

On the Reproducibility and Replicability 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

Context: 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.

Objective: Although many DL studies have reported substantial advantages over other state-of-the-art models on effectiveness, they often ignore two factors: (1) *reproducibility* - whether the reported experimental results can be obtained by other researchers using authors' artifacts (i.e., source code and datasets) with the same experimental setup; and (2) *replicability* - whether the reported experimental result can be obtained by other researchers using their re-implemented artifacts with a different experimental setup. We observed that 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, unlike classical supervised machine learning (ML) methods (e.g., random forest). This study aims to investigate the urgency and importance of reproducibility and replicability for DL studies on SE tasks.

Method: In this study, we conducted a literature review on 147 DL studies recently published in 20 SE venues and 20 AI (Artificial Intelligence) venues to investigate these issues. We also re-ran four representative DL models in SE to investigate important factors that may strongly affect the reproducibility and replicability of a study.

Results: Our statistics show the urgency of investigating these two factors in SE, where only 10.2% of the studies investigate any research question to show that their models can address at least one issue of replicability and/or reproducibility. More than 62.6% of the studies do not even share high-quality source code or complete data to support the reproducibility of their complex models. Meanwhile, our experimental results show the importance of reproducibility and replicability, where the reported performance of a DL model could not be reproduced for an unstable optimization process. Replicability could be substantially compromised if the model training is not convergent, or if performance is sensitive to the size of vocabulary and testing data.

*Corresponding Author: Xin Xia.

Authors' addresses: Chao Liu, liuchao@zju.edu.cn, College of Computer Science and Technology, Zhejiang University, China, and PengCheng Laboratory, China; Cuiyun Gao, gaocuiyun@hit.edu.cn, Harbin Institute of Technology (Shenzhen), China; Xin Xia, Xia@monash.edu, Monash University, Australia; David Lo, davidlo@smu.edu.sg, Singapore Management University, Singapore; John Grundy, John.Grundy@monash.edu, Monash University, Australia; Xiaohu Yang, yangxh@zju.edu.cn, College of Computer Science and Technology, Zhejiang University, China.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2021 Association for Computing Machinery.

1049-331X/2021/1-ART1 \$15.00

<https://doi.org/10.1145/1122445.1122456>

Conclusion: It is urgent for the SE community to provide a long-lasting link to a high-quality reproduction 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

ACM Reference Format:

Chao Liu, Cuiyun Gao, Xin Xia, David Lo, John Grundy, and Xiaohu Yang. 2021. On the Reproducibility and Replicability of Deep Learning in Software Engineering. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (January 2021), 46 pages. <https://doi.org/10.1145/1122445.1122456>

1 INTRODUCTION

Deep learning (DL) has become a key branch of Machine Learning (ML) [70, 99, 158], and now is a core component in systems for many aspects of modern society, such as autonomous cars [173], medical image diagnosis [108], financial market prediction [46], etc. The popularity of DL technologies mainly derive from their success in computer vision [47, 171], natural language processing [37, 132], machine translation [42, 180], etc. Generally, DL models aim to facilitate representation learning by leveraging big data and a specially designed neural network [99], such as the Recurrent Neural Network (RNN) [134, 135] and the Convolutional Neural Network (CNN) [87, 95].

In recent years, DL models have been increasingly used in the Software Engineering (SE) domain for building things such as code search engines [59, 165, 181], code summarization tools [98, 182], vulnerability identification tools [53, 71], 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 [52, 113, 181, 182, 216]. 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 [9, 99, 112, 216].

Apart from effectiveness, *reproducibility* and *replicability* are also widely accepted as important considerations in scientific research [19, 20, 85, 129]. Researchers have defined reproducibility and replicability in various ways. To harmonize the contradictions in existing definitions, ACM recently (Aug. 24, 2020) updated the definitions¹ of reproducibility and replicability in a unified way. Reproducibility refers to whether the reported experimental results can be obtained by other researchers using authors' artifacts (i.e., source code and datasets) with the same experimental setup. Nonetheless, high reproducibility is just a requirement for the repeated experiment and it does not guarantee that the reported results represent reality under new experiments [8, 19, 85]. Therefore, replicability is required, which refers to whether the reported experimental result can be obtained by other researchers using their re-implemented artifacts with different experimental setup [85, 129]. Notably, reproducibility requires authors to share a reproduction package including source code and data [20, 85, 124], which can help other researchers to build confidence in the scientific merit of a reported result [19, 20, 103]. Although provision of a reproduction package is not a necessity for replicability, it can largely save other researchers' replication efforts due to the complexity of DL studies.

Despite the scientific merit of reproducibility and replicability, many DL-based studies in SE often ignore these two issues. It is uncertain the degree to which their reported experimental results can be reproduced or replicated. This is because although other researchers reproduce the model with the same source code provided by the authors, reproducibility could also be compromised due to the strong randomness in model training, where the model is commonly initialized with a random seed and optimized by a list of data batch shuffled with another random seed [103]; and the replicability affected by the changed experimental settings when re-implementing the target DL model, such as testing the model on a newly collected data when the original

¹<https://www.acm.org/publications/policies/artifact-review-and-badging-current>

study provides no dataset, training the model with more iterations to obtain the reported model performance, etc. Although some studies provided reproduction packages to support reproducibility, they missed the replicability issues and left solving them to future work [59, 98, 175]. The representative cases are that a reported result is difficult to be replicated due to the unaddressed high randomness in model training and the sampling errors in datasets. Previously, Mahmood et al. [129] carried out a related work on reproducibility and replicability. They checked model reproducibility/replicability by comparing the reported experimental results in 208 software defect prediction studies. They found that only 6% of the 208 studies were replicated in other studies, and it is difficult to confirm the consistency between original and replicated studies due to different performance measures. However, they did not analyze how the quality of reproduction packages and model features (e.g., randomness in model optimization) affect the model reproducibility and replicability, especially for the DL-based models that have been shown substantial advantages over traditional ML techniques in various SE tasks.

To understand the prevalence and importance of reproducibility and replicability 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 Reproducibility and Replicability. The first three RQs explore the prevalence of reproducibility and replicability issues via a detailed literature review.

- **RQ1. How are DL models used and evaluated in SE studies?** We performed a meta-analysis on 147 DL models published in the last five years in 20 SE venues and 20 AI venues. Our statistics show that 73.4% of them are published in the last two years, indicating DL's rising popularity in SE. 85% 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 reproduction packages to support the reproducibility of DL models?** We observed that *only* 37.4% of the reviewed DL studies provide publicly accessible and high-quality reproduction packages. We thus highly recommend that DL all studies must share source code and data to support their reproducibility, considering the complexity of the DL technique.
- **RQ3. What RQs do SE studies investigate related to DL reproducibility/replicability?** We found that *only* 10.2% of the DL studies investigated any RQs to show that their models can address at least one issue of reproducibility and/or replicability, including the model stability, optimization convergence, the out-of-vocabulary (OOV) issue, and the performance sensitivity to testing data scale. Therefore, the reproducibility and replicability of most SE DL-based studies is unknown and future DL studies in SE must pay more attention to these two factors.

Part II: Experiments on DL Reproducibility and Replicability. Although some DL studies investigated RQs related to DL reproducibility/replicability, as described in RQ3, we observed that they conducted these RQs mainly because they can address the issues of model stability, optimization convergence, the OOV issue, and the performance sensitivity. To analyze the importance of reproducibility and replicability, the RQs 4-7 rerun four typical DL models and investigate how these four issues affect reproducibility/replicability respectively. These four DL models include an information retrieval model DeepCS [59], an RNN based generation model RRGGen [49], a generation model based on reinforcement learning plus neural network (RLNN) [122], and a classification model ASTNN [207]. 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 and high-quality reproduction packages. Note that as our study focused on investigating whether the factors (e.g., the out-of-vocabulary issue and the testing data scale), not the re-implementation errors or differences, may strongly affect the replicability, the RQs 5-7 used the high-quality reproduction packages provided by the original studies as substitutions for the correctly re-implemented models.

Specifically, to measure the reproducibility of a study, researchers commonly re-run the reproduction package and compare their reproduced result with the one reported in the original paper. However, the stability in DL

models may affect the measurement, because their networks are initialized with random seeds and optimized by a list of randomly shuffled data batches. If the randomness is high, the measured reproducibility would be low as the original study usually reported the best result that can be reproduced with a rare chance. Due to the time-consuming training of DL models and limited computation resources, it is also impractical to re-run the reproduction package multiple times until obtaining the reported result. Thus, we investigated this in RQ4:

- **RQ4. How does model stability affect reproducibility?** After re-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 reproducibility, as the reported results would be reproduced with a low probability.

Different from reproducibility, DL-based study replicability focuses on the re-implemented models instead of the reproduction package and it allows the investigation of the impact of the changed experimental settings using the original or newly collected dataset. The basic requirement of replicability is that the re-implementation can obtain the reported result using the original dataset. Although a replication study may also have the same stability issues as a reproduction study, if the convergence of DL models is high the reported result can be replicated even with the extended training iterations (or epochs). However, we found that only 13.6% of our reviewed SE DL-based studies show that their model optimization can converge so that a reported result is easier to be replicated. Meanwhile, the other studies, which did not report the convergence level of their proposed models, commonly stop model optimization with manually set iterations or choose the best model on validation data. Our RQ5 investigated the effects of convergence. If the effect is high on the original dataset, it should be an important issue for the replication study on new datasets.

- **RQ5. How does model convergence affect replicability?** When training each DL model with more iterations on the dataset provided by the original study, 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 replicability, due to the low probability of reproducing a good reported result.

A higher requirement of replicability is that the reported result can be replicated by using a newly collected dataset, not just the original one. One case is that when optimizing the model with (new) training data, the out-of-vocabulary (OOV) issue widely occurs, as many of our reviewed studies apply a DL model for natural language processing. The OOV issue can be regarded as a sampling issue, where the words selected from the (newly sampled) training data cannot cover the words in the original/new testing data. To analyze the relationship between the OOV issue and replicability, we investigated this in our RQ6:

- **RQ6. How does the out-of-vocabulary issue affect replicability?** We used the dataset provided by the original study but train a model with different vocabulary sizes (ranging from 10% to 100% with a step 10%). We find that *the performance of one DL model is highly sensitive to the vocabulary size*, and whose performance tested on a smaller vocabulary size can be increased by 3.4% to 104.2% for larger sizes. The results imply a low replicability of this model as its reported performance cannot be reproduced under the new testing data with many out-of-vocabulary words. We also observed that the pointer generator [159] in RLNN [122] 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 [207] can substantially mitigate the influence of OOV.

Another case is that a reported result cannot be replicated on a newly collected testing data with larger scale. If the performance of a model is highly sensitive to the testing data scale, the answer to the investigated RQ

may not be replicable, e.g., it is unclear whether a proposed model outperforms the state-of-the-art baselines. In this case, the claimed outperformance can be caused by sampling errors of testing data. This is a common case because the reviewed DL studies would like to use the train/test split to estimate their model performance. However, if the testing data scale is not large enough, the reported good performance would have a small effect size and the model performance would be sensitive to the testing data scale. Although cross-validation is a better choice, most DL studies would not like to run the model multiple times but adopt a train/test split mainly because of the high model complexity with time-consuming training. The following RQ investigated the effect of the testing data size under the setting of train/test split:

- **RQ7. How does testing data size affect replicability?** 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% when increasing the testing data size. In this case, results for a model whose performance is highly sensitive to testing data size can not be satisfactorily replicated.

In summary, due to the high complexity of DL models with time-consuming training, many of our reviewed SE DL-based studies commonly estimate their models based on non-ideal experimental settings (e.g., with insufficient repetition) and report only their one good performance. In this case, the reported performance may be difficult to be reproduced or replicated. Our controlled experiments showed that DL-based study reproducibility could be affected by the quality of the reproduction package and model stability. Meanwhile, the low replicability could be caused by the low model convergence, the OOV issue, and the sensitivity to testing data scale. To strengthen DL reproducibility and replicability, it is essential and urgent for the SE community to consider the following experimental guidelines:

- *To ensure DL reproducibility*, it is recommended for all SE DL-based studies to enhance the DL stability (e.g., specifying the random seed for model initialization, and recording the random seed for shuffling data batches in model optimization) and provide an accessible, long-lasting, and high-quality reproduction package with complete data and source code with the same setting as described in the papers.
- *To support DL replicability*, SE studies using DL should not only boost the model stability (e.g., providing the random seeds for model initialization and optimization) and make sure the optimization converges at a high level, but also provide available data (i.e., training/testing data) to help other researchers check the correctness of their re-implementations and estimate the replicability on the original dataset.
- *To strengthen DL replicability*, DL studies should also estimate the replicability of the correctly re-implemented model on a new dataset (especially for larger scales) and consider mitigating the negative effects of the replicability issues (e.g., the low model convergence, the OOV issue, and the sensitivity to testing data scale) when applying the model on new datasets.
- *To make DL reproducibility/replicability meaningful*, DL studies need to estimate the reproducibility and replicability of their models with sufficient repetitions (e.g., using the cross-validation) in terms of the standard error and the effect size to ensure that the reported result is good not just because of the high randomness in model training and the used dataset (with sampling and measurement issues).
- *To promote DL reproducibility/replicability studies*, SE studies using DL should reduce model training time (e.g., reducing the model complexity or accelerating the training via the parallelization solutions) so that the burden of computation resources will not be an obstacle for reproducibility/replicability studies.

The main contributions of this study are:

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

- Providing some guidelines for the SE community to mitigate the reproducibility and replicability issues in many DL-based SE studies.
- Providing all the collected accessible links to reproduction packages for the reviewed studies in the reference to reduce future replication efforts.

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 [103, 122, 181]. For example, in many recent SE studies, a large number of challenges derive from the semantic comprehension of code in programming languages [110, 111, 200, 201, 207], text in natural languages [49, 202], or their mutual transformation [59]. As code and text involves some form of natural language processing (NLP), it commonly starts with encoding words by a fixed size of vocabulary [59]. Afterward, 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 [13, 71, 73, 122, 197].

The DL models used in SE are commonly based on three different types [68, 207, 211]. The first type is the traditional artificial neural network (ANN) that consists of fully connected neural networks, such as the multilayer perceptron (MLP) [34, 45, 54], deep belief network (DBN) [54], etc. To enhance representation learning, researchers developed two popular DL types. One is the convolutional neural network (CNN) [87, 95] that can sense regional characteristics of a matrix data via special convolution functions. The other is the recurrent neural network (RNN) [134, 135] 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 [75, 156].

By leveraging a DL model to learn the representation from the vectorized code/text, the model can be optimized for a specific SE task [16, 63]. 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 [216].

2.2 Reproducibility and Replicability in SE

Reproducibility and Replicability are considered to be key aspects of the scientific method [83, 83, 124]. This is because they help researchers confirm the merit of previous experimental findings and promote scientific innovations [124]. In the SE domain, reproducibility and replicability have been widely claimed as essential for all empirical studies due to the unsatisfactory quality of previous research findings [7, 26, 40, 55, 56, 86, 127]. However, they are rarely investigated [57, 90, 167, 206].

Based on the findings of empirical studies, a large proportion of recent SE studies leveraged ML techniques to solve SE tasks [2, 130]. When verifying the validity of newly proposed ML models, the replication of previous state-of-the-art baselines are required. However, the reproducibility and replicability 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 [129]. To improve the reproducibility of ML models, it

is suggested using replication infrastructures (e.g., OpenML²) to mitigate the replication efforts [129], leveraging the Docker container to save the experimental environment [35], and providing standard replication guidelines [129].

In recent years, many DL models have been successfully applied for diverse SE tasks [113, 181]. Although DL models often show substantial outperformance over traditional ML models, we noticed that they often overlooked reproducibility and replicability, 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 [58, 99]. Therefore, the requirement for DL reproducibility and replicability may have many differences. Due to the lack of study of DL reproducibility and replicability, we conducted this study.

3 RESEARCH QUESTIONS

We present our study's seven key RQs on the reproducibility and replicability 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 Reproducibility and Replicability. The first part studies the prevalence of reproducibility and replicability issues via a literature review.

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

Before investigating the reproducibility/replicability issues, we need to analyze how SE studies using DL models are reported in the SE and AI literature and what are the characteristics of these studies. To do this we performed a literature review on studies published in 20 top SE venues and 20 top AI venues in the last five years.

RQ2. How often do SE studies provide reproduction packages to support the reproducibility of their DL models?

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

RQ3. What RQs do SE studies investigate related to DL reproducibility/replicability?

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

Part II: Experiments on DL Reproducibility and Replicability. The second part of our study evaluates the importance of reproducibility/replicability in SE studies using DL models by a suite of experiments on four representative DL models. These are DeepCS [59], RRGGen [49], RLNN [122], and ASTNN [207]. These four models were selected because they cover four different SE tasks and are published in the last two years with accessible reproduction packages.

²<https://www.openml.org/>

RQ4. How does DL model stability affect reproducibility?

DL model stability is affected by the random initialization and the iterative optimization with randomly shuffled 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 reproducibility.

RQ5. How does DL model convergence affect replicability?

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 replicability, 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 replicability, because a good reported performance would be more difficult to be replicated.

RQ6. How does the out-of-vocabulary issue affect replicability?

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 replicability. To analyze whether the OOV issue is a high potential threat to replicability, 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 replicability?

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 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 replicability, 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 replicability 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?

To answer the research question, we searched SE studies that use DL techniques from 20 SE venues (Section 4.1.1) and 20 AI venues (Section 4.1.2). In total, we obtained 147 relevant studies. The following two subsections show our search method and analysis results in detail.

4.1.1 Review in SE Venues.

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

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 reproducibility/replicability issues.

Method. We identified a set of search terms in DL studies in SE that were already known to us. All the candidate search terms are related to DL techniques (e.g., "deep", "learning", "convolutional", "neural", "network", etc.). We refined the candidate search terms by checking the titles, abstracts, and keywords of 41 relevant papers. We selected terms that can match all the relevant papers, and form the search string with logical "OR": "*deep OR neural OR network*". We used the search string to perform an automated search on three digital library portals, limiting to the past five years (2015-2019), including IEEE Xplore³, ACM digital library⁴, and Web of Science⁵ for 20 widely read SE journals and conferences listed in Table 1. Once we retrieved candidate studies relevant to the DL studies, we performed a relevance assessment according to the following inclusion and exclusion criteria:

- ✓ *The paper must be a full research paper published in the selected twenty SE venues.*
- ✓ *The paper must propose a new SE model including the DL technique.*
- ✗ *Conference studies with extended journal versions are discarded.*
- ✗ *The studies (e.g., empirical/case/user study) that only re-ran existing DL-based models are excluded.*

³<https://ieeexplore.ieee.org>

⁴<https://dl.acm.org>

⁵<http://apps.webofknowledge.com>

Table 2. Selection of DL studies in SE venues.

Conference	ICSE	ICSEM	FSE	ASE	ISSTA	SANER	ESEM	ICPC	ISSRE	ICST	Total
Automated Search	50	26	22	37	17	24	6	9	8	2	201
Inclusion/Exclusion	14	9	6	16	2	12	0	4	1	2	66
Journal	TOSEM	TSE	ASEJ	ESE	IETS	IST	JSS	JSEP	STVR	SQJ	Total
Automated Search	3	36	13	57	39	78	210	39	12	16	503
Inclusion/Exclusion	2	11	1	1	1	4	6	0	1	0	27

Table 3. Number of papers published in the recent five years from SE venues.

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

The inclusion and exclusion criteria were piloted by the first and second authors starting with the assessment of 50 randomly selected primary studies. The agreement of authors' assessments was measured using pairwise inter-rater reliability with Cohen's Kappa statistic [36]. The agreement rate in the pilot study was "moderate" (0.71). The pilot study helped us develop a collective understanding of the inclusion/exclusion criteria and confirm the effectiveness of the criteria. Then, the assessment was performed for the full list of the identified studies with the same inclusion/exclusion criteria. The agreement rate in the full assessment was "substantial" (0.77). Disagreements were resolved after open discussions between the first and second authors. For the case that they did not reach a consensus, the third author was consulted as a tie-breaker. As shown in Table 2, we obtained 704 DL studies (201 conference papers and 503 journal papers) by the automated search. After checking the full text of these studies, we finally obtained 93 relevant DL studies.

Results. Table 3 shows that **77.4% of SE studies using DL** 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 3 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 5. Note that as the study [207] 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 5 we can observe that **63.8% of the DL studies focus on the study of code and defects**, followed by non-code software artifact, test, and Q&A. Furthermore, we notice that **45.7% of the proposed DL models work on eight SE tasks**, as illustrated in Fig. 6. Here code clone detection [51, 207] and defect prediction [126, 185] 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) [89], convolutional neural network (CNN) [195], deep feed-forward network

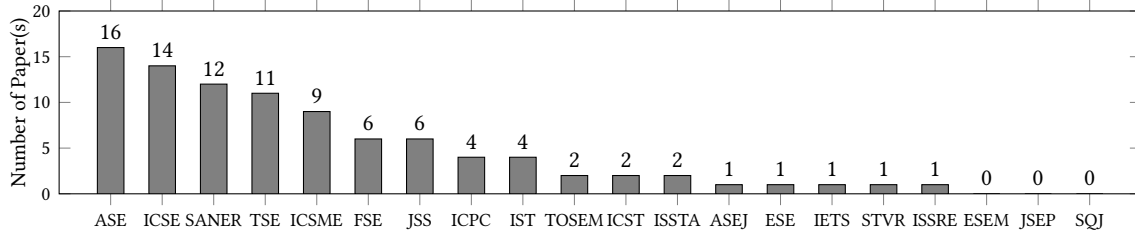


Fig. 1. Number of papers published in twenty SE venues.

Table 4. Number of papers addressing the semantics of code or text for different SE studies.

Study Subject	Description	#Paper
Code	The source code or binary code in software programming.	36
Defect	A condition that an implemented code does not meet its development requirements.	24
Non-Code Artifact	The programming related artifacts, such as comment, images, requirement, etc.	7
Test	An investigation on the quality of the software under test.	6
Q&A	The question and answer from a programming knowledge forum (e.g., Stack Overflow).	5
App	The software works for mobile phones.	5
Commit	The code change submitted to a version control system (e.g., GitHub).	4
Effort	The cost for software development in terms of human resources or time.	3
Bug Report	The condition that a software works in an unexpected way, which is commonly recorded in a bug tracking system (e.g., BugZilla).	2
Screencast	The programming video (e.g., from YouTube).	1
Energy	The energy consumption when running a software.	1
Total	-	94

(DFFN) [82], deep belief network (DBN) [186], and reinforcement learning (RL) [122]. 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 7 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 understanding semantics of code and text and their transformation (e.g., code Summarization [98, 182], code change generation [175], etc.) in SE tasks, as illustrated in Fig. 4. Furthermore, Table 7 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).

4.1.2 Review in AI Venues.

Motivation. As researchers also submit many SE studies to AI venues, we thus extended our survey on DL studies that solve SE tasks and that were published in top AI venues.

Table 5. Categories of ninety three DL studies in our literature review. There are ninety four studies in the table because the study [207] performs predictions on two SE tasks (code clone detection and code classification).

Study Subject	Tasks and References	#Task	#Study
Code	Code clone detection [22, 51, 61, 102, 142, 193, 199, 205, 207, 212] Code search [25, 59, 60, 181, 187], Code classification [44, 97, 207] code readability classification/prediction [133, 152], Function type inferring [72, 131] Code generation [50, 145], Code Summarization [98, 182], Code Decompilation [89, 93] Code change generation [175], Data structure classification [136], Reverse execution [140] Design pattern detection [172], Technical debt detection [153], Story points prediction [33] Inconsistent method name refactoring [118], Stable patch detection [74]	16	36
Defect	Defect prediction [121, 126, 174, 185, 186, 191], Vulnerability prediction [41, 53, 71, 203] Bug localization/detection [81, 94, 105, 188, 195, 210], Code repair [16, 177, 192] Code smell detection [115, 116], Anti-pattern identification [12] Code naturalness prediction [73], Static checker alarm classification [100]	8	24
Non-Code Artifact	Code comment generation [76, 215, 217], Software resources tracking [63] Software configuration prediction [68], Software incidents triage [30] requirement actor/action extraction [1]	5	7
Test	Test case generation [13, 27, 91, 119, 176, 214]	1	6
Q&A	Link classification [197], Question retrieval [29], API extraction [128] Tag recommendation [117], Intention classification [78]	5	5
App	GUI generation [28, 137], Communication identification [213] Review response classification/generation [49, 62]	3	5
Commit	Issue-commit link recovery [155, 196] Commit message generation [84], Pull request description generation [122]	3	4
Effort	Effort prediction [17, 92], Duration prediction [123]	2	3
Bug Report	Duplicate bug detection [43], Bug report summarization [104]	2	2
Screencast	Programming screencast action recognition [211]	1	1
Energy	Software energy consumption prediction [154]	1	1
Total	-	47	94

Table 6. The top-8 popular SE tasks in SE venues in terms of the number of published papers.

SE Task	Description	#Paper
Code Clone Detection	Detecting whether two code share the same requirement.	10
Defect Prediction	Predicting whether a code that contains defect, bug, or error.	6
Bug Localization/Detection	Locating where a defect/bug/error occurs in a software project.	6
Test Case Generation	Automatically generating test case for a specific code.	6
Code Search	Searching source code from a large-scale code corpus with a text-based query.	5
Vulnerability Prediction	Predicting whether a code that involves vulnerability issues.	4
Code Comment Generation	Automatically generating comment for a piece of code.	3
Code Repair	Automatically repairing a code with a specific fault/defect/error.	3
Total	-	43

Table 7. Statistics of basic model settings from ninety three DL studies in SE venues. 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%

Method. For AI venues, it is difficult to identify the search terms in the first place, because AI venues may cover a wide range of SE tasks. Therefore, we manually checked all the papers (29k in total) published in ten popular AI conferences (AAAI, NeurIPS, ACL, ICML, etc.) and ten top AI/ML journals (TKDE, ML, AI etc) as listed in Table 8 by using the DBLP⁶ database. The publication date was limited to the past five years (2015-2019). The first and second authors spent one week to independently filter the papers according to the following inclusion/exclusion criteria:

- ✓ *The paper must be a full research paper.*
- ✓ *The paper must propose a new model to solve a SE task.*
- ✓ *The proposed model must use the DL technique.*
- ✗ *Keynote records are excluded.*
- ✗ *The studies (e.g., empirical/case/user study) that only analyze existing DL models are excluded.*

As illustrated in Table 9, we obtained 71 studies working on SE tasks. After excluding the studies that did not use a DL technique, the reviewed ten AI conferences cover 54 DL studies for SE tasks. The agreement of authors' assessments was measured using pairwise inter-rater reliability with Cohen's Kappa statistic [36]. The agreement rate was "almost perfect" (0.96) because the total number of publications is large (around 29k), and it is easy to observe that all relevant papers show keywords related to SE tasks (e.g., program and bug) in their titles. Some disagreements were resolved after open discussion between the first and second authors.

Results. Table 9 shows that **all the 54 obtained SE studies using the DL technique** are published in seven AI conferences, including ICLR, NeurIPS, AAAI, ICML, ACL, IJCAI, and KDD. As illustrated in Table 10, researchers rarely published their SE DL studies in AI venues in 2015. However, **the publication number has been increased substantially in the following years** (2016-2019), where 87% of the studies were published in the recent three years. Moreover, Table 11 shows that these **54 studies focus on five study subjects** (i.e., code, defect, test, non-code artifact, and commit) and aim to solve 16 different SE tasks. From Table 12, we can observe that **87% of papers work on the four popular SE tasks**: code generation, bug localization/detection, code

⁶<https://dblp.uni-trier.de/>

Table 8. The selected AI conferences and journals for literature review.

Type	No.	Venue	Full Name
Conference	1	AAAI	AAAI Conference on Artificial Intelligence
	2	NeurIPS	Annual Conference on Neural Information Processing Systems
	3	ACL	Annual Meeting of the Association for Computational Linguistics
	4	ICML	International Conference on Machine Learning
	5	IJCAI	International Joint Conference on Artificial Intelligence
	6	ICLR	International Conference on Learning Representations
	7	KDD	International Conference on Knowledge Discovery and Data Mining
	8	SIGIR	International Conference on Research on Development in Information Retrieval
	9	ECAI	European Conference on Artificial Intelligence
	10	COLING	International Conference on Computational Linguistics
Journal	11	AI	Artificial Intelligence
	12	TPAMI	IEEE Transactions on Pattern Analysis and Machine Intelligence
	13	JMLR	Journal of Machine Learning Research
	14	TKDE	IEEE Transactions on Knowledge and Data Engineering
	15	TAP	ACM Transactions on Applied Perception
	16	TNNLS	IEEE Transactions on Neural Networks and learning systems
	17	JAIR	Journal of Artificial Intelligence Research
	18	ML	Machine Learning
	19	NC	Neural Computation
	20	NN	Neural Networks

Table 9. Selection of DL studies for SE tasks in AI venues.

Conference	AAAI	NeurIPS	ACL	ICML	IJCAI	ICLR	KDD	SIGIR	ECAI	COLING	Total
#Total Publications	4601	4093	1998	2426	3928	1152	1410	1208	1167	669	22652
#SE Tasks	11	14	9	8	9	15	2	0	0	0	68
#DL Models	9	10	6	7	6	15	1	0	0	0	54
Journal	AI	TPAMI	JMLR	TKDE	TAP	TNNLS	JAIR	ML	NC	NN	Total
#Total Publications	377	1026	790	1046	116	1620	325	386	495	854	7035
#SE Tasks	0	0	0	1	1	1	0	0	0	0	3
#DL Models	0	0	0	0	0	0	0	0	0	0	0

repair, and code comment generation. From Table 11, we can also observe that **seven studies solved different SE tasks**, comparing with the tasks listed in Table 5, including code decompilation [48], generating source code from binary code; code alignment [101], aligning a source code to its compiled binary code; code optimization [23], transforming a given code to a more efficient version while preserving its input-output behavior; code completion [170], completing code sentences according to the part that has been typed in according to the code context; program verification [166], proving that a program satisfies a development requirement; automatic code review [160], automatically determining a changed source code to be accepted or rejected for code review; and code feedback propagation [148], providing personalized feedback for massive code submitted by many students with the same development requirement based on existing code feedbacks. To solve the SE tasks, the reviewed 54 studies are based on four DL techniques including RNN, CNN, DFFN, and GNN (graph neural network) [157].

Table 10. Number of papers published in the recent five years from AI venues.

AI Venues	2019	2018	2017	2016	2015	Total
ICLR	6	6	3	0	0	15
NeurIPS	3	6	0	1	0	10
AAAI	5	1	2	1	0	9
ICML	1	3	1	1	1	7
ACL	2	0	3	1	0	6
IJCAI	1	1	2	2	0	6
KDD	1	0	0	0	0	1
Total	19	17	11	6	1	54

Table 11. Categories of 54 DL studies for SE tasks published in AI venues.

Study Subject	Tasks and References	#Task	#Study
Code	Code clone detection [190, 209], Code classification [15, 138] Code generation [11, 15, 18, 21, 24, 31, 32, 39, 64, 88, 109, 141, 144, 146, 147, 150, 151, 162–164, 168, 169, 178, 189, 204, 208, 218] Code decompilation [48], Code alignment [101] Code optimization [23], Code completion [170], Automatic code review [160]	8	36
Defect	Bug localization/detection [4, 66, 79, 80] Code repair [65, 67, 179, 184], Vulnerability detection [96]	3	9
Test	Test case generation [120], Program verification [166]	2	2
Non-Code Artifact	Code comment generation [5, 6, 77, 106], Code feedback propagation [148]	2	5
Commit	Commit message generation [125, 198]	1	2
Total	-	16	54

Table 12. The top-7 popular SE tasks in AI venues in terms of the number of published papers.

SE Task	Description	#Paper
Code Generation	Automatically generating code for a specific requirement.	27
Bug Localization/Detection	Locating where a defect/bug/error occurs in a software project.	4
Code Repair	Automatically repairing a code with a specific fault/defect/error.	4
Code Comment Generation	Automatically generating comment for a specific code.	4
Code Clone Detection	Detecting whether two code share the same requirement.	2
Code Classification	Predicting the classification of target code.	2
Commit Message Generation	Automatically generating message for a committed code in GitHub.	2
Total	-	45

Table 13 shows that **59% of studies utilized the RNN, and 9% of studies solved SE tasks by combining multiple DL techniques.**

Implications. In the last three years, researchers frequently published their studies in AI conferences (e.g., ICLR, NeurIPS, and AAAI). The reviewed 54 studies mainly focused on code related SE tasks (e.g., code generation,

Table 13. Statistics of basic model settings from ninety three DL studies in AI venues. The basic DL models include recurrent neural network (RNN), convolutional neural network (CNN), deep feed-forward network (DFFN), and graph neural network (GNN).

Configuration	Model Setting	#Study	%Study
The study with individual DL model	RNN	32	59.3%
	CNN	5	09.3%
	DFFN	8	14.8%
	GNN	4	07.4%
The study with combined DL models	RNN + CNN	3	05.6%
	RNN + GNN	1	01.9%
	RNN + CNN + DFFN	1	01.9%

code repair, and code comment generation) by using RNN-based models. These studies also cover seven new SE tasks, such as code completion and program verification, comparing with the ones published in SE venues.

4.2 RQ2. How Often do SE Studies Provide reproduction packages to Support the Reproducibility for DL Models?

Motivation. reproduction package is a fundamental requirement for model reproducibility. DL reproducibility can however be compromised when developers provide no reproduction package, or the quality of the supplied reproduction package is low (e.g., no data or incomplete source code). Sharing high-quality reproduction packages is a key to better support DL model reproducibility as it allows other researchers or practitioners to much more easily repeat results reported in a paper. Therefore, to analyze whether reproducibility is a supported issue for DL studies in SE, we investigate the percentage of the reviewed DL studies that share reproduction packages and also check their quality.

Method. To investigate the prevalence of support for reproducibility issues in SE studies, we inspected the reviewed 147 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 reproduction packages. If no link was found in a paper or a link is not accessible, we searched the reproduction package from GitHub by using the paper title as the search term. Whether the reproduction packages belonged to the study authors can be confirmed by checking the owner's profile (e.g., name, university, and email) or the citation information presented in the README file. Moreover, for a paper with no accessible reproduction packages found in paper or GitHub, we sent an email to the first author to request their reproduction package. If another corresponding author (i.e., not the first author) was marked in the paper, the email was also CCed to the corresponding author. To assess the quality of the obtained reproduction packages, we first check whether they reflect the model described in the paper. The DL reproducibility can be better supported by a high-quality reproduction package.

Results. Table 14 shows the statistics of the reviewed DL studies that share links to their reproduction packages. We notice that **only 38.1% of these SE studies using DL** provided a link to publicly shared reproduction packages. By checking the links with a browser, we found that 4 links are broken. Therefore, only 35.4% of the 147 DL studies provide *accessible* links to their reproduction packages. How long these remaining reproduction package links will remain available is unknown. From Table 14, we can notice that 59.3% of 54 studies from AI venues provide accessible reproduction packages in the first place, while only 33.3% of 93 studies from SE venues have done the same. After searching the paper titles in GitHub, we found ten more studies that shared

reproduction packages after the publication of their papers. In this way, we found accessible reproduction packages for 63 studies.

Moreover, we sent emails to request reproduction packages for the other 85 papers. **28 authors replied to our emails but only 15 emails provide accessible reproduction packages.** The other 13 emails told reasons why they provide no reproduction packages, as illustrated in Table 15. One major reason is that five studies collaborated with enterprises (e.g., Google and Microsoft) and their source code and data cannot be shared in public due to nondisclosure agreements. Seven authors told us that they provided no reproduction packages for now. Additionally, one author explained that the reproduction package is not a requirement of their published venue. From the statistics in Table 15, we can observe that only 17.6% of emails were replied with that resulted in us obtaining reproduction packages. This result indicates that sending emails is not a reliable way to request reproduction packages.

In total we were able to obtain 78 accessible reproduction packages. We have provided links to these for the corresponding papers in the references. To evaluate the quality of these reproduction packages, we downloaded them and checked their integrity at first. As shown in Table 15, we found that **29.5% of them did not provide complete source code or data.** Specifically, as shown in Table 16, ten reproduction packages did not include data or provide data links in the README file or paper. One reproduction package only presented testing data without training data. 12 reproduction packages suffered from the issue of code integrity. We found that the source code does not exist in 6 reproduction packages; 3 reproduction packages only provided either training code or testing code; 3 reproduction packages just included the core model component, and other components are required to be re-implemented and combined to the supplied core component. Due to the low rates of email replies, there would be less chance to request the full data and code after the publication. Therefore, it is also suggested to check the integrity of the reproduction package in the first place.

For the remaining 55 reproduction packages, we spent two weeks to check whether their source code follows the models described in their papers, including model structure and settings. Results show that all of them can reflect the development requirement described in papers. Therefore, among the 147 reviewed DL studies, **only 37.4% of them provide high-quality reproduction packages to support model reproducibility.** We also noticed that 50% of the studies in AI venues provide accessible and high-quality reproduction packages, while the number is only 30% for the SE venues. Therefore, the reproducibility of DL models in the SE community needs to be further improved.

Additionally, as shown in Table 14 we found that, among the 147 reviewed DL studies, 56 papers provided links to their reproduction packages but only 52 links were accessible at the time we checked them. This result indicates a sign of the aging process of the liveness of the links. Afterward, we obtained 26 more accessible links by sending request emails for the authors in December, 2020, where we obtained no update for the previously identified 4 inaccessible links. To investigate whether the aging process continues, we checked these 78 links six months later. We found that **one link was not accessible any more even if it has a large number of citations.** Meanwhile, for the 4 links that were still inaccessible after sending emails, the authors did not make any updates during the six months. Therefore, DL studies are required to provide long-lasting reproduction packages in consideration of the aging process of their links.

Implications. Most of our reviewed DL studies for SE tasks do not provide high-quality reproduction packages that are publicly accessible (with the link given in the paper), leaving prevalent threats to low reproducibility. Under this circumstance, other researchers or practitioners need to replicate existing studies involving DL models from scratch and may miss important implementation details. We found that sending emails is not a reliable way to request a reproduction package. We suggest that all further studies should provide accessible and long-lasting links to their reproduction packages (with complete source code and data) in the papers.

Table 14. Collecting reproduction packages and checking their quality.

Process	Item	SE Venue	AI Venue	Total
0. Review papers	Number of papers	93	54	147
1. Search reproduction package from paper and GitHub	Links in papers	27	29	56
	Accessible links in papers	24	28	52
	Reproduction packages in GitHub	6	4	10
	Total accessible reproduction packages	31	32	63
2. Request reproduction package from authors	Sending emails	63	22	85
	Replied emails	24	4	28
	Accessible reproduction packages	12	3	15
3. Check quality of replication packages	Total accessible reproduction packages	43	35	78
	Reproduction packages with data issue	6	5	11
	Reproduction packages with code issue	9	3	12
	High-quality reproduction packages	28	27	55

Table 15. The reasons why studies did not provide reproduction packages.

Reason	Description	SE Venue	AI Venue	Total
Closed projects	The source code and data are belonged to enterprises (Google, Microsoft, Facebook, Samsung, and an unknown company) and not allowed to share in public.	4	1	5
Not prepared	No reproduction package for now.	7	0	7
Not required	Not required for the published venue.	0	1	1
Total	-	11	2	13

Table 16. Statistics of the integrity issue for reproduction packages.

Type	Issue	SE Venue	AI Venue	Total
Data issue	No training and testing data	5	5	10
	No training data	1	0	1
Code issue	No (main) code	4	2	6
	No training code	2	0	2
	No testing code	0	1	1
	Model components	3	0	3
Total	-	15	8	23

4.3 RQ3. What RQs do SE studies investigate related to DL reproducibility/replicability?

Motivation. According to our observation, some internal factors within DL models may affect study reproducibility and replicability, although sharing reproduction packages can help support DL reproducibility. For example, a model with unstable performance will lead to low reproducibility. Model stability can be influenced by randomly initialized network weights for the DL model. Although some reproduction 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, other researchers can hardly obtain the reported experimental results on a different testing data no matter how correct they re-implement the DL models. In other words, the DL-based SE study replicability would be unsatisfactory. Therefore, discussing these kinds of internal factors in DL models can further strengthen the DL reproducibility and replicability. We analyzed and classified the RQs related to reproducibility or replicability from the reviewed DL studies.

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 reproducibility/replicability, we inspected what the RQs in each paper aim to investigate. We then calculated the percentage of reviewed studies in Section 4.1 investigating DL reproducibility or replicability. If DL studies in SE rarely perform experiments on such internal factors, the DL reproducibility/replicability issue would be still prevalent even if some DL studies may provide reproduction packages.

Results. As shown in Table 17, 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 reproducibility/replicability – the degree to which that the reported model performance can be replicated/reproduced considering some model/experimental settings. Table 17 shows that **all the 147 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 27.2% of the studies** provided runtime information that informs the efficiency of their proposed approach for training and prediction. As to the model reproducibility/replicability, **only 10.2% of the total studies** investigated RQs related to these factors. Besides, we can notice that the studies published in SE venues show more effects in RQs on efficiency and reproducibility/replicability, comparing with the studies obtained in AI venues. This result implies that the SE community cares more factors on the practical usage of the proposed models.

To understand the characteristics of RQs on DL reproducibility/replicability, we classify them into four RQ types as described in Table 18. These RQ types include model randomness [17, 71, 174], whether the randomness in DL model (e.g., random optimization process) affects model reproducibility; model convergence [89, 105], whether the model optimization is divergent so that the turbulent performance leads to low reproducibility; out-of-vocabulary (OOV) issue [5, 13, 71, 73, 122, 147, 197], whether model performance is sensitive to the size of vocabulary for encoding words into vectors, resulting in low replicability; sensitivity to testing data size [79, 141, 142], whether the model performance is sensitive to the size of testing data (i.e., low replicability). We can notice that when analyzing the sensitivity to testing data size, Nafi et al. [142] tested the model performance on a different dataset. However, this experimental setup may not enough to investigate reproducibility. To strengthen the experimental setup, Huo et al. [79] tested the model on four individual datasets and their combination. In contrast, as Murali et al. [141] only collected one testing data, to facilitate the experiment they ran a model on subsets of testing data. A model with higher replicability would obtain similar results on the whole and subsets of testing data.

Besides, Table 18 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 issue is low** (7 papers, 4.8% of total studies), since 85% of the reviewed 147 DL studies need to encode code/text

Table 17. Statistics of studies obtained from SE and AI venues that discuss RQs related to model effectiveness, efficiency, or replicability/reproducibility.

Type of RQ	Description	SE Venue	AI Venue	Total
Effectiveness	Whether the proposed new DL model outperforms some baselines in terms of some evaluation metrics.	93	54	147
Efficiency	Whether the proposed new DL model works faster than some baselines during model training and testing respectively.	38	2	40
Reproducibility or Replicability	The degree to which the reported model performance can be replicated/reproduced considering some model settings or experimental settings.	11	4	15

Table 18. Classification of RQs related to reproducibility/replicability issues among the studies obtained from SE and AI venues.

RQ Type	Description	SE Venue	AI Venue	Total
Model randomness	Whether the randomness in model training (e.g., parameter initialization and random optimization process) affects DL reproducibility.	3	0	3
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	0	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 hard to be replicated.	5	2	7
Sensitivity to testing data size	Whether the model performance is sensitive to the size of testing data, resulting in low replicability.	1	2	3
Total	-	11	4	14

by using a fixed size of vocabulary (as illustrated in Fig. 4). The number of papers related to the other three RQ types is also very low. Although some studies provided reproduction packages to support the reproducibility, they missed the replicability issues and left solving them to future work [16, 59, 98, 175]. One major obstacle to addressing DL reproducibility/replicability 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. As most DL studies have a lack of replication materials (e.g., the random seeds for model initialization and optimization) and reporting with insufficient experiments, their reproducibility could be strongly affected by the model randomness while the replicability might be highly influenced by some internal factors (e.g., model convergence, OOV issue, and sensitivity to testing data size in DL models). However, these factors are widely overlooked in DL-based SE studies. Therefore, there is a need for more studies on the impact of reproducibility and replicability 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.

Table 19. 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 [59]
Review response generation	Generating high-quality response to a user's review of an App.	RRGen [49]
Pull request description generation	Generating high-quality description for an empty pull request.	RLNN [122]
Code classification	Classifying code fragments according to their functionality.	ASTNN [207]

5 DEEP LEARNING MODELS

According to the literature review in Section 4, we know that the DL reproducibility/replicability issues are commonly overlooked and often regarded as minor threats in the SE domain. To analyze the importance of DL reproducibility/replicability, we designed four controlled 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 [59], review response generation [49], pull request description generation [122], and code classification [207]. These DL models are selected as study subjects because they are published in top SE venues in the last two years with accessible and high-quality reproduction packages. Table 19 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 [59] 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) [134] and one multilayer perceptron (MLP) [34] to embeds code method (c) and query (q) into vectors respectively:

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

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 be 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):

$$L(\theta) = \sum_{\langle c, q^+, q^- \rangle \in P} \max(0, 0.05 - \cos(v_c, v_q^+) + \cos(v_c, v_q^-)), \quad (2)$$

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 RRGGen

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 [49]. To automatically generate high-quality responses to user reviews, an RNN-based model called RRGGen (review response generation) was developed to learn the relationship between review-response pairs [49].

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

$$y = RNN(x_t, MLP(x_c, x_l, x_r, x_s)) \quad (3)$$

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, RRGGen constructs a vocabulary with top-10k frequently occurred words in training data. Moreover, to optimize the randomly initialized parameters (θ) in RRGGen, the model utilized a loss function (L):

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

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 [122]. To help developers generate high-quality PR descriptions automatically, a reinforcement learning (RL) based RNN model was proposed [122]. 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}):

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

where w the RNN parameters; $\hat{y}_0, \dots, \hat{y}_{j-1}$ 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):

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

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:

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

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, \dots, 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 [207], 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:

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

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 its ground-truth category (\hat{x}), which is optimized by the following loss function:

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

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 20 provides their important settings related to the last four RQs in Section 3. We run the public reproduction 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 [59], 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

Table 20. 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

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' reproduction package⁷ 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 [49]. 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 [49]. 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 [10]. 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 correlated with human judgments than other settings [114]. We re-ran RRGen by using the reproduction package⁸ 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 reproducibility 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 [107]. Specifically, the F1-score of ROUGE-L is adopted in the model evaluation and its detailed definition can be found in the RLNN study [122]. We re-ran their public reproduction package⁹ shared on GitHub.

ASTNN. For code classification, the validity of ASTNN is verified by a public dataset built by Mou et al. [139]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' reproduction package¹¹ 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<https://github.com/guxd/deep-code-search>

⁸<https://github.com/armor-ai/RRGen>

⁹<https://github.com/Tbabm/PRSummarizer>

¹⁰<https://sites.google.com/site/treebasedcnn/>

¹¹<https://s3.us-east-2.amazonaws.com/icse2018/index.html>

7 EXPERIMENTAL RESULTS

We present our experimental results to help answer the last four RQs described in Section 3, investigating the importance of DL reproducibility and replicability. 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 reproducibility and replicability, and whether this influence is strong enough to threaten the validity of the studies.

7.1 Examining Reproducibility and Replicability of DL Experiments under Original Settings.

7.1.1 RQ4. How does Model Stability Affect Reproducibility?

Motivation. DL reproducibility can be largely supported by sharing a reproduction package. Ideally, re-running the source code and data provided can replicate the reported model performance. However, a publicly shared reproduction 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 [103]. Therefore, DL model performance could be varied for different researchers, so that a model with unstable performance would be of low reproducibility if the reproduction package does not provide the random seed for model initialization and record the lists of randomly shuffled data batches in model optimization. This is the reason why the DL reproducibility 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 reproducibility.

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 reproducibility, we utilize the mean value of ten experimental results as the performance that the DL model likely produces with high 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 reproducibility.

Results. Fig. 2 illustrates the ten experimental results of four DL models (i.e., DeepCS, RGen, 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. 2 are listed in Table 21. 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 RGen, 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 21 shows that the c_v values of four models are 5.92%, 6.27%, 0.67%, and 0.1% respectively. Thus, **the DeepCS and RGen 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 reproducibility. 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 reproducibility may strongly affect the overall validity of the study**. For example, as shown in Table 21 the model DeepCS obtains a mean performance of 0.535. However, if a baseline model achieves a performance of 0.54 (outperforming the mean value of DeepCS by 1%), the effectiveness of DeepCS over this baseline would be biased when just reporting the

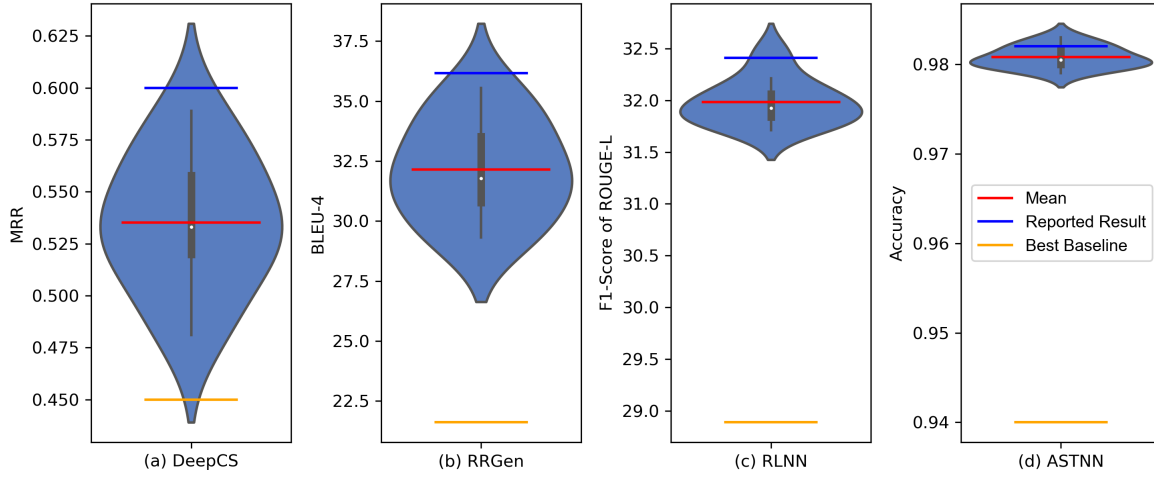


Fig. 2. Violin plots of four DL models with ten repeated experimental results, where white dot and red line indicate median and mean values respectively; blue and orange lines denote the performance of four models and the best baseline reported in the original studies, respectively.

Table 21. 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%

maximum value of 0.589 with high difficulty for reproduction. In this case, the model validity is compromised due to the low reproducibility caused by the high model randomness and poor reporting. Besides, although providing random seeds used in model training are helpful to solve the issue of low reproducibility, such seeds just cover up the real reason why the proposed model is more effective than the baseline, i.e., due to the randomness not any improvement of the model. Hence, such high reproducibility with random seed is meaningless. Thus, we should also prevent someone from purposely leveraging the trick of the "meaningless high reproducibility" to prove the validity of the proposed model (i.e., running the model multiple times, reporting the best result, and providing the random seeds).

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.**

Table 22. Statistics of different stop conditions for 147 reviewed DL studies.

Specific Stop Condition	SE Venue	AI Venue	Total	Percent
Converge	15	5	20	13.6%
Fixed epochs	31	6	37	25.2%
Maximum epochs or converge	34	23	57	38.8%
Best model on validation data	6	2	8	5.4%
Total	93	54	147	100%

Table 23. Experiment setup for four DL models.

Model	Specific Stop Condition
DeepCS	Fixed epochs
RRGen	Fixed epochs
RLNN	Best model on validation data
ASTNN	Fixed epochs

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 reproducibility will produce greater negative effects on an unstable model. Therefore, it is important to estimate the DL stability with sufficient repetitions of experiments in terms of the mean and standard errors, instead of only reporting one good experimental result. The reported mean performance is easier to be reproduced, which would also mitigate the threats to model validity.

7.1.2 RQ5. How does Model Convergence Affect Replicability?

Motivation. In model training, we observed that DL studies stopped the optimization mainly in four ways, as illustrated in Table 22. From the table, we can notice that 52.4% of models stop optimization when the training converges or the total number of iterations reaches the pre-defined maximum value; 25.2% of models were trained with an empirically set number of epochs; 5.4% of models were selected based on the performance on validation data. Besides, Table 23 shows how the studied four DL models chose the corresponding strategy to stop their optimization. However, due to the model randomness, a re-implemented model may not achieve the reported performance. In this case, researchers have to train a model with more iterations. Ideally, if the optimization process of a model tends to be convergent, more iterations would mitigate the effect of model randomness and obtain the reported performance. Thus, the goal is to estimate the DL model convergence and investigate whether the convergence level is an influential factor for DL study replicability.

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 7.1.1, 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 [143]. The overfitting issue is one major reason [149].

Results. Table 24 shows our experimental results from the four DL models with extended optimization iterations. From the Table 24, we can notice that, after the extended training, the mean performance of DeepCS, RRGGen, 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 replicability issue of the first two models, DeepCS and RRGGen, show much higher negative effects over the other two.

We can notice that, comparing with the values in Table 21, Table 24 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.

Table 24. 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.

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

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. 3 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 25, 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%, 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 replicability issue.

As extending model training with more iterations cannot guarantee to obtain the reported model performance, Table 22 shows that 5.4% of DL studies (including RLNN) chose the best model based on the performance of the trained model tested on the validation data. To evaluate how this model training strategy affects DL-based SE study replicability, we ran DL models (RRGGen, RLNN, and ASTNN) on their validation and testing data with different training epochs or iterations. In this experiment, we excluded the model DeepCS because it provides no validation data. Fig. 4(a) shows that training RRGGen with more iterations can achieve even better performance

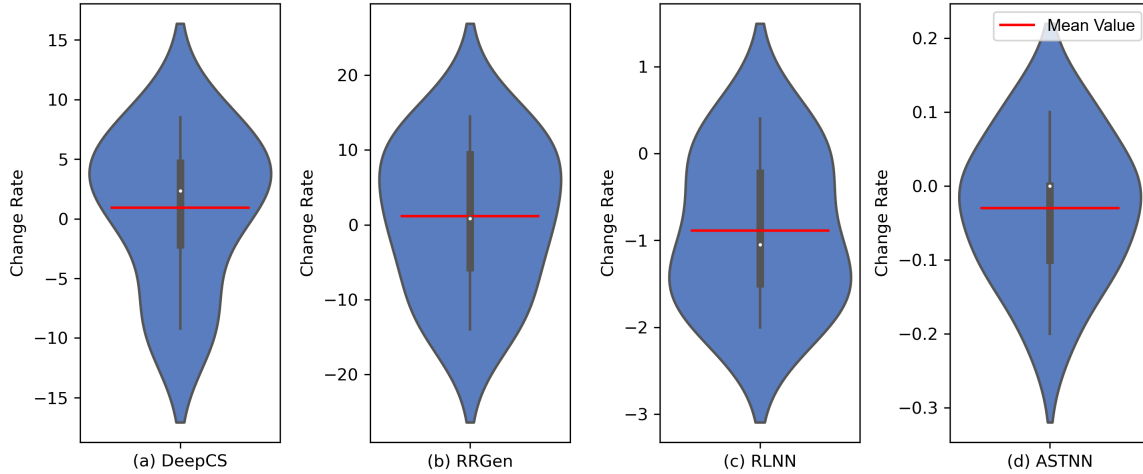


Fig. 3. Violin plots of the performance change rates after training models with extended iterations/epochs, where the red horizontal line indicates the average change rate.

Table 25. 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 minimum, 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%

(BLEU = 0.355) over the reported one (0.324). Moreover, linear regression analysis in Fig. 4(b) shows that the performance on validation data is proportional to the one on testing data, where slope equals 1.01 and the coefficient of determination R^2 is 1. This result implies that a better model on validation data would also achieve good performance on its testing data. For the RLNN and ASTNN, Figs. 4(c-f) show similar results although the R^2 of RLNN is smaller with a value of 0.66. These experimental results imply that compared with the training with an empirically set number of epochs, **choosing the best model on validation data is a better way to improve the DL replicability and help the re-implemented model reach the reported performance on a new experiment setting.** However, we need to note that choosing the best model on validation data is not the optimal solution as it may also introduce bias. For example, one may keep finding the best model on validation data with more iterations until the model achieves a satisfactory performance on testing data. Therefore, DL studies need to focus on how to improve model convergence.

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 replicability. To enhance confidence in any SE study with reported good effectiveness, it is necessary to verify the convergence level of the used DL model and improve the

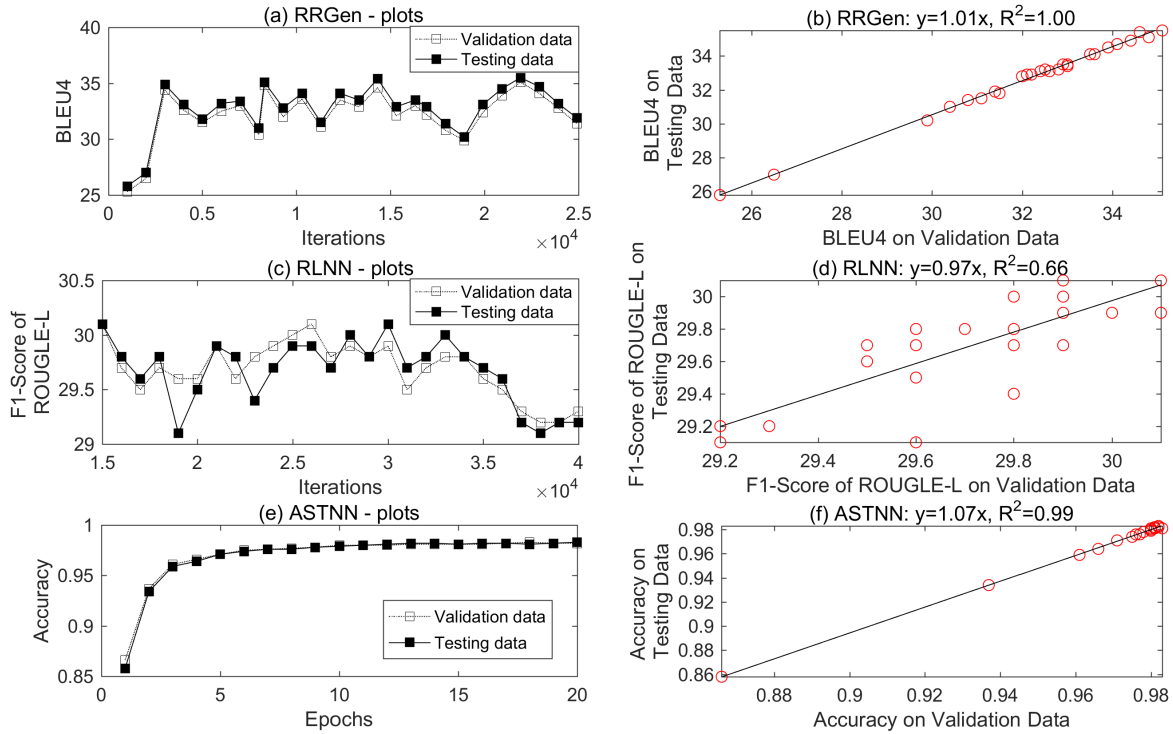


Fig. 4. Plots and regression analysis for three models (RRGen, RLNN, and ASTNN) working on validation and testing data.

DL replicability using appropriate training strategy. Although choosing the best model on validation data can achieve better replicability than stopping optimization with manually set iterations or epochs, it may introduce bias threatening the validity of the study. To avoid this issue, DL studies need to estimate and improve the model convergence, also with better experimental settings (e.g., using the cross-validation to suppress the bias).

For a paper without reproduction package, it would be difficult to analyze whether the re-implementation is correct. This is because the reported experimental result is unlikely to be precisely obtained again due to randomness in the DL models. In this case, it is unclear whether the bias from the reported result comes from the model randomness or from re-implementation errors. Therefore, we suggest DL studies providing high-quality reproduction packages in the first place.

7.2 Examining the Replicability of DL Experiments under Altered Settings

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

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. When replicating existing DL studies, the problem is that most of them provided no data (as shown in Table 14). Therefore, other researchers would have to collect their training and testing data by themselves. As the new dataset is not necessarily the same as the one collected by the original study, the degree of OOV issue (i.e., the

proportion of vocabulary built from training data covers the words in testing data) would also be different. Under this situation, the reported model performance could be difficult to be obtained, leading to low replicability.

Method. To simulate the replication studies with different degrees of OOV issue, we trained four DL models (DeepCS, RRGGen, RLNN, and ASTNN) with different sizes of vocabulary and performed predictions on their corresponding testing data. If the replicability is high, it is expected that the model performance shows little variation as the vocabulary size increases. Otherwise, one good reported model performance may not be replicated in new experiments. This is because the DL model is likely to misunderstand the semantics of code/text with new words, resulting in substantially poorer performance. Note that this RQ purposefully altered the experimental settings of the original studies.

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 of 10%. The full scale (100%) indicates that we used the vocabulary with the default setting in the original studies. Table 20 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 7.1.1.

Results. Table 26 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 26 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 RRGGen model, when increasing the vocabulary scale from 10% to 20%, the model performance reduces by 1.2%. However, the RRGGen 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%. These results indicate that **we can obtain close results as the reported ones for RRGGen, RLNN, and ASTNN only using the top 20% frequently used words in the vocabulary.** But for DeepCS, its reported performance could be difficult to be obtained with decreased scales of vocabulary.

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 [38] at a 5% significance level. Table 28 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 [194] at a 5% significance level.

The statistical results in Table 28 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

Table 26. 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 27. Comparison of DL models (RRGen and RLNN) with or without the byte-pair encoding (BPE) technique.



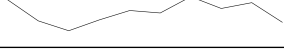
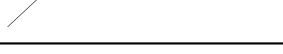
Model	BLEU4	Model	F1-Score of ROUGLE-L
RRGen	36.2	RLNN	32.4
RRGen+BPE	33.8	RLNN+BPE	21.7

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 replicability.**

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 replicability 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 [159] to replace the OOV words with the ones in training data in an appropriate way.

Except for the above OOV addressing techniques, one other critical solution is to tokenize words using the Byte-Pair Encoding (BPE) technique [161]. Generally, the BPE provides a text compression scheme that can better unify the words used in training and testing data. To support the feature of other non-English (e.g., Chinese

Table 28. 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%

and Japanese), Wang et al. [183] extended BPE with the byte-level BPE, short for BBPE. To investigate how the tokenization strategy affects the OOV issue, we incorporated the BPE into RRGen and RLNN. Note that DeepCS and ASTNN were excluded because they presented no raw data (i.e., the string of source code) to let BPE work properly.

Table 28 shows that by adding BPE to RRGen the performance drops by 6.6% from 36.2 to 33.8 in terms of BLEU4. Meanwhile, using the BPE technique, the performance of RLNN also decreased by 33% from 32.4 to 21.7 in terms of the F1-score of ROUGLE-L. These experimental results imply that RRGen and RLNN cannot well capture the semantics of the words compressed by using BPE. Therefore, although the BPE is a prevalent solution to address the OOV issue in the NLP domain, **directly applying the BPE technique to a SE task may not improve the model performance**. But it is still worthy of exploring how to leverage the advantages of BPE to overcome the OOV issue for further SE studies.

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 replicability. Therefore, DL-based studies should address the OOV issue and mitigate its negative effect as much as possible.

7.2.2 RQ7. How does Testing Data Size Affect Replicability?

Motivation. Sections 7.1.1-7.2.1 show that even if a replication study uses the same training data, the reported model performance may not be obtained due to model randomness, optimization convergence, and the OOV issue. A further question is how the testing data affects the DL-based SE study replicability. This question is important because replication studies commonly collect a subset of data from the real-world environment (e.g., GitHub and Stack Overflow) with different sizes. Also, 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 replicability.

Method. To simulate the replication studies with different sizes of testing data, 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, RRGGen, RLNN, and ASTNN, they contain around 16m, 14k, 4k, and 10k testing data in default as shown in Table 20. We sampled a subset of testing data at ten different scales, ranging from 10% to 100% with a step of 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 replicability 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. Note that as this research question focuses on the simulation of replication studies with different sizes of testing data and fixed size of training data, we perform the test data sampling instead of the cross-fold validation. Note that this RQ purposefully altered the experimental settings of the original studies.

Results. Table 29 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 of 10%. Table 29 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 RRGGen 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 29 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 7.2.1, we performed a Cox Stuart trend test [38] 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.



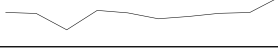
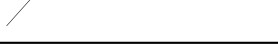
The trendlines and statistical results in Table 30 indicate that **all four models (DeepCS, RRGGen, RLNN, and ASTNN) show neither increasing nor decreasing trend with statistical significance** (p-values > 0.05). However, their performance variation does differ. Specifically, the c_v of DeepCS is 4% 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. RRGGen shows smaller performance variation with $c_v = 3.5%$ with no significant change at any scale. Similar to DeepCS, RRGGen 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 replicability of studies using a DeepCS or RRGGen 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 RRGGen, **its performance is not as sensitive to the testing data size as RRGGen**, as RRGGen 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 replicability 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

Table 29. 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 30. 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%

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 on 20 SE venues and 20 AI venues. 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 and AI venues. When selecting relevant DL studies, we excluded the studies that did not propose new DL models but perform systematic reviews in our search scope. This is because we found that the only related study [3] just summarized the findings from existing studies on software effort prediction (e.g., the artificial neural network is widely used

and outperforms other ML models), instead of making either reproduction or replication. However, further studies may need to consider these if they are reproduction or replication studies. To analyze the characteristics of the reviewed DL studies, we manually classified them into 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. Moreover, although we provided all the links to the studies with accessible reproduction packages in the reference, we cannot ensure whether their liveness would drop in the near future.

Limited Experiments. In the experiments, we selected only four DL models for use in our experiments. Running more DL models would help researchers better understand the DL reproducibility/replicability 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 reproducibility/replicability issues. Due to the same reason, we ran each DL model ten times for each research question.

To estimate the importance of the DL reproducibility/replicability, 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 reproducibility/replicability – model stability, convergence, OOV issue, and the testing data size. This investigation on DL reproducibility/replicability 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 reproducibility/replicability issue. For example, the OOV issue could be regarded a sampling issue. If a DL model trained with representative samples from the population, its experimental results could be easily replicated.

In addition to our investigated replicability issues, there are many other interacting factors that could also lead to problems of replicating the published results, e.g., differences in data cleaning, the splitting of training and testing data, and the choice of hyper-parameters vs. taking the defaults, etc. Overall one would expect that the more complex a DL model and the more complex the experimental design, then the harder it will be to generate the same results without a reproduction package. In the future, we plan to investigate other influential factors relating to DL reproducibility/replicability. We also suggest the research community investigating more replicability issues in the further DL studies.

Besides, when investigating the replicability issues, we did not re-implement the DL models by ourselves, but used the high-quality reproduction packages provided by the original studies as the correctly re-implemented models. Although this controlled experimental setting is helpful for analyzing whether the factors (e.g., the OOV issue and the testing data scale) could strongly affect the replicability, it excludes the analysis of the difficulties in model re-implementation. We would like to investigate the re-implementation issues in the future.

Manual Evaluation. When we re-ran the code search model DeepCS [59], 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 reproducibility and replicability issues are prevalent and important for DL-based studies in SE. To mitigate these reproducibility and replicability issues, DL studies in SE should consider the following guidelines:

Provide a long-lasting link for a high-quality reproduction package. An accessible reproduction package with complete 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 reproducibility 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. To improve the DL reproducibility, we also suggest that DL studies provide the random seeds used for model initialization (i.e., the seed to initialize network weights) and optimization (i.e., the seeds for a list of randomly shuffled data batches) in the reproduction package.

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. For the unconvergent model trained with a fixed number of optimization iterations, choosing the best model based on the performance on validation data is a viable way to improve the DL replicability.

Use an automated evaluation approach to avoid the effect of human bias and subjectivity. The reproducibility 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 [122], transform new words [69], or compress words using the Byte-Pair Encoding (BPE) techniques [161, 183].

Measure DL-based study replicability on different training and testing data. Study replicability requires that an experimental finding can be reproduced using different sampled training and testing data. Thus, one reported result on only one testing dataset will be not enough. It is suggested to train/test the model using different sizes of training/testing data and analyze the model performance (e.g., using the cross-validation).

Providing the trained model and applying the parallelization solutions. Time-consuming model optimization is a large obstacle when DL-based studies investigate model reproducibility. Therefore, it is recommended that authors try where possible to provide their trained model to mitigate the reproduction efforts. But for the model replicability, researchers have to re-implement and train the model by themselves. Thus, we recommend further DL-based studies trying the parallel and distributed solutions for DL [14] to accelerate the model training.

10 CONCLUSION

Reproducibility and replicability are important for scientific research but DL-based studies in SE often ignore or minimize them. To investigate the merit of SE study reproducibility/replicability 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 40 prevalent SE/AI venues in the recent five years. The observation from 147 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 beginning to show its potential to further improve the performance of existing DL-based models.

To assess the prevalence of reproducibility/replicability issues in SE studies using DL, we checked all the links in the reviewed DL-based studies and found that only 35.4% of studies provided accessible links to their reproduction packages. Thus, the DL reproducibility cannot be assessed without the authors' source code and dataset. Also, the DL replicability cannot be ensured because other researchers need to replicate existing studies from scratch and may miss important implementation details. Although 28 more reproduction packages can be obtained by searching paper title from GitHub and sending request emails from authors, this is not a reliable way. Among the

78 total obtained reproduction packages, only 71% of them provided complete source code and dataset. We also counted the percentage of DL studies that investigated any research questions on the reproducibility/replicability issues, and found that only 10.2% of studies considered these issues. Therefore, the reproducibility/replicability issue is prevalent, and it is suggested for further DL-based studies in SE to provide the links to their complete source code and data in the paper, and better analyze their DL-based study reproducibility/replicability issues to better support their DL model validity.

To investigate the importance of reproducibility/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 reproduction packages. Our experiments considered how the model stability affects DL reproducibility, and analyzed how model convergence, the out-of-vocabulary issue, and testing data size all impact DL-based SE study replicability. Our experimental results show that DL models possess randomness by nature so that study reproducibility can be compromised for an unstable model. A DL model with a low convergence level can produce highly turbulent and unstable performance, leading to low replicability. 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 reproducibility/replicability. We also need to investigate viable solutions to strengthen the study DL-based model validity instead of just presenting the advantages of chosen DL model effectiveness. Additionally, the investigated reproducibility/replicability issues in this study may be applicable for other ML or information retrieval (IR) models used in software engineering. In the near future, we plan to compare ML/IR models with DL models and analyze their differences. The reproduction package for our literature review and experiments can be found at: <https://bitbucket.org/ChaoLiuCQ/dlrep>.

ACKNOWLEDGEMENTS

This research was partially supported by the National Science Foundation of China (No. U20A20173), Key Research and Development Program of Zhejiang Province (No.2021C01014), the National Research Foundation, Singapore under its Industry Alignment Fund – Prepositioning (IAF-PP) Funding Initiative, and the ARC Laureate Fellowship (FL190100035). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore, Australia, and Huawei, China. This research is a non-Huawei service achievement.

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. *IST 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 *ICISA*. IEEE, 1–4.
- [3] Asad Ali and Carmine Gravino. 2019. A systematic literature review of software effort prediction using machine learning methods. *Journal of Software: Evolution and Process* 31, 10 (2019), e2211.
- [4] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2017. Learning to represent programs with graphs. *ICLR* (2017). <https://github.com/Microsoft/gated-graph-neural-network-samples>
- [5] Miltiadis Allamanis, Hao Peng, and Charles Sutton. 2016. A convolutional attention network for extreme summarization of source code. In *ICML*. 2091–2100. <http://groups.inf.ed.ac.uk/cup/codeattention/>
- [6] Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2018. code2seq: Generating sequences from structured representations of code. *ICLR* (2018). <https://github.com/tech-srl/code2seq>
- [7] Sven Amann, Stefanie Beyer, Katja Kevic, and Harald Gall. 2013. Software mining studies: Goals, approaches, artifacts, and replicability. In *Software Engineering*. Springer, 121–158.
- [8] 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. *TSE* 35, 3 (2008), 407–429.

- [9] Anders Arpteg, Björn Brinne, Luka Crnkovic-Friis, and Jan Bosch. 2018. Software engineering challenges of deep learning. In *SEAA*. IEEE, 50–59.
- [10] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
- [11] Matej Balog, Alexander L Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. 2016. Deepcoder: Learning to write programs. *ICLR* (2016).
- [12] Antoine Barbez, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2019. Deep Learning Anti-patterns from Code Metrics History. In *ICSME*. IEEE, 114–124. <https://github.com/antoineBarbez/CAME/>
- [13] Raja Ben Abdesslem, Shiva Nejati, Lionel C Briand, and Thomas Stifter. 2016. Testing advanced driver assistance systems using multi-objective search and neural networks. In *ASE*. 63–74.
- [14] Tal Ben-Nun and Torsten Hoefler. 2019. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR)* 52, 4 (2019), 1–43.
- [15] Tal Ben-Nun, Alice Shoshana Jakobovits, and Torsten Hoefler. 2018. Neural code comprehension: A learnable representation of code semantics. In *NIPS*. 3585–3597. <https://github.com/spcl/ncc>
- [16] Sahil Bhatia, Pushmeet Kohli, and Rishabh Singh. 2018. Neuro-symbolic program corrector for introductory programming assignments. In *ICSE*. IEEE, 60–70.
- [17] Manjubala Bisi and Neeraj Kumar Goyal. 2016. Software development efforts prediction using artificial neural network. *IET Software* 10, 3 (2016), 63–71.
- [18] Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Representing schema structure with graph neural networks for text-to-sql parsing. *ACLs* (2019). <https://github.com/benbogin/spider-schema-gnn>
- [19] 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.
- [20] 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.
- [21] Marc Brockschmidt, Miltiadis Allamanis, Alexander L Gaunt, and Oleksandr Polozov. 2018. Generative code modeling with graphs. *ICLR* (2018). <https://github.com/Microsoft/graph-based-code-modelling>
- [22] Lutz Büch and Artur Andrzejak. 2019. Learning-based recursive aggregation of abstract syntax trees for code clone detection. In *SANER*. IEEE, 95–104.
- [23] Rudy Bunel, Alban Desmaison, M Pawan Kumar, Philip HS Torr, and Pushmeet Kohli. 2016. Learning to superoptimize programs. *ICLR* (2016).
- [24] Rudy Bunel, Matthew Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. 2018. Leveraging grammar and reinforcement learning for neural program synthesis. *ICLR* (2018). https://github.com/bunelr/GandRL_for_NPS
- [25] Jose Cambronero, Hongyu Li, Seohyun Kim, Koushik Sen, and Satish Chandra. 2019. When deep learning met code search. In *FSE*. 964–974.
- [26] Jeffrey C Carver, Natalia Juristo, Maria Teresa Baldassarre, and Sira Vegas. 2014. Replications of software engineering experiments.
- [27] 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 *ICSME*. IEEE, 48–58.
- [28] 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 *ICSE*. 665–676. <http://tagreorder.appspot.com/ui2code.html>
- [29] Guibin Chen, Chunyang Chen, Zhenchang Xing, and Bowen Xu. 2016. Learning a dual-language vector space for domain-specific cross-lingual question retrieval. In *ASE*. IEEE, 744–755.
- [30] 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 *ASE*. IEEE, 364–375.
- [31] Xinyun Chen, Chang Liu, and Dawn Song. 2017. Towards synthesizing complex programs from input-output examples. *ICLR* (2017). <https://drive.google.com/file/d/1WdBIjm4ynv58xCuDaWccm1y0cNC49OPW/view?usp=sharing>
- [32] Xinyun Chen, Chang Liu, and Dawn Song. 2018. Execution-guided neural program synthesis. In *ICLR*. <https://drive.google.com/file/d/17df2etYGoZBP8kRKftpUnzLV2Zc4wS8w/view?usp=sharing>
- [33] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Trang Pham, Aditya Ghose, and Tim Menzies. 2018. A deep learning model for estimating story points. *TSE* 45, 7 (2018), 637–656.
- [34] 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.
- [35] 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.
- [36] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

- [37] 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.
- [38] David Roxbee Cox and Alan Stuart. 1955. Some quick sign tests for trend in location and dispersion. *Biometrika* 42, 1/2 (1955), 80–95.
- [39] Milan Cvitkovic, Badal Singh, and Animashree Anandkumar. 2019. Open vocabulary learning on source code with a graph-structured cache. In *ICML*. PMLR, 1475–1485. <https://github.com/mwcvitkovic/Open-Vocabulary-Learning-on-Source-Code-with-a-Graph-Structured-Cache>
- [40] 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. *ESE* 19, 3 (2014), 501–557.
- [41] 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. *TSE* (2018).
- [42] 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.
- [43] Jayati Deshmukh, Sanjay Podder, Shubhashis Sengupta, Neville Dubash, et al. 2017. Towards accurate duplicate bug retrieval using deep learning techniques. In *ICSME*. IEEE, 115–124.
- [44] Bui Nghi DQ, Yijun Yu, and Lingxiao Jiang. 2019. Bilateral dependency neural networks for cross-language algorithm classification. In *SANER*. IEEE, 422–433. <https://github.com/bdqngi/bi-tbcnn>
- [45] Terrence L Fine. 2006. *Feedforward neural network methodology*. Springer Science & Business Media.
- [46] 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.
- [47] David A Forsyth and Jean Ponce. 2002. *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference.
- [48] Cheng Fu, Huili Chen, Haolan Liu, Xinyun Chen, Yuandong Tian, Farinaz Koushanfar, and Jishen Zhao. 2019. Coda: An end-to-end neural program decompiler. In *NIPS*. 3708–3719.
- [49] Cuiyun Gao, Jichuan Zeng, Xin Xia, David Lo, Michael R Lyu, and Irwin King. 2019. Automating App Review Response Generation. In *ASE*. IEEE, 163–175. <https://github.com/armor-ai/RRGen>
- [50] 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 *SANER*. IEEE, 414–421.
- [51] 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 *ICSME*. IEEE, 145–156.
- [52] 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.
- [53] Patrice Godefroid, Hila Peleg, and Rishabh Singh. 2017. Learn&fuzz: Machine learning for input fuzzing. In *ASE*. IEEE, 50–59.
- [54] Anthony TC Goh. 1995. Back-propagation neural networks for modeling complex systems. *Artificial Intelligence in Engineering* 9, 3 (1995), 143–151.
- [55] Omar S Gómez, Natalia Juristo, and Sira Vegas. 2010. Replications types in experimental disciplines. In *ESEM*. 1–10.
- [56] Omar S Gómez, Natalia Juristo, and Sira Vegas. 2014. Understanding replication of experiments in software engineering: A classification. *IST* 56, 8 (2014), 1033–1048.
- [57] Jesús M González-Barahona and Gregorio Robles. 2012. On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. *ESE* 17, 1-2 (2012), 75–89.
- [58] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.
- [59] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *ICSE*. IEEE, 933–944. <https://github.com/guxd/deep-code-search>
- [60] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep API learning. In *FSE*. 631–642. <http://home.cse.ust.hk/~xguaa/deepapi/>
- [61] Chenkai Guo, Dengrong Huang, Naipeng Dong, Quanqi Ye, Jing Xu, Yaqing Fan, Hui Yang, and Yifan Xu. 2019. Deep review sharing. In *SANER*. IEEE, 61–72. <https://github.com/huangdengrong/Deep-Review-Sharing>
- [62] 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 *SANER*. IEEE, 242–252.
- [63] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. 2017. Semantically enhanced software traceability using deep learning techniques. In *ICSE*. IEEE, 3–14. <https://github.com/jin-guo/TraceNN>
- [64] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. *ACL* (2019). <https://github.com/zhanzecheng/IRNet>
- [65] Rahul Gupta, Aditya Kanade, and Shirish Shevade. 2019. Deep reinforcement learning for syntactic error repair in student programs. In *AAAI*, Vol. 33. 930–937. <https://bitbucket.org/iiscseal/rlassist>

- [66] Rahul Gupta, Aditya Kanade, and Shirish Shevade. 2019. Neural Attribution for Semantic Bug-Localization in Student Programs. In *NIPS*. 11884–11894. <https://bitbucket.org/iiscseal/nbl/src/master/>
- [67] Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish Shevade. 2017. Deepfix: Fixing common c language errors by deep learning. In *AAAI*. 1345–1351. <https://bitbucket.org/iiscseal/deepfix/src/master/>
- [68] Huong Ha and Hongyu Zhang. 2019. DeepPerf: performance prediction for configurable software with deep sparse neural network. In *ICSE*. IEEE, 1095–1106. <https://github.com/DeepPerf/DeepPerf>
- [69] 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.
- [70] Junxiao Han, Emad Shihab, Zhiyuan Wan, Shuiguang Deng, and Xin Xia. 2020. What do Programmers Discuss about Deep Learning Frameworks. *ESE* (2020).
- [71] Zhuobing Han, Xiaohong Li, Zhenchang Xing, Hongtao Liu, and Zhiyong Feng. 2017. Learning to predict severity of software vulnerability using only vulnerability description. In *ICSME*. IEEE, 125–136.
- [72] Vincent J Hellendoorn, Christian Bird, Earl T Barr, and Miltiadis Allamanis. 2018. Deep learning type inference. In *FSE*. 152–162. <https://github.com/DeepTyper/DeepTyper>
- [73] Vincent J Hellendoorn and Premkumar Devanbu. 2017. Are deep neural networks the best choice for modeling source code?. In *FSE*. 763–773. <https://github.com/SLP-team/SLP-Core>
- [74] 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. *TSE* (2019). <https://github.com/hvdthong/PatchNetTool>
- [75] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
- [76] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *ICPC*. 200–210. <https://github.com/xing-hu/DeepCom>
- [77] Xing Hu, Ge Li, Xin Xia, David Lo, Shuai Lu, and Zhi Jin. 2018. Summarizing source code with transferred api knowledge. (2018). <https://github.com/xing-hu/TL-CodeSum>
- [78] Qiao Huang, Xin Xia, David Lo, and Gail C Murphy. 2018. Automating intention mining. *TSE* (2018). <https://github.com/tkdsheep/Intention-Mining-TSE>
- [79] Xuan Huo and Ming Li. 2017. Enhancing the Unified Features to Locate Buggy Files by Exploiting the Sequential Nature of Source Code.. In *IJCAI*. 1909–1915.
- [80] Xuan Huo, Ming Li, Zhi-Hua Zhou, et al. 2016. Learning unified features from natural and programming languages for locating buggy source code.. In *IJCAI*. 1606–1612.
- [81] Xuan Huo, Ferdian Thung, Ming Li, David Lo, and Shu-Ting Shi. 2019. Deep transfer bug localization. *TSE* (2019).
- [82] 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.
- [83] Darrel C Ince, Leslie Hatton, and John Graham-Cumming. 2012. The case for open computer programs. *Nature* 482, 7386 (2012), 485–488.
- [84] Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically generating commit messages from diffs using neural machine translation. In *ASE*. IEEE, 135–146. <https://sjiang1.github.io/commitgen/>
- [85] Natalia Juristo and Omar S Gómez. 2010. Replication of software engineering experiments. In *Empirical software engineering and verification*. Springer, 60–88.
- [86] Natalia Juristo and Sira Vegas. 2011. The role of non-exact replications in software engineering experiments. *ESE* 16, 3 (2011), 295–324.
- [87] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188* (2014).
- [88] Ashwin Kalyan, Abhishek Mohta, Oleksandr Polozov, Dhruv Batra, Prateek Jain, and Sumit Gulwani. 2018. Neural-guided deductive search for real-time program synthesis from examples. *ICLR* (2018).
- [89] Deborah S Katz, Jason Ruchti, and Eric Schulte. 2018. Using recurrent neural networks for decompilation. In *SANER*. IEEE, 346–356.
- [90] 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. *ESE* 25, 1 (2020), 353–401.
- [91] Jinkyu Koo, Charitha Saumya, Milind Kulkarni, and Saurabh Bagchi. 2019. PySE: Automatic Worst-Case Test Generation by Reinforcement Learning. In *ICST*. IEEE, 136–147. <https://bitbucket.org/helix979/pyse/src/master/>
- [92] Lov Kumar and Santanu Ku Rath. 2016. Hybrid functional link artificial neural network approach for predicting maintainability of object-oriented software. *JSS* 121 (2016), 170–190.
- [93] 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 *ASE*. IEEE, 628–639.
- [94] 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 *ICPC*. IEEE, 218–229.

- [95] 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.
- [96] Tue Le, Tuan Nguyen, Trung Le, Dinh Phung, Paul Montague, Olivier De Vel, and Lizhen Qu. 2018. Maximal divergence sequential autoencoder for binary software vulnerability detection. In *ICLR*. <https://github.com/dascimal-org/MDSeqVAE>
- [97] Alexander LeClair, Zachary Eberhart, and Collin McMillan. 2018. Adapting neural text classification for improved software categorization. In *ICSME*. IEEE, 461–472.
- [98] Alexander LeClair, Siyuan Jiang, and Collin McMillan. 2019. A neural model for generating natural language summaries of program subroutines. In *ICSE*. IEEE, 795–806. <https://s3.us-east-2.amazonaws.com/icse2018/index.html>
- [99] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.
- [100] 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 *ICST*. IEEE, 391–401.
- [101] Dor Levy and Lior Wolf. 2017. Learning to align the source code to the compiled object code. In *ICML*. 2043–2051. <https://github.com/DorLevyML/learn-align>
- [102] Liuqing Li, He Feng, Wenjie Zhuang, Na Meng, and Barbara Ryder. 2017. Cclearner: A deep learning-based clone detection approach. In *ICSME*. IEEE, 249–260. <https://github.com/liuqingli/CCLearner>
- [103] Liam Li and Ameet Talwalkar. 2019. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638* (2019).
- [104] Xiaochen Li, He Jiang, Dong Liu, Zhilei Ren, and Ge Li. 2018. Unsupervised deep bug report summarization. In *ICPC*. 144–155.
- [105] Xia Li, Wei Li, Yuqun Zhang, and Lingming Zhang. 2019. Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization. In *ISSTA*. 169–180. <https://github.com/DeepFL/DeepFaultLocalization>
- [106] Yuding Liang and Kenny Q Zhu. 2018. Automatic generation of text descriptive comments for code blocks. *AAAI* (2018). https://github.com/liang2024086/code_comment_generation
- [107] Chin-Yew Lin and FJ Och. 2004. Looking for a few good metrics: ROUGE and its evaluation. In *Ntcir Workshop*.
- [108] 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.
- [109] Chang Liu, Xinyun Chen, Eui Chul Shin, Mingcheng Chen, and Dawn Song. 2016. Latent attention for if-then program synthesis. In *NIPS*. 4574–4582. <https://github.com/Jungyhuk/Latent-Attention>
- [110] Chao Liu, Dan Yang, Xin Xia, Meng Yan, and Xiaohong Zhang. 2018. Cross-project change-proneness prediction. In *COMPSAC*, Vol. 1. IEEE, 64–73.
- [111] Chao Liu, Dan Yang, Xin Xia, Meng Yan, and Xiaohong Zhang. 2019. A two-phase transfer learning model for cross-project defect prediction. *IST* 107 (2019), 125–136.
- [112] Chao Liu, Dan Yang, Xiaohong Zhang, Haibo Hu, Jed Barson, and Baishakhi Ray. 2018. A recommender system for developer onboarding. In *ICSE-C*. 319–320.
- [113] 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.
- [114] 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).
- [115] Hui Liu, Jiahao Jin, Zhifeng Xu, Yifan Bu, Yanzhen Zou, and Lu Zhang. 2019. Deep learning based code smell detection. *TSE* (2019). <https://github.com/liuhuigmail/DeepSmellDetection>
- [116] Hui Liu, Zhifeng Xu, and Yanzhen Zou. 2018. Deep learning based feature envy detection. In *ASE*. 385–396. <https://github.com/liuhuigmail/FeatureEnvy>
- [117] Jin Liu, Pingyi Zhou, Zijiang Yang, Xiao Liu, and John Grundy. 2018. FastTagRec: fast tag recommendation for software information sites. *ASE* 25, 4 (2018), 675–701.
- [118] 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 *ICSE*. IEEE, 1–12. <https://github.com/SerVal-DTF/debug-method-name>
- [119] Peng Liu, Xiangyu Zhang, Marco Pistoia, Yunhui Zheng, Manoel Marques, and Lingfei Zeng. 2017. Automatic text input generation for mobile testing. In *ICSE*. IEEE, 643–653.
- [120] Xiao Liu, Xiaoting Li, Rupesh Prajapati, and Dinghao Wu. 2019. Deepfuzz: Automatic generation of syntax valid c programs for fuzz testing. In *AAAI*, Vol. 33. 1044–1051. <https://github.com/s3team/DeepFuzz>
- [121] Yibin Liu, Yanhui Li, Jianbo Guo, Yuming Zhou, and Baowen Xu. 2018. Connecting software metrics across versions to predict defects. In *SANER*. IEEE, 232–243.
- [122] Zhongxin Liu, Xin Xia, Christoph Treude, David Lo, and Shanping Li. 2019. Automatic Generation of Pull Request Descriptions. (2019). <https://github.com/Tbabm/PRSummarizer>

- [123] Cuauhtémoc López-Martín and Alain Abran. 2015. Neural networks for predicting the duration of new software projects. *JSS* 101 (2015), 127–135.
- [124] Panos Louridas and Georgios Gousios. 2012. A note on rigour and replicability. *ACM SIGSOFT Software Engineering Notes* 37, 5 (2012), 1–4.
- [125] Pablo Loyola, Edison Marrese-Taylor, and Yutaka Matsuo. 2017. A neural architecture for generating natural language descriptions from source code changes. *ACL* (2017). <https://github.com/epochx/commitgen>
- [126] Pablo Loyola and Yutaka Matsuo. 2017. Learning feature representations from change dependency graphs for defect prediction. In *ISSRE*. IEEE, 361–372.
- [127] Jonathan Lung, Jorge Aranda, Steve Easterbrook, and Gregory Wilson. 2008. On the difficulty of replicating human subjects studies in software engineering. In *ICSE*. IEEE, 191–200.
- [128] 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. *TSE* (2019).
- [129] Zaheed Mahmood, David Bowes, Tracy Hall, Peter CR Lane, and Jean Petrić. 2018. Reproducibility and replicability of software defect prediction studies. *IST* 99 (2018), 148–163.
- [130] 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.
- [131] Rabee Sohail Malik, Jibesh Patra, and Michael Pradel. 2019. NL2Type: inferring JavaScript function types from natural language information. In *ICSE*. IEEE, 304–315. <https://github.com/sola-da/NL2Type>
- [132] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *ACL*. 55–60.
- [133] Qing Mi, Jacky Keung, Yan Xiao, Solomon Mensah, and Yujin Gao. 2018. Improving code readability classification using convolutional neural networks. *IST* 104 (2018), 60–71. <https://github.com/CityU-QingMi/DeepCRM>
- [134] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.
- [135] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *ICASSP*. IEEE, 5528–5531.
- [136] Facundo Molina, Renzo Degiovanni, Pablo Ponzio, Germán Regis, Nazareno Aguirre, and Marcelo Frias. 2019. Training binary classifiers as data structure invariants. In *ICSE*. IEEE, 759–770. <https://sites.google.com/site/learninginvariants>
- [137] Kevin Moran, Carlos Bernal-Cárdenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk. 2018. Machine learning-based prototyping of graphical user interfaces for mobile apps. *TSE* (2018).
- [138] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2014. Convolutional neural networks over tree structures for programming language processing. *AAAI* (2014). <https://sites.google.com/site/treebasedcnn/>
- [139] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional neural networks over tree structures for programming language processing. In *AAAI*.
- [140] 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 *ASE*. IEEE, 924–935.
- [141] Vijayaraghavan Murali, Letao Qi, Swarat Chaudhuri, and Chris Jermaine. 2017. Neural sketch learning for conditional program generation. *ICLR* (2017). <https://github.com/trishullab/bayou>
- [142] 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 *ASE*. IEEE, 1026–1037. <https://github.com/Kawser-nerd/CLCDSA>
- [143] 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).
- [144] Arvind Neelakantan, Quoc V Le, Martin Abadi, Andrew McCallum, and Dario Amodei. 2016. Learning a natural language interface with neural programmer. *ICLR* (2016).
- [145] 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 *SANER*. IEEE, 323–334.
- [146] Maxwell Nye, Luke Hewitt, Joshua Tenenbaum, and Armando Solar-Lezama. 2019. Learning to infer program sketches. *ICML* (2019). https://github.com/mtensor/neural_sketch
- [147] Yin Pengcheng, Lu Zhengdong, Li Hang, and Kao Ben. 2016. Neural Enquirer: Learning to Query Tables in Natural Language. In *IJCAI*.
- [148] Chris Piech, Jonathan Huang, Andy Nguyen, Mike Phulsuksombati, Mehran Sahami, and Leonidas Guibas. 2015. Learning program embeddings to propagate feedback on student code. *ICML* (2015).
- [149] 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).
- [150] Yewen Pu, Zachery Miranda, Armando Solar-Lezama, and Leslie Kaelbling. 2018. Selecting representative examples for program synthesis. In *ICML*. PMLR, 4161–4170. https://github.com/evanthebouncy/icml2018_selecting_representative_examples

- [151] Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. *ACL* (2017).
- [152] Pooja Rani and GS Mahapatra. 2018. Neural network for software reliability analysis of dynamically weighted NHPP growth models with imperfect debugging. *STVR* 28, 5 (2018), e1663.
- [153] 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. *TOSEM* 28, 3 (2019), 1–45.
- [154] Stephen Romansky, Neil C Borle, Shaiful Chowdhury, Abram Hindle, and Russ Greiner. 2017. Deep green: Modelling time-series of software energy consumption. In *ICSME*. IEEE, 273–283.
- [155] Hang Ruan, Bihuan Chen, Xin Peng, and Wenyun Zhao. 2019. DeepLink: Recovering issue-commit links based on deep learning. *JSS* 158 (2019), 110406. <https://github.com/ruanhang1993/DeepLink>
- [156] Hasim Sak, Andrew W Senior, and Françoise Beaufays. 2014. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. (2014).
- [157] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80. <https://doi.org/10.1109/TNN.2008.2005605>
- [158] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.
- [159] 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).
- [160] Shu-Ting Shi, Ming Li, David Lo, Ferdian Thung, and Xuan Huo. 2019. Automatic code review by learning the revision of source code. In *AAAI*, Vol. 33. 4910–4917.
- [161] Yusuxke Shibata, Takuya Kida, Shuichi Fukamachi, Masayuki Takeda, Ayumi Shinohara, Takeshi Shinohara, and Setsuo Arikawa. 1999. *Byte Pair encoding: A text compression scheme that accelerates pattern matching*. Technical Report. Technical Report DOI-TR-161, Department of Informatics, Kyushu University.
- [162] Eui Chul Shin, Illia Polosukhin, and Dawn Song. 2018. Improving neural program synthesis with inferred execution traces. In *NIPS*. 8917–8926.
- [163] Richard Shin, Neel Kant, Kavi Gupta, Christopher Bender, Brandon Trabucco, Rishabh Singh, and Dawn Song. 2019. Synthetic datasets for neural program synthesis. *ICLR* (2019).
- [164] Chengxun Shu and Hongyu Zhang. 2017. Neural programming by example. *AAAI* (2017).
- [165] Jianhang Shuai, Ling Xu, Chao Liu, Meng Yan, Xin Xia, and Yan Lei. 2020. Improving Code Search with Co-Attentive Representation Learning. (2020).
- [166] Xujie Si, Hanjun Dai, Mukund Raghothaman, Mayur Naik, and Le Song. 2018. Learning loop invariants for program verification. In *NIPS*. 7751–7762. <https://github.com/PL-ML/code2inv>
- [167] 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. *TSE* 31, 9 (2005), 733–753.
- [168] Shao-Hua Sun, Hyeonwoo Noh, Sriram Somasundaram, and Joseph Lim. 2018. Neural program synthesis from diverse demonstration videos. In *ICML*. 4790–4799. <https://shaohua0116.github.io/demo2program/>
- [169] Zeyu Sun, Qihao Zhu, Lili Mou, Yingfei Xiong, Ge Li, and Lu Zhang. 2019. A grammar-based structural cnn decoder for code generation. In *AAAI*, Vol. 33. 7055–7062. <https://github.com/zysszy/GrammarCNN>
- [170] Alexey Svyatkovskiy, Ying Zhao, Shengyu Fu, and Neel Sundaresan. 2019. Pythia: ai-assisted code completion system. In *KDD*. 2727–2735.
- [171] Richard Szeliski. 2010. *Computer vision: algorithms and applications*. Springer Science & Business Media.
- [172] Hannes Thaller, Lukas Linsbauer, and Alexander Egyed. 2019. Feature maps: A comprehensible software representation for design pattern detection. In *SANER*. IEEE, 207–217.
- [173] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deepest: Automated testing of deep-neural-network-driven autonomous cars. In *ICSE*. 303–314.
- [174] Haonan Tong, Bin Liu, and Shihai Wang. 2018. Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. *IST* 96 (2018), 94–111.
- [175] Michele Tufano, Jevgenija Pantiuchina, Cody Watson, Gabriele Bavota, and Denys Poshyvanyk. 2019. On learning meaningful code changes via neural machine translation. In *ICSE*. IEEE, 25–36. <https://sites.google.com/view/learning-codechanges>
- [176] 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 *ICSME*. IEEE, 301–312.
- [177] 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. *TOSEM* 28, 4 (2019), 1–29. <https://sites.google.com/view/learning-fixes>
- [178] Lazar Valkov, Dipak Chaudhari, Akash Srivastava, Charles Sutton, and Swarat Chaudhuri. 2018. Houdini: Lifelong learning as program synthesis. In *NIPS*. 8687–8698. <https://github.com/trishullab/houdini>

- [179] Marko Vasic, Aditya Kanade, Petros Maniatis, David Bieber, and Rishabh Singh. 2019. Neural program repair by jointly learning to localize and repair. *ICLR* (2019).
- [180] 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).
- [181] 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 *ASE*. IEEE, 13–25. https://github.com/wanyao1992/mman_public
- [182] 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 *ASE*. 397–407. https://github.com/wanyao1992/code_summarization_public
- [183] Changhan Wang, Kyunghyun Cho, and Jiatao Gu. 2020. Neural machine translation with byte-level subwords. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 9154–9160.
- [184] Ke Wang, Rishabh Singh, and Zhendong Su. 2017. Dynamic neural program embedding for program repair. *ICLR* (2017). <https://github.com/keowang/dynamic-program-embedding>
- [185] Song Wang, Taiyue Liu, Jaechang Nam, and Lin Tan. 2018. Deep semantic feature learning for software defect prediction. *TSE* (2018).
- [186] Song Wang, Taiyue Liu, and Lin Tan. 2016. Automatically learning semantic features for defect prediction. In *ICSE*. IEEE, 297–308.
- [187] Xu Wang, Chunyang Chen, and Zhenchang Xing. 2019. Domain-specific machine translation with recurrent neural network for software localization. *ESE* 24, 6 (2019), 3514–3545.
- [188] 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 *ISSRE*. IEEE, 239–249.
- [189] Bolin Wei, Ge Li, Xin Xia, Zhiyi Fu, and Zhi Jin. 2019. Code generation as a dual task of code summarization. In *NIPS*. 6563–6573. <https://github.com/code-gen/cscg>
- [190] Huihui Wei and Ming Li. 2017. Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code.. In *IJCAI*. 3034–3040.
- [191] Ming Wen, Rongxin Wu, and Shing-Chi Cheung. 2018. How well do change sequences predict defects? sequence learning from software changes. *TSE* (2018).
- [192] Martin White, Michele Tufano, Matias Martinez, Martin Monperrus, and Denys Poshyvanyk. 2019. Sorting and transforming program repair ingredients via deep learning code similarities. In *SANER*. IEEE, 479–490. <https://github.com/SpoonLabs/astor>
- [193] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. 2016. Deep learning code fragments for code clone detection. In *ASE*. IEEE, 87–98. <https://github.com/micheletufano/AutoenCODE>
- [194] Frank Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometrics bulletin* 1, 6 (1945), 80–83.
- [195] Yan Xiao, Jacky Keung, Kwabena E Bennin, and Qing Mi. 2019. Improving bug localization with word embedding and enhanced convolutional neural networks. *IST* 105 (2019), 17–29. <https://github.com/yanxiao6/BugLocalization-dataset>
- [196] 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 *SANER*. IEEE, 434–444.
- [197] 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 *ASE*. IEEE, 51–62.
- [198] Shengbin Xu, Yuan Yao, Feng Xu, Tianxiao Gu, Hanghang Tong, and Jian Lu. 2019. Commit Message Generation for Source Code Changes.. In *IJCAI*. 3975–3981. <https://github.com/SoftWiser-group/CoDiSum>
- [199] 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. *JSS* 158 (2019), 110402. <https://sites.google.com/view/jss-ldfr>
- [200] Meng Yan, Mengning Yang, Chao Liu, and Xiaohong Zhang. 2016. Self-learning Change-prone Class Prediction. In *SEKE*. 134–140.
- [201] 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. *IST* 92 (2017), 1–16.
- [202] 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 *ICSE-C*. IEEE, 268–270.
- [203] Ruiho Yan, Xi Xiao, Guangwu Hu, Sancheng Peng, and Yong Jiang. 2018. New deep learning method to detect code injection attacks on hybrid applications. *JSS* 137 (2018), 67–77.
- [204] Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. *ACL* (2017). <https://github.com/pcyin/NL2code>
- [205] 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 *ICPC*. IEEE, 70–80. <https://github.com/yh1105/TBCCD>
- [206] Carmen Zannier, Grigori Melnik, and Frank Maurer. 2006. On the success of empirical studies in the international conference on software engineering. In *ICSE*. 341–350.
- [207] 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 *ICSE*. IEEE, 783–794. <https://github.com/zhangj111/astnn>

- [208] Lisa Zhang, Gregory Rosenblatt, Ethan Fetaya, Renjie Liao, William Byrd, Matthew Might, Raquel Urtasun, and Richard Zemel. 2018. Neural guided constraint logic programming for program synthesis. In *NIPS*. 1737–1746. <https://github.com/xuexue/neuralkanren>
- [209] Yan-Ya Zhang and Ming Li. 2019. Find Me if You Can: Deep Software Clone Detection by Exploiting the Contest between the Plagiarist and the Detector. In *AAAI*, Vol. 33. 5813–5820.
- [210] Zhuo Zhang, Yan Lei, Xiaoguang Mao, and Panpan Li. 2019. CNN-FL: An effective approach for localizing faults using convolutional neural networks. In *SANER*. IEEE, 445–455.
- [211] Dehai Zhao, Zhenchang Xing, Chunyang Chen, Xin Xia, and Guoqiang Li. 2019. ActionNet: vision-based workflow action recognition from programming screencasts. In *ICSE*. IEEE, 350–361. https://github.com/DehaiZhao/ActionNet_0
- [212] Gang Zhao and Jeff Huang. 2018. Deepsim: deep learning code functional similarity. In *FSE*. 141–151. <https://github.com/parasol-aser/deepsim>
- [213] Jinman Zhao, Aws Albarghouthi, Vaibhav Rastogi, Somesh Jha, and Damien Ocateau. 2018. Neural-augmented static analysis of Android communication. In *FSE*. 342–353.
- [214] 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 *ASE*. IEEE, 772–784. <https://github.com/NeteaseFuxiRL/wuji>
- [215] Liu Zhongxin, Xia Xin, Lo David, Xing Zhenchang, E. Hassan Ahmed, and Li Shanping. 2019. Which Variables Should I Log? *TSE* (2019).
- [216] 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? *IST* 109 (2019), 1–13.
- [217] Yu Zhou, Xin Yan, Wenhua Yang, Taolue Chen, and Zhiqiu Huang. 2019. Augmenting Java method comments generation with context information based on neural networks. *JSS* 156 (2019), 328–340.
- [218] Amit Zohar and Lior Wolf. 2018. Automatic program synthesis of long programs with a learned garbage collector. In *NIPS*. 2094–2103. <https://github.com/amitz25/PCCoder>