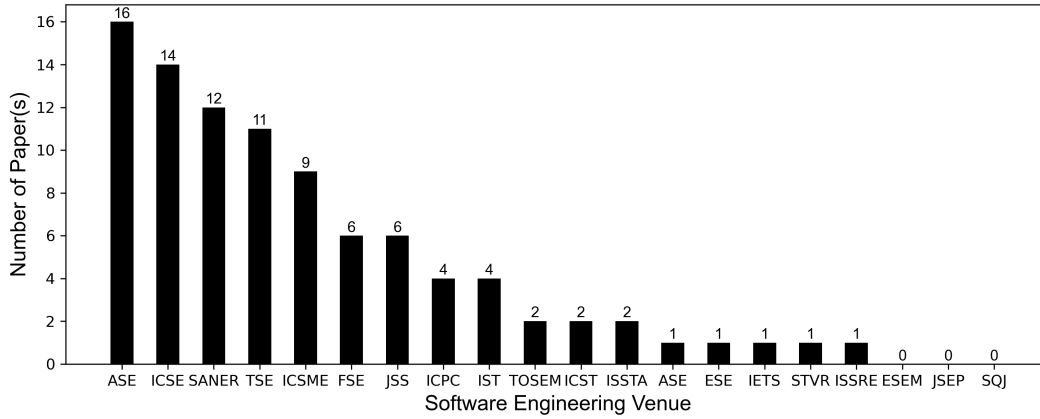| Year	2019	2018	2017	2016	2015	Total
#Papers in SE conferences	34	15	11	6	0	66
#Papers in SE journals	13	10	1	2	1	27
#Papers in total	47	25	12	8	1	93

Fig. 1. Number of papers published in twenty SE venues.

Results. Table 2 shows that **77.4% of SE studies using SL** are published in these top SE venues in the last two years. The number of papers published in 2019 and 2018 nearly doubled compared with the previous three years. This result shows the growing popularity of the DL technique in SE. Table 2 also shows that there are **far more DL studies published in SE conferences instead of journals** (66 vs. 27). This implies that researchers would like to present their new DL studies at top SE conferences. Fig. 1 illustrates the total number of DL studies published in each SE venue. From the figure, we can observe that the **top-7 SE venues that published these SE studies using DL** are ASE, ICSE, SANER, TSE, ICSME, FSE, and JSS. These SE venues contribute 79.6% of the total number of DL studies in the recent five years.

To understand the characteristics of these 93 reviewed studies, we classified them into 47 SE tasks and 10 study subjects as shown in Table 3. Note that as the study [154] applied the proposed DL model on two different SE tasks (code classification and code clone detection), therefore we have 94 total number of studies in this table instead of 93. From Table 3 we can observe that **63.8% of the DL studies focus on the study of code and defects**, followed by GitHub, non-code software artifact, and Stack Overflow. Furthermore, we notice that **45.7% of the proposed DL models work on eight SE tasks**, as illustrated in Fig. 3. Here code clone detection [36, 154] and defect prediction [98, 136] are the two most popular SE tasks for DL.

We observed that the reviewed DL models used are based on **five fundamental DL techniques**, including recurrent neural network (RNN) [66], convolutional neural network (CNN) [144], deep feed-forward network (DFFN) [60], deep belief network (DBN) [137], and reinforcement learning (RL) [95]. Note that a fundamental technique may contain many variations. For example, the long-short term memory (LSTM) belongs to the RNN model. Statistics in Table 4 show that **76.3% of DL studies utilized the RNN, CNN, or their combination** for modeling SE tasks. We notice that most of DL studies leverage new DL techniques to overcome the challenges in
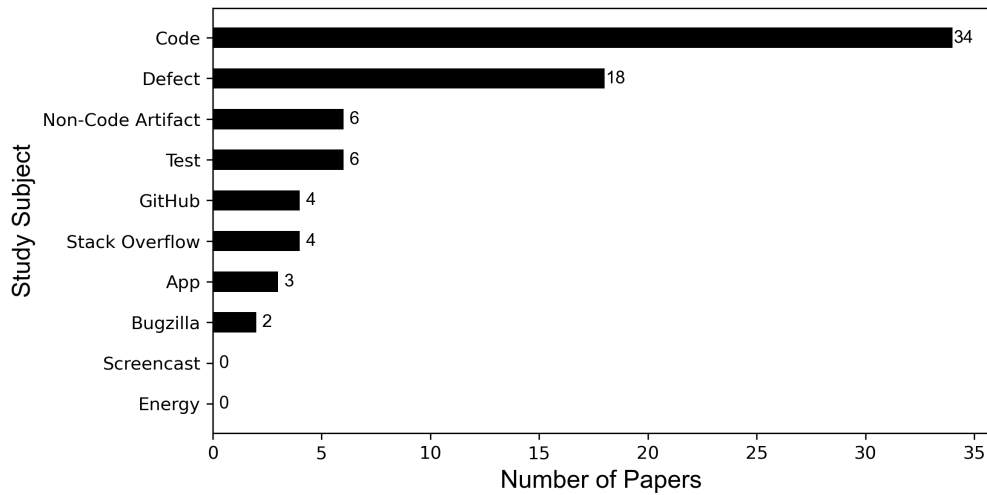
Fig. 2. Number of papers addressing the semantics of code or text for different SE studies.
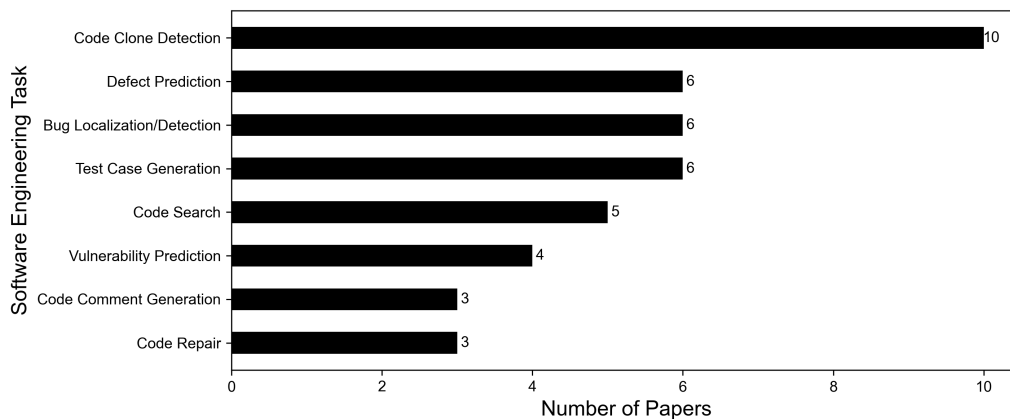
Fig. 3. The top-8 popular SE tasks in terms of the number of published papers.

understanding semantics of code and text and their transformation (e.g., code Summarization [75, 135], code change generation [130], etc.) in SE tasks, as illustrated in Fig. 2. Furthermore, Table 4 also shows that 5.4% of the DL studies have begun to leverage the RL technique to improve model performance.

Implications. In the last two years, using DL techniques has gained increasing popularity in the SE community. The DL models are widely applied to various SE tasks (e.g., code clone detection, defect prediction, etc.) due to their better representation learning of code/text semantics. RNN and CNN based models and their combinations are the most frequently used DL techniques. Meanwhile, the RL technique shows its potential to further enhance the performance of existing DL models (e.g., RNN and CNN).

Table 3. Categories of ninety three DL studies in our literature review. There are ninety four studies in the table because the study [154] performs predictions on two SE tasks (code clone detection and code classification).

Study Subject	Tasks and References	#Task	#Study
Code	Code clone detection [14, 36, 46, 78, 112, 142, 147, 152, 154, 157] Code search [15, 44, 45, 134, 138], Code classification [30, 74, 154] code readability classification/prediction [105, 117], Function type inferring [53, 103] Code generation [35, 115], Code Summarization [75, 135], Code Decompilation [66, 71] Code change generation [130], Data structure classification [108], Reverse execution [111] Design pattern detection [127], Technical debt detection [118], Story points prediction [21] Inconsistent method name refactoring [92], Stable patch detection [55]	16	36
Defect	Defect prediction [94, 98, 129, 136, 137, 140], Vulnerability prediction [27, 38, 52, 151] Bug localization/detection [59, 72, 81, 139, 144, 155] , Code repair [10, 132, 141] Code smell detection [89, 90], Anti-pattern identification [7] Code naturalness prediction [54], Static checker alarm classification [77]	8	24
GitHub	Issue-commit link recovery [120, 145], Effort prediction [11, 70], Duration prediction [96] Commit message generation [62], Pull request description generation [95]	5	7
Non-Code Artifact	Code comment generation [57, 160, 162], Software resources tracking [48] Software configuration prediction [49], Software incidents triage [20] requirement actor/action extraction [1]	5	7
Test	Test case generation [8, 17, 69, 93, 131, 159]	1	6
Stack Overflow	Link classification [146], Question retrieval [19], API extraction [100] Tag recommendation [91], Intention classification [58]	5	5
App	GUI generation [18, 109], Communication identification [158] Review response classification/generation [34, 47]	3	5
Bugzilla	Duplicate bug detection [29], Bug report summarization [80]	2	2
Screencast	Programming screencast action recognition [156]	1	1
Energy	Software energy consumption prediction [119]	1	1
Total	-	47	94

4.2 RQ2. How Often do SE Studies Provide Replication Packages to Support the Replicability for DL Models?

Motivation. Replicability is a fundamental requirement for DL models. DL replicability can however be compromised when other individual researchers re-implement a model by themselves with some careless errors or omissions. Sharing replication packages is a key to better support DL model replicability as it allows other researchers or practitioners to much more easily repeat results reported in a paper. Therefore, to analyze whether replicability is a supported issue for DL studies in SE, we investigate the percentage of the reviewed DL studies that share links to replication packages and whether the links are accessible.

Method. To investigate the prevalence of support for replicability issues in SE studies, we inspected the reviewed 93 DL studies in Section 4.1 one by one. For each study, we searched all the links within papers, and checked whether these links are related to replication packages. Finally, we counted the ratio of DL studies providing accessible replication packages. A lower ratio indicates lack of support for replication packages and hence replicability issues in SE studies using DL models.

Table 4. Statistics of basic model settings from ninety three DL studies in literature review. The basic DL models include recurrent neural network (RNN), convolutional neural network (CNN), deep belief network (DBN), deep feed-forward network (DFFN), and reinforcement learning (RL).

Configuration	Model Setting	#Study	%Study
The study with individual DL model	RNN	36	38.7%
	CNN	25	26.9%
	DFFN	20	21.5%
	DBN	2	02.2%
	RL	0	00.0%
The study with combined DL models	RNN + CNN	3	03.3%
	RNN + RL	2	02.2%
	CNN + RL	2	02.2%
	RNN + DFFN	1	01.1%
	RNN + CNN + DFFN	1	01.1%
	RNN + CNN + RL	1	01.1%

Table 5. Statistics of ninety three studies that share links to their replication packages.

Item	#Study	%Study
Link included	27	29.0%
Accessible link	24	25.8%

Results. Table 5 shows the statistics of the reviewed DL studies that share links to their replication packages. We notice that **only 29% of these SE studies using DL** provided a link to publicly shared source code and data. By entering the links with a browser, we found that three links are broken. Therefore, only 25.8% of the 93 DL studies provide *accessible* links to their replication packages. How long these remaining replication package links will remain available is unknown.

Implications. Most of our reviewed DL studies in SE do not provide replication packages that are publicly accessible (with the link given in the paper), leaving prevalent threats to low replicability. Under this circumstance, other researchers or practitioners need to replicate existing studies involving DL models from scratch and may miss important implementation details.

4.3 RQ3. How Often do SE Studies Investigate RQs Affecting DL Replicability/Reproducibility?

Motivation. According to our observation, some internal factors within DL models may affect study replicability and reproducibility, although sharing replication packages can help support DL replicability. For example, a model with unstable performance will lead to low replicability. Model stability can be influenced by randomly initialized network weights for the DL model. Although some replication packages may provide initialized weights, the random optimization process with many manually set parameters may still influence model stability. Moreover, if the model performance is sensitive to the size of the testing data, DL-based study reproducibility would be unsatisfactory. Therefore, discussing these kind of internal factors in DL models can further strengthen the DL replicability and reproducibility. We analyzed how often DL studies in SE investigate RQs affecting DL replicability/reproducibility and further classify the related RQs. A work that discusses no RQ on DL replicability/reproducibility would possess potential threats to its validity.

Method. To investigate how DL studies in SE discuss internal factors (e.g., model stability, the performance sensitivity on testing data size, etc.) on DL-based study replicability/reproducibility, we inspected what the RQs in each paper aim to investigate. We then calculated the percentage of reviewed studies in Section 4.1 that conduct an RQ investigating DL replicability or reproducibility. If DL studies in SE rarely perform experiments on such internal factors, the DL replicability/reproducibility issue would be still prevalent even if some DL studies may provide replication packages.

Results. As shown in Table 6, we observe that the reviewed DL studies focus on three types of RQ, including effectiveness – whether the proposed DL model outperforms the state-of-the-art baseline; efficiency – whether the DL model works faster than the baseline; and replicability/reproducibility – the degree to which that the reported model performance can be replicated/reproduced considering some model/experimental settings. Table 6 shows that **all the 93 studies** investigate the effectiveness of the model e.g., in terms of precision, recall, F1, etc. This is because verifying model effectiveness is a basic requirement for DL studies. However, **only 40.9% of the studies** provided runtime information that informs the efficiency of their proposed approach for training and prediction. As to the model replicability/reproducibility, **only 10.8% of the total studies** investigated RQs related to these factors.

To understand the characteristics of RQs on DL replicability/reproducibility, we classify them into four RQ types as described in Table 7. These RQ types include model randomness [11, 52, 129], whether the randomness in DL model (e.g., random optimization process) affects model replicability; model convergence [66, 81], whether the model optimization is divergent so that the turbulent performance leads to low replicability; out-of-vocabulary (OOV) issue [8, 52, 54, 95, 146], whether model performance is sensitive to the size of vocabulary for encoding words into vectors, resulting in low reproducibility; sensitivity to testing data size [112], whether the model performance is sensitive to the size of testing data (i.e., low reproducibility). Table 7 shows that the OOV issue is the most frequently discussed RQs because this issue is widely considered in the domain of the natural language processing. However, **the number of papers that discuss this issueis low** (5 papers, 5.4% of total studies), since 76.3% of the reviewed 93 DL studies need to encode code/text by using a fixed size of vocabulary (as illustrated in Fig. 2). The number of papers related to the other three RQ types is also very low.

Although some studies realized the importance of replicability/reproducibility, they just regarded them as some among many threats to validity and left them to future work [10, 44, 75, 130]. One major obstacle to addressing DL replicability/reproducibility issue is the time-consuming nature of DL training. This is because researchers' limited computational resources have to be used for verifying the model effectiveness at first.

Implications. Some internal factors, such as model randomness, convergence, OOV issue, and sensitivity to testing data size in DL models, may threaten DL replicability and reproducibility even if the authors provide accessible replication packages. However, these factors are widely overlooked in DL-based SE studies. Therefore, there is a need for more studies on the impact of replicability and reproducibility issues in DL applications in SE. To promote such studies, reducing the time it takes to train DL models is also important because of limited computational resources.

5 DEEP LEARNING MODELS

According to the literature review in Section 4, we know that the DL replicability/reproducibility issues are commonly overlooked and often regarded as minor threats in the SE domain. To analyze the importance of DL replicability/reproducibility, we designed four experiments, corresponding to the last four RQs in Section 3, using four DL models for SE tasks. This section provides brief descriptions on these four DL models, which are used for different SE tasks including code search [44], review response generation [34], pull request description generation [95], and code classification [154]. These DL models are selected as study subjects because they are published in

Table 6. Statistics of studies that discuss RQs related to model effectiveness, efficiencey, or replicability/reproducibility.

Type of RQ	Description	#Study	%Study
Effectiveness	Whether the proposed new DL model outperforms some baselines in terms of some evaluation metrics.	93	100.0%
Efficiency	Whether the proposed new DL model works faster than some baselines during model training and testing respectively.	38	40.9%
Replicability/Reproducibility	The degree to which the reported model performance can be replicated/reproduced considering some model settings or experimental settings.	10	10.8%

Table 7. Classification of RQs related to replicability/reproducibility issues among reviewed DL studies in SE. There are ten DL studies in total, and one study discusses two RQ types.

RQ Type	Description	#Study	%Study
Model randomness	Whether the randomness in model training (e.g., parameter initialization and random optimization process) affects DL replicability.	3	3.2%
Model convergence	Whether the model optimization stops due to convergence instead of the limited size of training data; If so, whether it strongly influences the DL replicability.	2	2.2%
Out-of-vocabulary issue	If a DL model involves a vocabulary for encoding words, whether the fixed size of vocabulary leads to the reported model performance is unreproducible.	5	5.4%
Sensitivity to testing data size	Whether the model performance is sensitive to the size of testing data, resulting in low reproducibility.	1	1.1%

Table 8. Summary of four DL models for different SE tasks.

Task	Description	Model Name
Code search	Searching related code from codebase according to a user's query.	DeepCS [44]
Review response generation	Generating high-quality response to a user's review of an App.	RRGen [34]
Pull request description generation	Generating high-quality description for an empty pull request.	RLNN [95]
Code classification	Classifying code fragments according to their functionality.	ASTNN [154]

top SE venues in the last two years with accessible replication packages. Table 8 summarizes these four models and their details are briefly described as follows.

5.1 Code Search by DeepCS

Task and Solution. Searching and reusing existing code can help developers accelerate software development. Code search research aims to provide developers a code search engine that returns some relevant code examples from a large scale codebase, e.g., GitHub, according to developers' search queries, such as "how to convert string to int". To solve this task, a DL model named DeepCS [44] first embeds a search query and all candidate code methods into a shared vector space, and then returns the top-10 methods relevant to the query in terms of the Cosine similarities.

Modeling and Optimization. To learn the relationship between queries in natural language and code methods in a programming language, DeepCS leverages three recurrent neural networks (RNNs) [106] and one multilayer perceptron (MLP) [22] to embeds code method (c) and query (q) into vectors respectively:

$$\begin{cases} v_c & = RNN(m) + RNN(a) + MLP(t), \\ v_q & = RNN(q), \end{cases} \tag{1}$$

where m, a, t are three components of a code method c including method name, API sequence, and token set in method body; v_c and v_q are the vectorized method and query, respectively.

Note that the token set t is processed by a MLP instead of a RNN because the token set only considers the token frequency instead of the token order. With these two vectors, the relevant code methods to a query can calculated by the cosine similarity as $cos(v_c, v_q) = (v_c^T v_q)/(\|v_c\|\|v_q\|)$. To optimize parameters (θ) in RNNs and MLP, DeepCS initialized them by a pseudo-random generator and trained by the loss function (L):

$$L(\theta) = \sum_{<C,Q+,Q->\in P} max(0, 0.05 - cos(v_c, v_q^+) + cos(v_c, v_q^-)), \tag{2}$$

where $v_c \in C$ is a code method randomly selected from training data (P); $v_q^+ \in Q^+$ is the code related comment to stand for the related query; and $v_q^- \in Q^-$ is an irrelevant comment randomly selected from other methods. The objective of this loss function is to shorten the similarity between the matched method (v_q^+) and query (v_q) while enlarging the similarity for the unmatched pairs.

5.2 App Review Response Generation by RRGen

Task and Solution. Studies show that replying to users' reviews of an App largely increases the chances of a user updating their given rating, so that the App can maintain its user base and even attract more [34]. To automatically generate high-quality responses to user reviews, an RNN-based model called RRGen (review response generation) was developed to learn the relationship between review-response pairs [34].

Modeling and Optimization. To generate a high-quality response (y), RRGen takes a review (x) as its input and builds an encoder-decoder model for x and y by leveraging RNN and MLP:

$$y = RNN(x_t, MLP(x_c, x_l, x_r, x_s)) \tag{3}$$

where x_t is a sequence of keywords in the review x; x_c, x_l, x_r, and x_s are four high-level attributes of a review, which are App category, review length, user rating, and user's positive/negative sentiment respectively.

To encode tokens in review keywords and responses, RRGen constructs a vocabulary with top-10k frequently occurred words in training data. Moreover, to optimize the randomly initialized parameters (θ) in RRGen, the model utilized a loss function (L):

$$L(\theta) = max_\theta \frac{1}{N} \sum_{i=1}^{N} log p_\theta(y_i'|y_i, x_{c_i}, x_{l_i}, x_{r_i}, x_{s_i}) \tag{4}$$

where p_θ calculates the cross-entropy between the generated response (y_i) and the ground-truth (y_i') for the i-th review; N is the total number of review-response pairs; $x_{c_i}, x_{l_i}, x_{r_i}, x_{s_i}$ are the category, review length, user rating, and sentiment for the i-th review. This loss function intends to maximize cross-entropy between all pairs of generated responses and their ground-truth.

5.3 Pull Request Description Generation by RLNN

Task and Solution. Developers contribute to a project through a pull request (PR) with a description. The description helps reviewers and other developers understand their contributions. However, more than 34% of real-world PR descriptions are empty [95]. To help developers generate high-quality PR descriptions automatically, a reinforcement learning (RL) based RNN model was proposed [95]. We call the model RLNN in this study. It takes a PR related context as model input, including commit messages, the added code comments, and then produces a summary as the PR description.

Modeling and Optimization. To generate the correct PR description (g), RLNN treats the commit message plus the related code comments as the model input (s). RLNN works in two phases. It first builds an RNN encoder-decoder model for the input-output pairs as $y = RNN(x)$ and it is pre-trained by a common loss function (L_{ml}):

$$L_{ml} = -\frac{1}{|y|} \sum_{j=1}^{|y|} log p(y_j|\hat{y}_0, \ldots, \hat{y}_{j-1}, w), \tag{5}$$

where w the RNN parameters; $\hat{y}_0, \ldots, \hat{y}_{j-1}$ are the 0-th to (j-1)-th tokens of the input; y_j is the j-th generated token. The loss function intends to estimate the negative log-likelihood of the generated description.

In the second phase, all of the words are encoded by a fixed vocabulary with top-50 words shown in training data in terms of frequency. To overcome the (OOV) issue, RLNN integrates a pointer generator to select a token from the vocabulary or to copy one from the model decoding step. To further enhance the quality of the generated description, RLNN optimizes RNN by a special loss function (L):

$$L = \gamma L_{rl} + (1 - \gamma)L_{ml}, \tag{6}$$

where L_{ml} is the loss function defined in Eq. (5); γ is a coefficient to balance two loss functions L_{ml} and L_{rl}; L_{rl} is an reinforcement loss function described as follows:

$$L_{rl} = -(r(y^s) - r(y))) \sum_{j=1}^{|y^s|} log p(y_j^s|y_0^s, \ldots, y_{j-1}^s, w), \tag{7}$$

where y^s is a sampled description by an Monte-Carlo method while y is the ground-truth description; $r(y^s)$ or $r(y)$ measures the generation accuracy between the generated description y and y^s or y in terms of the ROUGE-L F1-score; $y_0^s, \ldots, y_{j-1}^s$ are the 0-th to (j-1)-th tokens of the sampled description; y_j^s is the j-th generated token for the sampled input. The major advantage of L_{rl} is the incorporation of the automated evaluation ROUGE into the model optimization, training the model in a more natural and accurate way.

5.4 Code Classification by ASTNN

Task and Solution. Correctly classifying code fragments by their functionalities helps developers understand and maintain software projects. To improve this code classification accuracy, an AST-based neural network (ASTNN) was proposed [154], which represents a code by a sequence of ASTs and performs the classification by using a CNN based model.

Modeling and Optimization. To learn the representation of a code fragment (x'), ASTNN first parses the code (x) into a sequence of small statement trees. Each sequence is used to train a RNN based model that is optimized by the following loss function:

Table 9. Experiment setup for four DL models.

Model	Training Iterations	Vocabulary Size	Testing Data Size	Evaluation Metric
DeepCS	500 epochs	10,000	16,262,602 methods	MRR
RRGen	3 epochs	10,000	14,727 reviews	BLEU-4
RLNN	22,000 iterations	50,000	4,100 pull requests	F1-Score of ROUGE-L
ASTNN	15 epochs	8,182	10,401 code fragments	Accuracy

$$L_{RNN} = \|n_1 - RNN(n_1)\|_2^2 + \|n_2 - RNN(n_2)\|_2^2, \tag{8}$$

where n_1 and n_2 are two children nodes of a parent node in a parsed code (x) separately; $RNN(n_1)$ and $RNN(n_2)$ are the generated vectors by an RNN model corresponding to the nodes n_1 and n_2 respectively. To learn the representation of the AST, the loss function leverages the second normal form to assess the learning accuracy.

Furthermore, to apply the pre-trained ASTNN to the code classification, $\hat{x} = W_0 x' + b_0$ is used to predict the its ground-truth category ($\hat{x}$), which is optimized by the following loss function:

$$L(\theta, \hat{x}, y) = \sum \left(-log \frac{exp(\hat{x}_y)}{\sum_j exp(\hat{x}_j)} \right), \tag{9}$$

where W_0 is a weight matrix and b_0 is a bias term; $\hat{x}$ is the predicted category while y is the ground-truth. The loss function L intends to increase the accuracy between the generated category and the ground-truth via the cross-entropy measurement.

6 EXPERIMENTAL SETUP

We describe the experimental setup for the four DL models summarised in Section 5. Table 9 provides their important settings related to the last four RQs in Section 3. We run the public replication packages shared by authors on two servers with eight GPUs in total. The detailed experiment setup is presented as follows.

DeepCS. Following the original study [44], DeepCS is trained using around 18 million commented Java methods with 10k fixed size of vocabulary, and the optimization stops at 500 epochs. Afterward, the model is tested using about 16 million Java code methods with top-50 search queries extracted from Stack Overflow as developers' search requirements. For each query, DeepCS returns top-10 relevant code. The performance of code search is estimated by a widely used metric MRR (mean reciprocal rank). MRR is defined as $Q^{-1} \sum_{q=1}^{Q} FRank_q^{-1}$, where Q is the total number of queries; $FRank$ is the rank of the first correct search to a query. We re-ran DeepCS by using the authors' replication package[7] from GitHub.

RRGen. For App review response generation, RRGen encodes words of reviews and responses in training and testing data by vocabulary with top-10k words appeared in training data. Subsequently, RRGen is trained using 279,792 review-response pairs with three epochs and tested using 14,727 reviews [34]. Note that the three epochs are long enough for model optimization, as each epoch takes more than 40 hours due to the large size of training data [34]. To assess the generation accuracy, the textual similarity between the generated responses ($\hat{y}$) and the ground-truth (y) is measured by the BLEU (bilingual evaluation understudy) score [6]. Generally, the BLEU score analyzes the co-occurrences of n-grams between y and $\hat{y}$, where n is set to 4 because it is demonstrated to be more

---

[7]https://github.com/guxd/deep-code-search

correlated with human judgments than other settings [88]. We re-ran RRGen by using the replication package[8] shared by the authors in GitHub.

RLNN. To generate a high-quality pull request description, RLNN is optimized using training data with 32.8k PRs. In the encoding phase, RLNN uses a fixed vocabulary with 50k words frequently occurred in training data. Note that RLNN is based on a pre-trained RNN model with 12,000 iterations and then performed an RL optimization on it with 22,000 iterations. To better understand the replicability of RLNN, we pre-trained the RNN one time and mainly investigated RQs on the RL part. Moreover, the validity of RLNN is verified on testing data involving 4.1k PRs. Model performance is evaluated by using ROUGE (recall-oriented understudy for gisting evaluation) metrics, which highly correlate with human assessment of summarized text quality [82]. Specifically, the F1-score of ROUGE-L is adopted in the model evaluation and its detailed definition can be found in the RLNN study [95]. We re-ran their public replication package[9] shared on GitHub.

ASTNN. For code classification, the validity of ASTNN is verified by a public dataset built by Mou et al. [110][10]. The dataset contains 46,887 code fragments with training data encoded using a vocabulary containing 8,182 words. It involves 10,402 code fragments in the testing dataset. The model training stops at 15 epochs. The classification accuracy is computed as $Accuracy = M^{-1} \sum_{m=1}^{M} f(x_m, y_m)$, where f is an indicator function that returns 1 if the predicted category of m-th code fragment (x_m) equals to the ground-truth (y_m), otherwise it returns 0; M denotes the total number of code fragments in the testing data. We re-ran the authors' replication package[11] that are shared publicly.

Note that the units (epoch or iteration) and size (3 or 500) of the model iterations are kept the same as the original studies, but their actual running time could be substantially different.

7 EXPERIMENTAL RESULTS

We present our experimental results to help answer the last four RQs described in Section 3, investigating the importance of DL replicability and reproducibility. The RQs aim to analyze how the internal factors (i.e., model stability, convergence, OOV issue, and testing data size) in DLs affect DL-based SE study replicability and reproducibility, and whether this influence is strong enough to threaten the validity of the studies.

7.1 RQ4. How does Model Stability Affect Replicability?

Motivation. DL replicability can be largely supported by sharing a replication package. Ideally, re-running the source code and data provided can replicate the reported model performance. However, a publicly shared replication package cannot guarantee that the models trained by different researchers are the same and produce the same experimental result as the reported one, due to the randomness in model initialization and optimization [79]. Therefore, the DL model performance could be varied for different researchers, so that a model with unstable performance would be of low replicability. This is the reason why the DL replicability requires that the reported performance by a DL study can be approximately reproduced (although not identically) with high probability. Thus, it is important to investigate whether DL models are stable enough and whether the stability level strongly affects the DL replicability.

Method. To estimate the four DL models' stability, we re-run each DL model ten times and analyze the statistics of our experimental results (e.g., mean, standard deviation, etc.). To assess the model replicability, we utilize the mean value of ten experimental results as the performance that the DL model likely produces with high
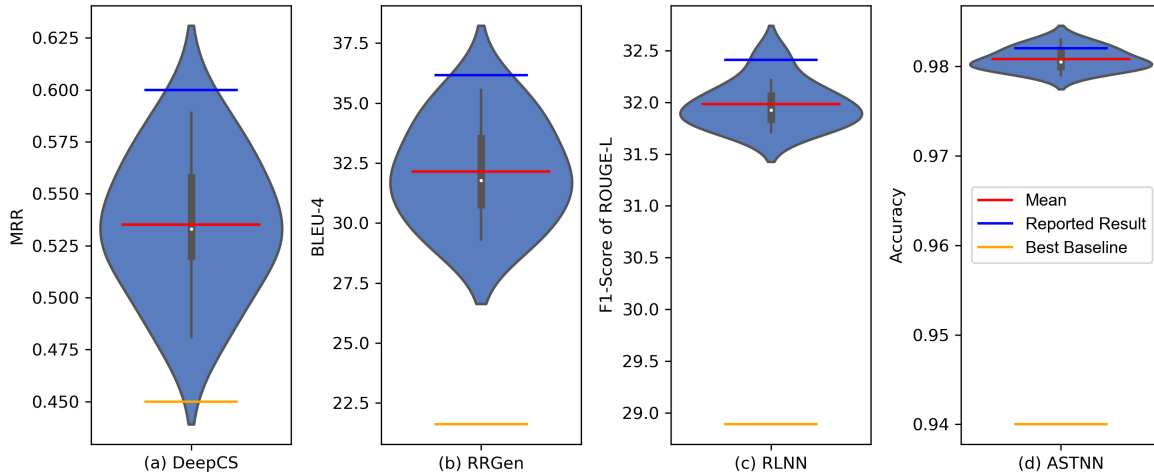

Fig. 4. Violin plots of four DL models with ten repeated experimental results, where white dot and red line indicate median and mean values respectively; blue and orange lines denote the performance of four models and the best baseline reported in the original studies, respectively.

probability. Hence, if the mean value is far from the result reported by the authors of the original studies, then the reported study results are difficult to reproduce i.e., it has low replicability.

Results. Fig. 4 illustrates the ten experimental results of four DL models (i.e., DeepCS, RRGen, RLNN, and ASTNn), where white dot and red line indicate the median and mean values, respectively; blue and orange lines show the performance of the DL model and the best baseline reported in the original study, respectively. The important statistics of Fig. 4 are listed in Table 10. From the table, we notice that the experimental performance of DeepCS obtains a mean 0.535 and standard deviation (std) 0.033 with the minimum 0.481 (min) and the maximum (max) 0.589. The mean performance of RRGen, RLNN, ASTNN are 32.144 (std = 0.032, min = 0.481, max = 0.589), 31.982 (std = 0.213, Min = 31.710, max = 32.457), 0.981 (std = 0.001, min = 0.979, max = 0.983) respectively.

To assess the relative stability between these four DL models, we calculated the coefficient of variation (c_v), equal to the standard deviation normalized by its mean. Table 10 shows that the c_v values of four models are 5.92%, 6.27%, 0.67%, and 0.1% respectively. Thus, **the DeepCS and RRGen DL models possess much lower stability** than the other two. We also observe that all of the reported results are larger or equal to the mean plus standard deviation of our multiple re-runs. This observation implies that **the reported performance of all four DL models can be achieved by other researchers with low probability**, i.e., low replicability. Therefore, a good reported DL performance without multiple runs could be achieved by chance.

Compared with the generated mean performance of the four DL models, the results reported in the original studies increased by 12.2%, 12.5%, 1.3%, and 0.1% respectively. This case suggests that **the performance of four models in the original studies are overestimated** to different degrees, where first two cases are more substantial (>12%) than the others (<1.4%). Thus, **a DL model with low replicability may strongly affect the overall validity of the study**. To analyze the correlation between the stability and replicability, we performed a Pearson correlation test [9] at a 5% significance level on the c_v values and the overestimation rates (12.2%, 12.5%, 1.3%, and 0.1%) between four models. This statistical test shows that DL stability and replicability are strongly correlated (cor = 1, p-value = 1). Our test result implies that **an unstable DL model leads to low study**

Table 10. Statistics of four DL models with ten repeated experimental results, where 'reported' and 'baseline' are the reported result of a model and the compared best baseline in author's study, respectively; Std is the standard deviation; c_v is the coefficient of variance equaling to std/mean.

Model	Min	Max	Mean	Std	Reported (vs. Mean)	Baseline	c_v
DeepCS	0.481	0.589	0.535	0.032	0.600 (+12.2%)	0.450	5.9%
RRGen	29.314	35.549	32.144	2.016	36.170 (+12.5%)	21.610	6.3%
RLNN	31.710	32.457	31.982	0.213	32.410 (+1.3%)	28.890	0.7%
ASTNN	0.979	0.983	0.981	0.001	0.982 (+0.1%)	0.940	0.1%

replicability. The premise is that the reported performance is far from the mean, especially larger than the mean plus standard deviation.

To further analyze the impacts of this overestimation, we compare the obtained mean values of multiple experiments with the performance of the best baselines reported in the DL-based SE studies. We find that the mean results of four models outperform the best baselines reported by four DL studies by 18.9%, 48.8%, 10.7%, and 4.4%, respectively. However, due to overestimation, the advantages of the four DL models over the best baselines are reduced by 43.3%, 27.6%, 12.2%, and 2.5% correspondingly. Therefore,**only reporting a good DL performance may threaten the experimental conclusions**.

Implications. DL models possess randomness, so that only reporting one good experimental result is not enough. If the reported performance is far from the mean values of multiple runs (especially larger than the mean plus standard deviation), **the reported result could be replicated with low probability**. In this situation, low DL replicability will produce greater negative effects on an unstable model. Therefore, it is important to estimate and improve the DL stability and replicability, instead of only reporting one good experimental result. Otherwise, the model validity could be largely threatened.

7.2 RQ5. How does Model Convergence Affect Reproducibility?

Motivation. DL models in SE are optimized with many manually set parameters (e.g., learning rate, the total number of training iterations, convergence coefficient, etc.), hypothesizing that the optimization is convergent or training model with more iterations would gain better performance. However, if this unverified assumption does not hold, the DL-based study reproducibility will not be high because the manually set parameters cannot adapt to data in different new experiments. Thus, the goal is to estimate the DL model convergence and investigate whether the convergence level is an influential factor for DL study reproducibility.

Method. To analyze the model convergence level, we investigate whether training DL models with more iterations can achieve better and more stable performance. We again use the four DL models, which have already trained ten times in Section 7.1, as our study subject. Thus, we have forty pre-trained models. For each DL model, we continue to optimize them with one-third more iterations/epochs and compare the change rate (CR) of performance before and after the new training iterations. A divergent model would produce highly turbulent performance, i.e., a large standard deviation of change rates (CR_{std}). Note that one-third more epochs or iterations are determined because this extended training is enough to analyze the turbulence level, and more iterations/epochs will substantially increase the total training time for forty DL models but our computation resource is limited. Besides, training DL models with too many iterations/epochs may not be always better [113]. The overfitting issue is one major reason [116].

Results. Fig. 5 shows our experimental results from the four DL models with default training iterations (black squares) vs. extended iterations (grey squares), whose statistics are listed in Table 10 and 11 respectively. From the Table 11, we can notice that, after the extended training, the mean performance of DeepCS, RRGen, RLNN, and ASTNN are 0.536 (min = 0.475, max = 0.572, and std = 0.033), 32.391 (min = 28.625, max = 35.399, std = 2.232), 31.697 (min = 31.395, max = 32.032, std = 0.181), and 0.981 (min = 0.978, max = 0.984, std = 0.002) respectively. Compared with our generated mean values, the model performance reported in the original study increased by 11.9%, 11.7%, 2.2%, and 0.1% respectively. These results indicate that **the reported results for these four DL models cannot be reproduced to different degrees** for experiments with extended model training. The reproducibility issue of the first two models, DeepCS and RRGen, show much higher negative effects over the other two.

For each DL model, we performed a Wilcoxon signed-rank test [143] at a 5% significance level between the experimental results before and after extended training. The tests show that all the statistical differences are not significant (p-value > 0.05), as shown in Table 11. This statistical analysis result implies that training all four DL models does not produce better performance. Also, we can notice that, comparing with the values in Table 10, Table 11 shows little difference, where the reported performance of four models cannot be better reproduced with the extended training (the reported performance outperforms the mean value by 11.9%, 11.7%, 2.2%, 0.2%); the relative model stability is not changed substantially (c_v values of four models are 6.2%, 6.9%, 0.6%, 0.2% respectively). Thus, **the extended training did not improve the model performance** due to the unchanged degree of model stability. The unchanged stability indicates that model training tends to be not convergent for a model with lower stability.
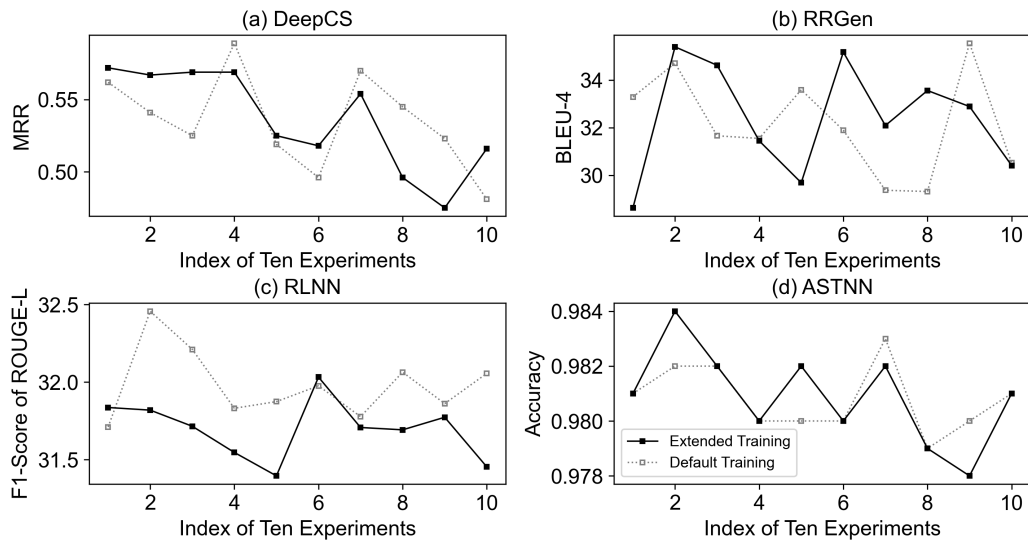

Fig. 5. Plots of four DL models with default training (grey squares) vs. extended training (black squares).

To estimate the convergence levels of these four DL models and analyze their effects, we calculated the change rates (CR) of model performance before and after training with more iterations/epochs. Fig. 6 illustrates violin plots of the change rates for each DL model, where the red horizontal line indicates the mean value of the change rate. The statistics of change rates are listed in Table 12, including the minimum (min) and maximum (max) values, mean, and standard deviation (std). We can observe that the convergence levels of four models are 6%,

Table 11. Statistics of four DL models trained with extended iterations/epochs for ten repeated experimental results, where 'reported' is the reported result of a model in the author's study, respectively; Std is the standard deviation; c_v is the coefficient of variance equaling to std/mean; p-value is the result of the Wilcoxon signed-rank test, at a 5% significance level, on the model performance before and after the extended training.

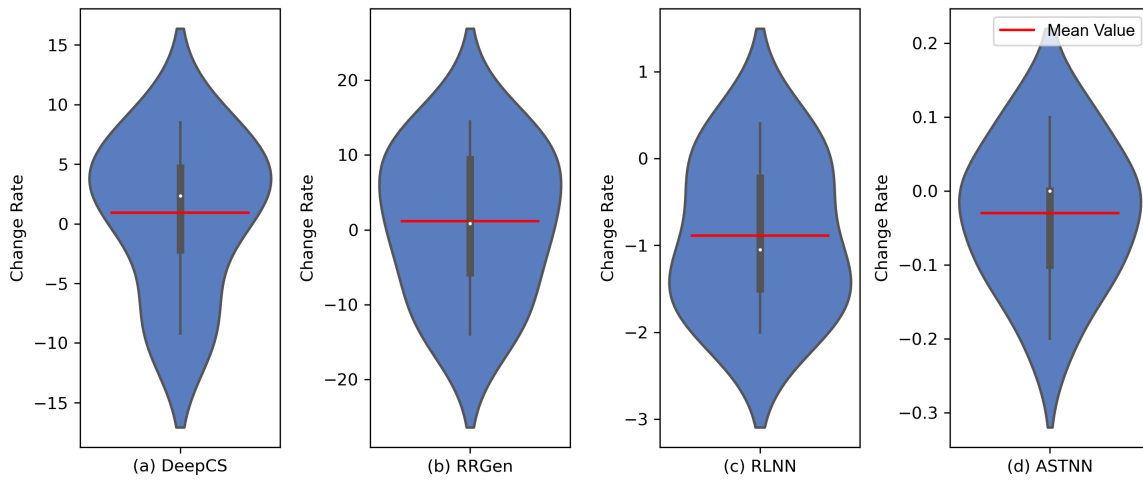| Model	Min	Max	Mean	Std	Reported (vs. Mean)	c_v	p-value
DeepCS	0.475	0.572	0.536	0.033	0.600 (+11.9%)	6.2%	>0.05
RRGen	28.625	35.399	32.391	2.232	36.170 (+11.7%)	6.9%	>0.05
RLNN	31.395	32.032	31.697	0.181	32.410 (+2.2%)	0.6%	>0.05
ASTNN	0.978	0.984	0.981	0.002	0.982 (+0.1%)	0.2%	>0.05

Fig. 6. Violin plots of the performance change rates after training models with extended iterations/epochs, where the red horzontal line indicates the average change rate.

9.3%, 0.8%, and 0.1% respectively in terms of the CR_{std} values, where the convergence level of the first two models is lower than the others. By comparing the min and max values, we can notice that a model with a lower convergence level may improve its performance by 14.5% or even reduce the performance by 9.1% after training the model with more iterations/epochs. In this case, **the variation caused by poor convergence levels may lead to different experimental conclusions**. Therefore, DL model convergence is an important factor for the DL study reproducibility issue.

Implications. We cannot assume that a DL model is convergent or the model performance can be improved with further training unless we provide any verification. This is because a model with a low convergence level may produce highly turbulent and unstable performance under different experimental data. In other words, DL model convergence is of high importance for DL study reproducibility. To enhance confidence in any SE study with reported good effectiveness, it is necessary to justify the parameters in the used DL model training and verify the convergence level of the used DL model.

Table 12. Statistics of change rates (CR) between ten experimental results on four DL models, where 'CR' indicates the change rate before and after more iterations or epochs of training; min, max, mean, and std indicate the minimim, maximum, average, standard deviation of the change rates respectively.

Model	CR_{min}	CR_{max}	CR_{mean}	CR_{std}
DeepCS	-9.1%	+8.5%	+0.3%	+6.0%
RRGen	-4.0%	+14.5%	+1.2%	+9.3%
RLNN	-2.0%	+0.4%	-0.9%	+0.8%
ASTNN	-0.2%	+0.1%	-0.0%	+0.1%

7.3 RQ6. How does the Out-of-Vocabulary Issue Affect Reproducibility?

Motivation. As described in Section 4.3, most of the DL-based studies in SE (76.3% of reviewed literature) address code/text by using a limited vocabulary size. Usually, the vocabulary only contains the top frequently occurred words. However, when addressing code/text representation, DL models commonly meet the out-of-vocabulary (OOV) issue where the vocabulary built from training data does not cover the words in testing data. This is because their testing data is collected from real-world environments e.g., GitHub and Stack Overflow, and is thus growing and changing frequently and new words will increase substantially as time goes by. Under this situation, a DL model can not capture the semantics of new words and may misunderstand the semantics of code/text. Therefore, DL models may substantially suffer from the OOV issue with different experimental settings, leading to low reproducibility.

Method. To investigate the impact of this OOV issue, we trained four DL models (DeepCS, RRGen, RLNN, and ASTNN) with different sizes of vocabulary and performed predictions on their corresponding testing data. If the impact is low, it is expected that the model performance shows little variation as the vocabulary size increases. Otherwise, one good reported model performance may not be reproduced in new experiments. This is because the DL model is likely misunderstand the semantics of code/text with new words, resulting in substantially poorer performance.

Specifically, the vocabulary size is determined by ten scales of the default vocabulary used in the original studies. The division scale ranges from 10% to 100% with a step 10%. The full scale (100%) indicates that we used the vocabulary with the default setting in the original studies. Table 9 shows that four DL models contain 10k, 10k, 50k, and 8k most frequently occurred words in their training data, respectively. However, we note that the original DL studies do not use all words in the training data to build their vocabularies, hence even the 100% scale does not mean that the OOV issue does not occur. Besides, when we trained a model with the vocabulary at a lower scale (e.g., 90%), we only excluded the (10%) words that have the least frequency in the vocabulary. If a vocabulary does not contain word frequency information, we collected the information from the training data by ourselves. In this way, we can avoid overestimating the effect of the OOV issue by suddenly dropping some frequently appearing words. After we obtained the vocabularies with lower scales, we addressed the encoded vectors in training and testing data one by one. For each vector, we removed the word that does not appear in a new vocabulary. Moreover, to mitigate the influence of model stability, we ran each model ten times for different vocabulary settings and reported the mean values of model performance following the setting described in Section 7.1.

Results. Table 13 lists the average performance of four DL models trained with different vocabulary scales in ten repeated runs. The 'improve' rows show the performance improvement from a certain scale (e.g., 90%) to the initial scale 10%. Table 13 shows that the performance of DeepCS keeps improving from 0.262 to 0.5 as the

vocabulary scale increases, where the relative improvement over the 10% scale ranges from 3.4% to 104.2%. For the RRGen model, when increasing the vocabulary scale from 10% to 20%, the model performance reduces by 1.2%. However, the RRGen gains improvements (6.8% to 17.6%) when the vocabulary scale keeps increasing from 30% to 90%, where the performance ranges from 32.144 to 35.394. The maximum performance occurs at the 70% scale of vocabulary size. The performance of the RLNN model ranges from 31.685 to 32.877 with no substantial improvement or deterioration. The performance of ASTNN (around 0.98) did not change with the vocabulary scale ranging from 20% to 100%.

To analyze the trend – increasing, decreasing, or neither – of the DL model performance when increasing the vocabulary scales, we performed a Cox Stuart trend test [25] at a 5% significance level. Table 14 illustrates the trendlines and statistical results, where '↑' indicates an increasing trend while '-' denotes no increasing/decreasing trend. c_v is the coefficient of variation (i.e., the standard deviation normalized by the mean) to measure the relative variation between model performance. 'scale(s)' means that for each listed vocabulary scale the model performance shows the significant difference compared with the performance at the previous scale, where the difference is tested by the Wilcoxon signed-rank test [143] at a 5% significance level.

The statistical results in Table 14 show that the performance of DeepCS is increasing (p-value < 0.05) for larger-scale vocabulary scales. **This means that DeepCS would suffer from the OOV issue with a limited size of vocabulary**, unless we enlarge the size as much as possible. The c_v value indicates that when increasing the vocabulary scale from 10% to 100%, the mean performance can be improved by more than 23.5%, where the significant improvements occur at the scales of 30%, 40%, and 90%. Therefore, **the OOV issue could strongly affect model performance, resulting in low study reproducibility**.
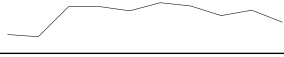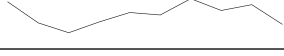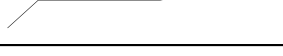
The performance of RRGen tends to be neither increasing nor decreasing (p-value > 0.05). Comparing with DeepCS, the performance variation (c_v = 5.9%) of RRGen is 74.9% smaller. A significant change only happened at the 30% scale. By excluding the scales 10% and 20%, the c_v drops to only 2.9%. Thus, **the new out-of-vocabulary words do not affect the performance of RRGen anywhere near as severely as DeepCS**. The performance of RLNN also shows no upward/downward trend (p-value > 0.05) as RRGen, with the variation in the degree of performance is smaller (c_v = 1.2%). There is some significant performance turbulence at the vocabulary scales 30%, 40%, 50%, and 70%. Therefore, **the OOV issue generates a less negative effect on the RLNN, compared with DeepCS and RRGen**. The performance of ASTNN is proportional to the vocabulary scale (p-value < 0.05) with the smallest variation (c_v = 0.2%). A significant improvement only takes place at the 20% scale. For the vocabulary scales larger than 10%, the c_v is reduced to no more than 0.01%. Thus, the performance of ASTNN converges to 0.981 for higher vocabulary scales. **This means that the ASTNN is not affected by the OOV issue and the reported performance of ASTNN can be reproduced for datasets with different scales of new words**. We observed that the reproducibility of ASTNN is not influenced by the OOV issue because it represents code as an abstract syntax tree (AST). In this way, the code classification task is not only affected by the code semantics but also strongly affected by the code structure. Moreover, RLNN can largely mitigate the effect of the OOV issue since it leverages the pointer generator technique [123] to replace the OOV words with the ones in training data in an appropriate way.

Implications. The OOV issue is prevalent when modeling code or text in SE tasks. This issue can strongly influence the DL model performance for new settings involving occurrences of new words that do not appear in the training data. Thus, the sensitivity of DL model performance to vocabulary size can lead to low study reproducibility. Therefore, DL-based studies should address the OOV issue and mitigate its negative effect as much as possible.

Table 13. Performance of four DL models trained with different sizes of vocabulary, where 'average' indicate the mean performance of a model with one vocabulary scale running in ten times; 'improve' indicates the percent of improvement comparing to the average performance with the 10% scale.

Model	Type	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
DeepCS	Average	0.262	0.271	0.324	0.421	0.446	0.445	0.470	0.463	0.520	0.540
	Improve	-	+3.4%	+23.7%	+60.7%	+70.2%	+69.8%	+79.4%	+76.7%	+98.5%	+104.2%
RRGen	Average	30.085	29.714	33.232	34.753	34.039	35.394	34.846	33.238	34.146	32.144
	Improve	-	-1.2%	+15.5%	+15.5%	+13.1%	+17.6%	+15.8%	+10.5%	+13.5%	+6.8%
RLNN	Average	32.766	32.031	31.685	32.064	32.393	32.307	32.877	32.465	32.668	31.982
	Improve	-	-2.2%	-3.3%	-2.1%	-1.1%	-1.4%	+0.3%	-0.9%	-0.3%	-2.4%
ASTNN	Average	0.975	0.980	0.980	0.980	0.980	0.980	0.981	0.981	0.981	0.981
	Improve	-	+0.5%	+0.5%	+0.5%	+0.5%	+0.5%	+0.5%	+0.5%	+0.5%	+0.5%

Table 14. Performance Trends of the four DL models trained with difference scales of vocabulary. The trend is determined by the Cox Stuart trend test at a 5% significance level, where '↑' indicates an increasing trend while '−' indicates no trend. For the trendlines, X axes indicate the vocabulary scale (ranging from 10% to 100% with a step 10%) while Y axes indicate the model performance. c_v is the coefficient of variation on model performance at different vocabulary scales, equaling to mean divided by standard deviation. 'Scale(s)' indicates that at each listed vocabulary scale the model performance is significantly different from the one with the previous scale, where the difference is tested by the Wilcoxon signed-rank test at a 5% significance level.

Model	Trend (p-value)	Trendline	c_v	Scale(s)
DeepCS	↑ (0.03*)		23.5%	30%,40%,90%
RRGen	− (0.50)		5.9%	30%
RLNN	− (0.50)		1.2%	30%,40%,50%,70%
ASTNN	↑ (0.03*)		0.2%	20%

7.4 RQ7. How does Testing Data Size Affect Reproducibility?

Motivation. It is common to verify the validity of a DL model by collecting a subset of data from the real-world environment, e.g., GitHub and Stack Overflow. However, researchers usually start an experiment with a limited size of testing data and assume that the model performance can be reproduced for a larger scale of testing data. However this assumption is rarely investigated. Therefore, our goal is to analyze how our four SE study DL models perform with different sizes of testing data, and further investigate how the testing data size affects DL-based study reproducibility.

Method. To analyze the effect of testing data size, we trained each DL model as default (namely, the training data does not change) but performed predictions on different sizes of testing data. Specifically, for the models for DeepCS, RRGen, RLNN, and ASTNN, they contain around 16m, 14k, 4k, and 10k testing data in default as shown in Table 9. We sampled a subset of testing data at ten different scales, ranging from 10% to 100% with a step 10%. As the sampling process for each scale is random, we repeated the sampling and testing ten times to

mitigate the effect of randomness. If the model performance is sensitive to the testing data scale, DL-based SE study reproducibility will be compromised. Therefore, it is expected that the performance of a DL model will not decrease substantially when the scale of testing data increases.

Results. Table 15 lists the average performance of DL models tested on different scales of testing data in ten repeated runs. The 'improve' rows show the performance improvement from a certain scale (e.g., 80% or 90%) to the initial scale 10%. Table 15 shows that when the testing data size is increased, the DeepCS performs the worst (0.531) at the 30% scale and the best (0.581) at the 80% scale. The performance of RRGen did not change largely from the scale 10% to 80%, ranging from 29.128 to 29.404. However its performance is largely improved at scales 90% and 100% (improved by 6.0% and 9.7%, compared with its performance at the 10% scale.) For the models, RLNN and ASTNN, Table 15 shows that their performance is hardly affected by the scale of their testing data, where the performance of RLNN is varied by no more than 1.3% while ASTNN shows negligible performance difference (<0.2%) for different scales of testing data.

Similar to Section 7.3, we performed a Cox Stuart trend test [25] at a 5% significance level on the DL model performance at different scales of testing data to analyze their performance trend – increasing, decreasing, or neither. We also calculated the c_v values to measure the variation of performance. We performed a Wilcoxon signed-rank test at a 5% significance level and recorded the scales at which the model performance is significantly different from the one in the previous scale.

The trendlines and statistical results in Table 16 indicate that **all four models (DeepCS, RRGen, RLNN, and ASTNN) show neither increasing nor decreasing trend with statistical significance** (p-values > 0.05). However their performance variation does differ. Specifically, the c_v of DeepCS is 4% with significant changes at the scales 50% and 90%. Compared with the other three DL models, DeepCS obtains a higher variation. We observe that this variation is mainly caused by the quality of the testing data used. This is because for a search query the number of ground-truth may be substantially changed due to random sampling. RRGen shows smaller performance variation with c_v = 3.5% with no significant change at any scale. Similarly to DeepCS, RRGen achieves varied performance at different testing data scales because the difficulty of review response generation could be largely varied for different testing data. Thus **the reproducibility of studies using a DeepCS or RRGen DL model may be low with different sizes of testing data used**. In contrast, the performance turbulence of RLNN is reduced by 80% in terms of c_v (0.7%) even if there is a significant improvement at the 100% scale. We observed that although RLNN is used for a generation task similar to RRGen, **its performance is not as sensitive to the testing data size as RRGen**, as RRGen leverages a reinforcement learning technique to optimize the generation result. Similarly, the c_v of ASTNN is 0%, even if there is a significant improvement at 20%. It means that the model performance converges. The model performance ASTNN is not sensitive to the testing data size because its performance is very high with accuracy 98.1%. Hence, no matter how we sampled a subset of testing data, the prediction accuracy would not differ. Therefore, if the characteristics of the testing data do not change, **the reproducibility of studies using ASTNN should be high**.

Implications. Testing DL-based models with different sizes of testing data may achieve substantially different model performance. In this case, a reported performance can be difficult to reproduce in new experiments when using newly sampled testing data. Therefore, mitigating the negative effect of the testing data size should be fully considered in DL-based SE studies.
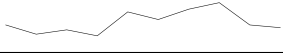
8 THREATS TO VALIDITY

This validity of this study could be affected by the following threats.

Manual Efforts in Literature Review. Like any manual evaluation, our literature review is subject to personal bias and subjectivity. As our research topic is on DL-based studies in SE, we performed our literature search only

Table 15. Performance of four DL models tested on different sizes of testing data, where 'average' indicate the mean performance of a model with one testing data scale running in ten times; 'improve' indicates the percent of improvement comparing to the average performance with the 10% scale.

Model	Type	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
DeepCS	Average	0.540	0.523	0.531	0.520	0.564	0.550	0.569	0.581	0.566	0.535
	Improve	-	-3.1%	-1.7%	-3.7%	+4.4%	+1.9%	+5.4%	+7.6%	+0.0%	-0.9%
RRGen	Average	29.290	29.218	29.396	29.373	29.316	29.128	29.326	29.404	31.305	32.144
	Improve	-	-0.2%	+0.4%	+0.3%	+0.1%	-0.6%	+0.1%	+0.4%	+6.9%	+9.7%
RLNN	Average	31.556	31.528	31.086	31.611	31.554	31.385	31.447	31.532	31.554	31.982
	Improve	-	-0.1%	-1.5%	+0.2%	+0.0%	-0.5%	-0.3%	-0.1%	+0.0%	+1.3%
ASTNN	Average	0.980	0.981	0.981	0.981	0.981	0.981	0.981	0.981	0.981	0.981
	Improve	-	+0.1%	+0.1%	+0.1%	+0.1%	+0.1%	+0.1%	+0.1%	+0.1%	+0.1%

Table 16. Performance Trends of the four DL models tested on difference scales of testing data. The trend is determined by the Cox Stuart trend test at a 5% significance level, where '↑' indicates an increasing trend while '−' indicates no trend. For the trendlines, X axes indicate the testing data scale (ranging from 10% to 100% with a step 10%) while Y axes indicate the model performance. c_v is the coefficient of variation on model performance at different scales of testing data, equaling to mean divided by standard deviation. 'Scale(s)' indicates that at each listed scale of testing data the model performance is significantly different from the one with the previous scale, where the difference is tested by the Wilcoxon signed-rank test at a 5% significance level.

Model	Trend (p-value)	Trendline	c_v	Scale(s)
DeepCS	− (0.19)		4.0%	50%,90%
RRGen	− (0.50)		3.5%	-
RLNN	− (0.50)		0.7%	100%
ASTNN	− (0.50)		0.0%	20%

on twenty selected SE journals or conferences. Any DL-based SE studies outside of this search scope would be excluded. We believe this threat should be minor because these selections are the most commonly published SE venues. To analyze how often DL studies provide replication packages, we checked all the links in the paper one by one. We may have missed replication packages not mentioned in the paper but existing on the Internet, e.g., GitHub. To analyze the characteristics of the reviewed DL studies, we manually classified them into ten different categories, e.g., study subjects, SE tasks, basic DL techniques, and types of RQs discussed in each paper. These manual efforts may involve some interpretation or mis-classification errors that threaten the validity of our findings.

Limited Experiments. To estimate the importance of the DL replicability/reproducibility, we conducted experiments on four representative DL models used in four representative SE tasks. Our experiments involve four prevalent factors that could strongly affect the DL replicability/reproducibility − model stability, convergence, OOV issue, and the testing data size. This investigation on DL replicability/reproducibility may not be exhaustive,

but they are the most commonly discussed factors in SE as described in Section 4.3, and hence are enough to show the importance of the DL replicability/reproducibility issue. We selected only four DL models for use in our experiments. Running more DL models would help researchers better understand the DL replicability/reproducibility issues for other SE tasks and other DL- based models. However, running DL models are time-consuming and we have limited computational resources ourselves. We therefore presented an exploratory meta-analysis in this study to help researchers understand the prevalence and importance of replicability/reproducibility issues. Due to the same reason, we ran each DL model ten times for each research question. In the future, we plan to replicate more DL studies and investigate more influential factors relating to DL replicability/reproducibility.

Manual Evaluation. When we re-ran the code search model DeepCS [44], the relevance of top-10 returned code methods to a query requires manual identification, which could suffer from subjectivity bias. To mitigate this threat, the manual analysis was performed by two independent experienced developers. If a conflict occurred, it was resolved by an open discussion. To save manual efforts, the relevancy of code-query pair was labeled by a script if that relevancy has been identified before.

9 DISCUSSION

Our experimental findings in Sections 4 and 7 imply that replicability and reproducibility issues are prevalent and important for DL-based studies in SE. To mitigate these replicability and replicability issues, DL studies in SE should consider the following guidelines:

Provide a long-lasting link for a replication package. An accessible replication package with source code and data can substantially facilitate study replication, help other researchers have a deeper understanding of the DL model used, and largely mitigate manual errors in any replication study.

Estimating stability and convergence of DL models. Reporting only one good experimental result likely threatens the DL model replicability due to its unverified stability and convergence. It is recommended to estimate the model stability by running it multiple times and assess the model convergence by reporting model performance at different optimization iterations or epochs.

Enhance stability and convergence of DL models. DL model stability and convergence can be improved by choosing an appropriate DL initialization method and training the model until convergence, instead of a fixed number of optimization iterations.

Use an automated evaluation approach to avoid the effect of human bias and subjectivity. The replicability of a SE study using a DL model could be compromised if the evaluation involves any manual process. Although many DL studies leverage some methods to mitigate the effect of this issue, a better way is to design an automated evaluation approach.

Mitigate the effect of OOV issue for code/text representation learning. The OOV issue is prevalent in many SE tasks and the prevalent solutions are to replace new words by existing words in the vocabulary [95], transforming new words [50], or build a character level vocabulary instead of the word level one [67].

Measure DL-based study reproducibility on different testing data. Study reproducibility requires that an experimental finding can be reproduced using different sampled testing data. Thus, one reported result on only one testing dataset will be not enough. It is suggested to test the model using different sizes of testing data and analyze the model performance.

Mitigate DL-based SE study reproducibility issues on different testing data. DL model performance commonly varies for different testing data. To mitigate this issue, it is recommended to incorporate some more robust features for representation learning, such as using abstract syntax trees [154].

Improve DL model efficiency. Time-consuming model optimization is a large obstacle when DL-based studies investigate model replicability and reproducibility. Therefore, reducing model complexity and accelerating model optimization will substantially promote the replicability and reproducibility studies.

10 CONCLUSION

Replicability and reproducibility are important for scientific research but DL-based studies in SE often ignore or minimalise them. To investigate the merit of SE study replicability/reproducibility using DL-based models, we first analyzed the characteristics of DL-based studies in SE. Specifically, we conducted a literature review on DL studies that are published on twenty prevalent SE venues in the recent five years. The observation from 93 reviewed literature shows that more than 70% of DL-based studies were published in the last two years. Most of the studies leverage DL techniques to better learn semantics in code/text, and CNN and RNN based models are the most frequently used DL techniques. Reinforcement learning is begining to show its potential to further improve the performance of existing DL-based models.

To assess the prevalence of replicability/replicability issues in SE studies using DL, we checked all the links in the reviewed DL-based studies and found that only 25.8% of studies provided accessible links to their replication packages. Thus, the DL replicability cannot be ensured because other researchers need to replicate existing studies from scratch and may miss important implementation details. We also counted the percentage of DL studies that investigated any research questions on the replicability/replicability issues, and found that only 10.8% of studies considered these issues. Therefore, the replicability/replicability issue is prevalent, and it is suggested for further DL-based studies in SE to share their source code and data publicly and better analyze their DL-based study replicability/replicability issues to better support their DL model validity.

To investigate the importance of replicability/replicability, we performed four experiments on four representative DL models used for four representative SE tasks. These studies were published in top SE venues in the last two years with accessible replication packages. Our experiments considered how the model stability affects DL replicability, and analyzed how model convergence, the out-of-vocabulary issue, and testing data size all impact on DL-based SE study reproducibility. Our experimental results show that DL models possess randomness by nature so that study replicability can be compromised for an unstable model. A DL model with low convergence level can produce highly turbulent and unstable performance, leading to low study reproducibility. The performance of a DL model can be highly sensitive to the size of vocabulary for training and the scale of testing data used. Thus a good reported model performance can be hard to reproduce for a new experimental dataset. Therefore, it is recommended for the SE community to pay more attention to these influential factors that may cause low DL-based study replicability/replicability. We also need to investigate viable solutions to strengthen study DL-based model validity instead of just presenting the advantages of chosen DL model effectiveness.

REFERENCES

[1] Aysh Al-Hroob, Ayad Tareq Imam, and Rawan Al-Heisa. 2018. The use of artificial neural networks for extracting actions and actors from requirements document. *Information and Software Technology* 101 (2018), 1–15.

[2] Hamdi A Al-Jamimi and Moataz Ahmed. 2013. Machine learning-based software quality prediction models: state of the art. In *2013 International Conference on Information Science and Applications (ICISA)*. IEEE, 1–4.

[3] Sven Amann, Stefanie Beyer, Katja Kevic, and Harald Gall. 2013. Software mining studies: Goals, approaches, artifacts, and replicability. In *Software Engineering*. Springer, 121–158.

[4] Bente CD Anda, Dag IK Sjøberg, and Audris Mockus. 2008. Variability and reproducibility in software engineering: A study of four companies that developed the same system. *IEEE Transactions on Software Engineering* 35, 3 (2008), 407–429.

[5] Anders Arpteg, Björn Brinne, Luka Crnkovic-Friis, and Jan Bosch. 2018. Software engineering challenges of deep learning. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 50–59.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[7] Antoine Barbez, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2019. Deep Learning Anti-patterns from Code Metrics History. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 114–124.

[8] Raja Ben Abdessalem, Shiva Nejati, Lionel C Briand, and Thomas Stifter. 2016. Testing advanced driver assistance systems using multi-objective search and neural networks. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. 63–74.

[9] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. In *Noise reduction in speech processing*. Springer, 1–4.

[10] Sahil Bhatia, Pushmeet Kohli, and Rishabh Singh. 2018. Neuro-symbolic program corrector for introductory programming assignments. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 60–70.

[11] Manjubala Bisi and Neeraj Kumar Goyal. 2016. Software development efforts prediction using artificial neural network. *IET Software* 10, 3 (2016), 63–71.

[12] John E Boylan, Paul Goodwin, Maryam Mohammadipour, and Aris A Syntetos. 2015. Reproducibility in forecasting research. *International Journal of Forecasting* 31, 1 (2015), 79–90.

[13] António Branco, Kevin Bretonnel Cohen, Piek Vossen, Nancy Ide, and Nicoletta Calzolari. 2017. Replicability and reproducibility of research results for human language technology: Introducing an LRE special section.

[14] Lutz Büch and Artur Andrzejak. 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)*. IEEE, 95–104.

[15] Jose Cambronero, Hongyu Li, Seohyun Kim, Koushik Sen, and Satish Chandra. 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*. 964–974.

[16] Jeffrey C Carver, Natalia Juristo, Maria Teresa Baldassarre, and Sira Vegas. 2014. Replications of software engineering experiments.

[17] Chao Chen, Wenrui Diao, Yingpei Zeng, Shanqing Guo, and Chengyu Hu. 2018. DRLgencert: Deep learning-based automated testing of certificate verification in SSL/TLS implementations. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 48–58.

[18] Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. 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*. 665–676.

[19] Guibin Chen, Chunyang Chen, Zhenchang Xing, and Bowen Xu. 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)*. IEEE, 744–755.

[20] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. Continuous incident triage for large-scale online service systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 364–375.

[21] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Trang Pham, Aditya Ghose, and Tim Menzies. 2018. A deep learning model for estimating story points. *IEEE Transactions on Software Engineering* 45, 7 (2018), 637–656.

[22] Jin Young Choi and Chong-Ho Choi. 1992. Sensitivity analysis of multilayer perceptron with differentiable activation functions. *IEEE Transactions on Neural Networks* 3, 1 (1992), 101–107.

[23] Jürgen Cito, Vincenzo Ferme, and Harald C Gall. 2016. Using docker containers to improve reproducibility in software and web engineering research. In *International Conference on Web Engineering*. Springer, 609–612.

[24] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of machine learning research* 12, Aug (2011), 2493–2537.

[25] David Roxbee Cox and Alan Stuart. 1955. Some quick sign tests for trend in location and dispersion. *Biometrika* 42, 1/2 (1955), 80–95.

[26] Fabio QB Da Silva, Marcos Suassuna, A César C França, Alicia M Grubb, Tatiana B Gouveia, Cleviton VF Monteiro, and Igor Ebrahim dos Santos. 2014. Replication of empirical studies in software engineering research: a systematic mapping study. *Empirical Software Engineering* 19, 3 (2014), 501–557.

[27] Hoa Khanh Dam, Truyen Tran, Trang Thi Minh Pham, Shien Wee Ng, John Grundy, and Aditya Ghose. 2018. Automatic feature learning for predicting vulnerable software components. *IEEE Transactions on Software Engineering* (2018).

[28] Thomas Deselaers, Saša Hasan, Oliver Bender, and Hermann Ney. 2009. A deep learning approach to machine transliteration. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*. Association for Computational Linguistics, 233–241.

[29] Jayati Deshmukh, Sanjay Podder, Shubhashis Sengupta, Neville Dubash, et al. 2017. Towards accurate duplicate bug retrieval using deep learning techniques. In *2017 IEEE International conference on software maintenance and evolution (ICSME)*. IEEE, 115–124.

[30] Bui Nghi DQ, Yijun Yu, and Lingxiao Jiang. 2019. Bilateral dependency neural networks for cross-language algorithm classification. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 422–433.

[31] Terrence L Fine. 2006. *Feedforward neural network methodology*. Springer Science & Business Media.

[32] Thomas Fischer and Christopher Krauss. 2018. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research* 270, 2 (2018), 654–669.

[33] David A Forsyth and Jean Ponce. 2002. *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference.

[34] Cuiyun Gao, Jichuan Zeng, Xin Xia, David Lo, Michael R Lyu, and Irwin King. 2019. Automating App Review Response Generation. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 163–175.

[35] Sa Gao, Chunyang Chen, Zhenchang Xing, Yukun Ma, Wen Song, and Shang-Wei Lin. 2019. A neural model for method name generation from functional description. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 414–421.

[36] Yi Gao, Zan Wang, Shuang Liu, Lin Yang, Wei Sang, and Yuanfang Cai. [n.d.]. TECCD: A Tree Embedding Approach for Code Clone Detection. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 145–156.

[37] Yongxin Ge, Min Chen, Chao Liu, Feiyi Chen, Sheng Huang, and Hongxing Wang. 2018. Deep metric learning for software change-proneness prediction. In *International Conference on Intelligent Science and Big Data Engineering*. Springer, 287–300.

[38] Patrice Godefroid, Hila Peleg, and Rishabh Singh. 2017. Learn&fuzz: Machine learning for input fuzzing. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 50–59.

[39] Anthony TC Goh. 1995. Back-propagation neural networks for modeling complex systems. *Artificial Intelligence in Engineering* 9, 3 (1995), 143–151.

[40] Omar S Gómez, Natalia Juristo, and Sira Vegas. 2010. Replications types in experimental disciplines. In *Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement*. 1–10.

[41] Omar S Gómez, Natalia Juristo, and Sira Vegas. 2014. Understanding replication of experiments in software engineering: A classification. *Information and Software Technology* 56, 8 (2014), 1033–1048.

[42] Jesús M González-Barahona and Gregorio Robles. 2012. On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. *Empirical Software Engineering* 17, 1-2 (2012), 75–89.

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning.* MIT press.

[44] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 933–944.

[45] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep API learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 631–642.

[46] Chenkai Guo, Dengrong Huang, Naipeng Dong, Quanqi Ye, Jing Xu, Yaqing Fan, Hui Yang, and Yifan Xu. 2019. Deep review sharing. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 61–72.

[47] Chenkai Guo, Weijing Wang, Yanfeng Wu, Naipeng Dong, Quanqi Ye, Jing Xu, and Sen Zhang. 2019. Systematic comprehension for developer reply in mobile system forum. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 242–252.

[48] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. 2017. Semantically enhanced software traceability using deep learning techniques. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 3–14.

[49] Huong Ha and Hongyu Zhang. 2019. DeepPerf: performance prediction for configurable software with deep sparse neural network. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1095–1106.

[50] Nizar Habash. 2008. Four techniques for online handling of out-of-vocabulary words in Arabic-English statistical machine translation. In *Proceedings of ACL-08: HLT, Short Papers*. 57–60.

[51] Junxiao Han, Emad Shihab, Zhiyuan Wan, Shuiguang Deng, and Xin Xia. [n.d.]. What do Programmers Discuss about Deep Learning Frameworks. ([n. d.]).

[52] Zhuobing Han, Xiaohong Li, Zhenchang Xing, Hongtao Liu, and Zhiyong Feng. 2017. Learning to predict severity of software vulnerability using only vulnerability description. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 125–136.

[53] Vincent J Hellendoorn, Christian Bird, Earl T Barr, and Miltiadis Allamanis. 2018. Deep learning type inference. In *Proceedings of the 2018 26th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering*. 152–162.

[54] Vincent J Hellendoorn and Premkumar Devanbu. 2017. Are deep neural networks the best choice for modeling source code?. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 763–773.

[55] Thong Hoang, Julia Lawall, Yuan Tian, Richard J Oentaryo, and David Lo. 2019. PatchNet: Hierarchical Deep Learning-Based Stable Patch Identification for the Linux Kernel. *IEEE Transactions on Software Engineering* (2019).

[56] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[57] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *Proceedings of the 26th Conference on Program Comprehension*. 200–210.

[58] Qiao Huang, Xin Xia, David Lo, and Gail C Murphy. 2018. Automating intention mining. *IEEE Transactions on Software Engineering* (2018).

[59] Xuan Huo, Ferdian Thung, Ming Li, David Lo, and Shu-Ting Shi. 2019. Deep transfer bug localization. *IEEE Transactions on Software Engineering* (2019).

[60] Jarmo Ilonen, Joni-Kristian Kamarainen, and Jouni Lampinen. 2003. Differential evolution training algorithm for feed-forward neural networks. *Neural Processing Letters* 17, 1 (2003), 93–105.

[61] Darrel C Ince, Leslie Hatton, and John Graham-Cumming. 2012. The case for open computer programs. *Nature* 482, 7386 (2012), 485–488.

[62] Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically generating commit messages from diffs using neural machine translation. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 135–146.

[63] Natalia Juristo and Omar S Gómez. 2010. Replication of software engineering experiments. In *Empirical software engineering and verification*. Springer, 60–88.

[64] Natalia Juristo and Sira Vegas. 2011. The role of non-exact replications in software engineering experiments. *Empirical Software Engineering* 16, 3 (2011), 295–324.

[65] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188* (2014).

[66] Deborah S Katz, Jason Ruchti, and Eric Schulte. 2018. Using recurrent neural networks for decompilation. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 346–356.

[67] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[68] Barbara Kitchenham, Lech Madeyski, and Pearl Brereton. 2020. Meta-analysis for families of experiments in software engineering: a systematic review and reproducibility and validity assessment. *Empirical Software Engineering* 25, 1 (2020), 353–401.

[69] Jinkyu Koo, Charitha Saumya, Milind Kulkarni, and Saurabh Bagchi. 2019. PySE: Automatic Worst-Case Test Generation by Reinforcement Learning. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 136–147.

[70] Lov Kumar and Santanu Ku Rath. 2016. Hybrid functional link artificial neural network approach for predicting maintainability of object-oriented software. *Journal of Systems and Software* 121 (2016), 170–190.

[71] Jeremy Lacomis, Pengcheng Yin, Edward Schwartz, Miltiadis Allamanis, Claire Le Goues, Graham Neubig, and Bogdan Vasilescu. 2019. Dire: A neural approach to decompiled identifier naming. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 628–639.

[72] An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N Nguyen. 2017. Bug localization with combination of deep learning and information retrieval. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 218–229.

[73] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. 1997. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks* 8, 1 (1997), 98–113.

[74] Alexander LeClair, Zachary Eberhart, and Collin McMillan. 2018. Adapting neural text classification for improved software categorization. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 461–472.

[75] Alexander LeClair, Siyuan Jiang, and Collin McMillan. 2019. A neural model for generating natural language summaries of program subroutines. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 795–806.

[76] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[77] Seongmin Lee, Shin Hong, Jungbae Yi, Taeksu Kim, Chul-Joo Kim, and Shin Yoo. 2019. Classifying False Positive Static Checker Alarms in Continuous Integration Using Convolutional Neural Networks. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 391–401.

[78] Liuqing Li, He Feng, Wenjie Zhuang, Na Meng, and Barbara Ryder. 2017. Cclearner: A deep learning-based clone detection approach. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 249–260.

[79] Liam Li and Ameet Talwalkar. 2019. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638* (2019).

[80] Xiaochen Li, He Jiang, Dong Liu, Zhilei Ren, and Ge Li. 2018. Unsupervised deep bug report summarization. In *Proceedings of the 26th Conference on Program Comprehension*. 144–155.

[81] Xia Li, Wei Li, Yuqun Zhang, and Lingming Zhang. 2019. Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 169–180.

[82] Chin-Yew Lin and FJ Och. 2004. Looking for a few good metrics: ROUGE and its evaluation. In *Ntcir Workshop*.

[83] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. 2017. A survey on deep learning in medical image analysis. *Medical image analysis* 42 (2017), 60–88.

[84] Chao Liu, Dan Yang, Xin Xia, Meng Yan, and Xiaohong Zhang. 2018. Cross-project change-proneness prediction. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. IEEE, 64–73.

[85] Chao Liu, Dan Yang, Xin Xia, Meng Yan, and Xiaohong Zhang. 2019. A two-phase transfer learning model for cross-project defect prediction. *Information and Software Technology* 107 (2019), 125–136.

[86] Chao Liu, Dan Yang, Xiaohong Zhang, Haibo Hu, Jed Barson, and Baishakhi Ray. 2018. A recommender system for developer onboarding. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. 319–320.

[87] Chao Liu, Dan Yang, Xiaohong Zhang, Baishakhi Ray, and Md Masudur Rahman. 2018. Recommending GitHub Projects for Developer Onboarding. *IEEE Access* 6 (2018), 52082–52094.

[88] Chia-Wei Liu, Ryan Lowe, Iulian V Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. 2016. How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. *arXiv preprint arXiv:1603.08023* (2016).

[89] Hui Liu, Jiahao Jin, Zhifeng Xu, Yifan Bu, Yanzhen Zou, and Lu Zhang. 2019. Deep learning based code smell detection. *IEEE Transactions on Software Engineering* (2019).

[90] Hui Liu, Zhifeng Xu, and Yanzhen Zou. 2018. Deep learning based feature envy detection. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 385–396.

[91] Jin Liu, Pingyi Zhou, Zijiang Yang, Xiao Liu, and John Grundy. 2018. FastTagRec: fast tag recommendation for software information sites. *Automated Software Engineering* 25, 4 (2018), 675–701.

[92] Kui Liu, Dongsun Kim, Tegawendé F Bissyandé, Taeyoung Kim, Kisub Kim, Anil Koyuncu, Suntae Kim, and Yves Le Traon. 2019. Learning to spot and refactor inconsistent method names. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1–12.

[93] Peng Liu, Xiangyu Zhang, Marco Pistoia, Yunhui Zheng, Manoel Marques, and Lingfei Zeng. 2017. Automatic text input generation for mobile testing. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 643–653.

[94] Yibin Liu, Yanhui Li, Jianbo Guo, Yuming Zhou, and Baowen Xu. 2018. Connecting software metrics across versions to predict defects. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 232–243.

[95] Zhongxin Liu, Xin Xia, Christoph Treude, David Lo, and Shanping Li. 2019. Automatic Generation of Pull Request Descriptions. *arXiv preprint arXiv:1909.06987* (2019).

[96] Cuauhtémoc López-Martín and Alain Abran. 2015. Neural networks for predicting the duration of new software projects. *Journal of Systems and Software* 101 (2015), 127–135.

[97] Panos Louridas and Georgios Gousios. 2012. A note on rigour and replicability. *ACM SIGSOFT Software Engineering Notes* 37, 5 (2012), 1–4.

[98] Pablo Loyola and Yutaka Matsuo. 2017. Learning feature representations from change dependency graphs for defect prediction. In *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 361–372.

[99] Jonathan Lung, Jorge Aranda, Steve Easterbrook, and Gregory Wilson. 2008. On the difficulty of replicating human subjects studies in software engineering. In *2008 ACM/IEEE 30th International Conference on Software Engineering*. IEEE, 191–200.

[100] Suyu Ma, Zhenchang Xing, Chunyang Chen, Cheng Chen, Lizhen Qu, and Guoqiang Li. 2019. Easy-to-Deploy API Extraction by Multi-Level Feature Embedding and Transfer Learning. *IEEE Transactions on Software Engineering* (2019).

[101] Zaheed Mahmood, David Bowes, Tracy Hall, Peter CR Lane, and Jean Petrić. 2018. Reproducibility and replicability of software defect prediction studies. *Information and Software Technology* 99 (2018), 148–163.

[102] Ruchika Malhotra, Arvinder Kaur, and Yogesh Singh. 2010. Application of machine learning methods for software effort prediction. *ACM SIGSOFT Software Engineering Notes* 35, 3 (2010), 1–6.

[103] Rabee Sohail Malik, Jibesh Patra, and Michael Pradel. 2019. NL2Type: inferring JavaScript function types from natural language information. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 304–315.

[104] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. 55–60.

[105] Qing Mi, Jacky Keung, Yan Xiao, Solomon Mensah, and Yujin Gao. 2018. Improving code readability classification using convolutional neural networks. *Information and Software Technology* 104 (2018), 60–71.

[106] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

[107] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 5528–5531.

[108] Facundo Molina, Renzo Degiovanni, Pablo Ponzio, Germán Regis, Nazareno Aguirre, and Marcelo Frias. 2019. Training binary classifiers as data structure invariants. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 759–770.

[109] Kevin Moran, Carlos Bernal-Cárdenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk. 2018. Machine learning-based prototyping of graphical user interfaces for mobile apps. *arXiv preprint arXiv:1802.02312* (2018).

[110] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional neural networks over tree structures for programming language processing. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[111] Dongliang Mu, Wenbo Guo, Alejandro Cuevas, Yueqi Chen, Jinxuan Gai, Xinyu Xing, Bing Mao, and Chengyu Song. 2019. RENN: Efficient Reverse Execution with Neural-network-assisted Alias Analysis. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 924–935.

[112] Kawser Wazed Nafi, Tonny Shekha Kar, Banani Roy, Chanchal K Roy, and Kevin A Schneider. 2019. CLCDSA: Cross Language Code Clone Detection using Syntactical Features and API Documentation. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1026–1037.

[113] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. 2019. Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292* (2019).

[114] Amadeu Anderlin Neto. 2019. A Strategy to Support Replications of Controlled Experiments in Software Engineering. *ACM SIGSOFT Software Engineering Notes* 44, 3 (2019), 23–23.

[115] Anh Tuan Nguyen, Trong Duc Nguyen, Hung Dang Phan, and Tien N Nguyen. 2018. A deep neural network language model with contexts for source code. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 323–334.

[116] Tomaso Poggio, Kenji Kawaguchi, Qianli Liao, Brando Miranda, Lorenzo Rosasco, Xavier Boix, Jack Hidary, and Hrushikesh Mhaskar. 2017. Theory of deep learning III: explaining the non-overfitting puzzle. *arXiv preprint arXiv:1801.00173* (2017).

[117] Pooja Rani and GS Mahapatra. 2018. Neural network for software reliability analysis of dynamically weighted NHPP growth models with imperfect debugging. *Software Testing, Verification and Reliability* 28, 5 (2018), e1663.

[118] Xiaoxue Ren, Zhenchang Xing, Xin Xia, David Lo, Xinyu Wang, and John Grundy. 2019. Neural Network-based Detection of Self-Admitted Technical Debt: From Performance to Explainability. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 28, 3 (2019), 1–45.

[119] Stephen Romansky, Neil C Borle, Shaiful Chowdhury, Abram Hindle, and Russ Greiner. 2017. Deep green: Modelling time-series of software energy consumption. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 273–283.

[120] Hang Ruan, Bihuan Chen, Xin Peng, and Wenyun Zhao. 2019. DeepLink: Recovering issue-commit links based on deep learning. *Journal of Systems and Software* 158 (2019), 110406.

[121] Hasim Sak, Andrew W Senior, and Françoise Beaufays. 2014. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. (2014).

[122] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.

[123] Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368* (2017).

[124] Jianhang Shuai, Ling Xu, Chao Liu, Meng Yan, Xin Xia, and Yan Lei. [n.d.]. Improving Code Search with Co-Attentive Representation Learning. ([n. d.]).

[125] Dag IK Sjøberg, Jo Erskine Hannay, Ove Hansen, Vigdis By Kampenes, Amela Karahasanovic, N-K Liborg, and Anette C Rekdal. 2005. A survey of controlled experiments in software engineering. *IEEE transactions on software engineering* 31, 9 (2005), 733–753.

[126] Richard Szeliski. 2010. *Computer vision: algorithms and applications*. Springer Science & Business Media.

[127] Hannes Thaller, Lukas Linsbauer, and Alexander Egyed. 2019. Feature maps: A comprehensible software representation for design pattern detection. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 207–217.

[128] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*. 303–314.

[129] Haonan Tong, Bin Liu, and Shihai Wang. 2018. Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. *Information and Software Technology* 96 (2018), 94–111.

[130] Michele Tufano, Jevgenija Pantiuchina, Cody Watson, Gabriele Bavota, and Denys Poshyvanyk. 2019. On learning meaningful code changes via neural machine translation. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 25–36.

[131] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. 2018. Learning how to mutate source code from bug-fixes. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 301–312.

[132] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. 2019. An empirical study on learning bug-fixing patches in the wild via neural machine translation. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 28, 4 (2019), 1–29.

[133] Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, et al. 2018. Tensor2tensor for neural machine translation. *arXiv preprint arXiv:1803.07416* (2018).

[134] Yao Wan, Jingdong Shu, Yulei Sui, Guandong Xu, Zhou Zhao, Jian Wu, and Philip Yu. 2019. Multi-modal attention network learning for semantic source code retrieval. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 13–25.

[135] Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, and Philip S Yu. 2018. Improving automatic source code summarization via deep reinforcement learning. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 397–407.

[136] Song Wang, Taiyue Liu, Jaechang Nam, and Lin Tan. 2018. Deep semantic feature learning for software defect prediction. *IEEE Transactions on Software Engineering* (2018).

[137] Song Wang, Taiyue Liu, and Lin Tan. 2016. Automatically learning semantic features for defect prediction. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 297–308.

[138] Xu Wang, Chunyang Chen, and Zhenchang Xing. 2019. Domain-specific machine translation with recurrent neural network for software localization. *Empirical Software Engineering* 24, 6 (2019), 3514–3545.

[139] Yaohui Wang, Hui Xu, Yangfan Zhou, Michael R Lyu, and Xin Wang. 2019. Textout: Detecting Text-Layout Bugs in Mobile Apps via Visualization-Oriented Learning. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 239–249.

[140] Ming Wen, Rongxin Wu, and Shing-Chi Cheung. 2018. How well do change sequences predict defects? sequence learning from software changes. *IEEE Transactions on Software Engineering* (2018).

[141] Martin White, Michele Tufano, Matias Martinez, Martin Monperrus, and Denys Poshyvanyk. 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)*. IEEE, 479–490.

[142] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. 2016. Deep learning code fragments for code clone detection. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 87–98.

[143] Frank Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometrics bulletin* 1, 6 (1945), 80–83.

[144] Yan Xiao, Jacky Keung, Kwabena E Bennin, and Qing Mi. 2019. Improving bug localization with word embedding and enhanced convolutional neural networks. *Information and Software Technology* 105 (2019), 17–29.

[145] Rui Xie, Long Chen, Wei Ye, Zhiyu Li, Tianxiang Hu, Dongdong Du, and Shikun Zhang. 2019. DeepLink: A code knowledge graph based deep learning approach for issue-commit link recovery. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 434–444.

[146] Bowen Xu, Deheng Ye, Zhenchang Xing, Xin Xia, Guibin Chen, and Shanping Li. 2016. Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 51–62.

[147] Zhou Xu, Shuai Li, Jun Xu, Jin Liu, Xiapu Luo, Yifeng Zhang, Tao Zhang, Jacky Keung, and Yutian Tang. 2019. LDFR: Learning deep feature representation for software defect prediction. *Journal of Systems and Software* 158 (2019), 110402.

[148] Meng Yan, Mengning Yang, Chao Liu, and Xiaohong Zhang. 2016. Self-learning Change-prone Class Prediction. In *SEKE*. 134–140.

[149] Meng Yan, Xiaohong Zhang, Chao Liu, Ling Xu, Mengning Yang, and Dan Yang. 2017. Automated change-prone class prediction on unlabeled dataset using unsupervised method. *Information and Software Technology* 92 (2017), 1–16.

[150] Meng Yan, Xiaohong Zhang, Chao Liu, Jie Zou, Ling Xu, and Xin Xia. 2017. Learning to aggregate: an automated aggregation method for software quality model. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 268–270.

[151] Ruibo Yan, Xi Xiao, Guangwu Hu, Sancheng Peng, and Yong Jiang. 2018. New deep learning method to detect code injection attacks on hybrid applications. *Journal of Systems and Software* 137 (2018), 67–77.

[152] Hao Yu, Wing Lam, Long Chen, Ge Li, Tao Xie, and Qianxiang Wang. 2019. Neural detection of semantic code clones via tree-based convolution. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 70–80.

[153] Carmen Zannier, Grigori Melnik, and Frank Maurer. 2006. On the success of empirical studies in the international conference on software engineering. In *Proceedings of the 28th international conference on Software engineering*. 341–350.

[154] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. 2019. A novel neural source code representation based on abstract syntax tree. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 783–794.

[155] Zhuo Zhang, Yan Lei, Xiaoguang Mao, and Panpan Li. 2019. CNN-FL: An effective approach for localizing faults using convolutional neural networks. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 445–455.

[156] Dehai Zhao, Zhenchang Xing, Chunyang Chen, Xin Xia, and Guoqiang Li. 2019. ActionNet: vision-based workflow action recognition from programming screencasts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 350–361.

[157] Gang Zhao and Jeff Huang. 2018. Deepsim: deep learning code functional similarity. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 141–151.

[158] Jinman Zhao, Aws Albarghouthi, Vaibhav Rastogi, Somesh Jha, and Damien Octeau. 2018. Neural-augmented static analysis of Android communication. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 342–353.

[159] Yan Zheng, Xiaofei Xie, Ting Su, Lei Ma, Jianye Hao, Zhaopeng Meng, Yang Liu, Ruimin Shen, Yingfeng Chen, and Changjie Fan. 2019. Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 772–784.

[160] Liu Zhongxin, Xia Xin, Lo David, Xing Zhenchang, E. Hassan Ahmed, and Li Shanping. 2019. Which Variables Should I Log? *IEEE Transactions on Software Engineering* (2019).

[161] Pingyi Zhou, Jin Liu, Xiao Liu, Zijiang Yang, and John Grundy. 2019. Is deep learning better than traditional approaches in tag recommendation for software information sites? *Information and software technology* 109 (2019), 1–13.

[162] Yu Zhou, Xin Yan, Wenhua Yang, Taolue Chen, and Zhiqiu Huang. 2019. Augmenting Java method comments generation with context information based on neural networks. *Journal of Systems and Software* 156 (2019), 328–340.