Pitfalls in Experiments with DNN4SE: An Analysis of the State of the Practice

Sira Vegas
sira.vegas@upm.es
Universidad Politécnica de Madrid
Madrid, Spain

Sebastian Elbaum
selbaum@virginia.edu
University of Virginia
Charlottesville, Virginia, USA

ABSTRACT

Software engineering techniques are increasingly relying on deep learning approaches to support many software engineering tasks, from bug triaging to code generation. To assess the efficacy of such techniques researchers typically perform controlled experiments. Conducting these experiments, however, is particularly challenging given the complexity of the space of variables involved, from specialized and intricate architectures and algorithms to a large number of training hyper-parameters and choices of evolving datasets, all compounded by how rapidly the machine learning technology is advancing, and the inherent sources of randomness in the training process. In this work we conduct a mapping study, examining 194 experiments with techniques that rely on deep neural networks appearing in 55 papers published in premier software engineering venues to provide a characterization of the state of the practice, pinpointing experiments' common trends and pitfalls. Our study reveals that most of the experiments, including those that have received ACM artifact badges, have fundamental limitations that raise doubts about the reliability of their findings. More specifically, we find: 1) weak analyses to determine that there is a true relationship between independent and dependent variables (87% of the experiments), 2) limited control over the space of DNN relevant variables, which can render a relationship between dependent variables and treatments that may not be causal but rather correlational (100% of the experiments), and 3) lack of specificity in terms of what are the DNN variables and their values utilized in the experiments (86% of the experiments) to define the treatments being applied, which makes it unclear whether the techniques designed are the ones being assessed, or how the sources of extraneous variation are controlled. We provide some practical recommendations to address these limitations.

CCS CONCEPTS

• **Software and its engineering → Empirical software validation**; • **Computing methodologies → Neural networks**.

This work was supported by grant PID2022-137846NB-I00 funded by MCIN/AEI/10.13039/501100011033, by "ERDF A way of making Europe", and by NSF grant #2019239.

ESEC/FSE '23, December 3–9, 2023, San Francisco, CA, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0327-0/23/12…$15.00
https://doi.org/10.1145/3611643.3616320

KEYWORDS

deep learning, machine learning for software engineering, software engineering experimentation

ACM Reference Format:
Sira Vegas and Sebastian Elbaum. 2023. Pitfalls in Experiments with DNN4SE: An Analysis of the State of the Practice. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3611643.3616320

1 INTRODUCTION

The application of deep learning (DL) techniques across the software development life cycle is becoming a thriving research thread in the software engineering (SE) community. Such emerging techniques, often grouped under labels such as DL4SE or DNN4SE, have rendered promising results supporting the automation of activities ranging from requirements engineering to code maintenance [6]. Similar to other DL application areas, the maturation of frameworks and tools that lowered the bar for the adoption for such technology has facilitated their application in the SE domain. In addition, our community is in an advantageous position in that we can tap into a continuously increasing number of public repositories with various types of software artifacts such as code and tests that constitute rich data sets on which DL techniques can thrive.

To assess such techniques, researchers perform experiments in which variables are manipulated in a controlled environment to investigate their impact over response variables [15]. Conducting such experiments, however, can be extremely challenging given the number and complexity of variables that may affect a technique that relies on DL. Tens of variables play a fundamental role in how a deep neural network (DNN) is set up as part of an experiment. Some of these variables are inherently complex as they point to optimization procedures that contain their own set of parameters. Other variables like those associated with datasets or competing models often point to online resources that may unsuspectingly evolve. Other variables, like those affecting the sample used by gradient descent to set the network weights or the proportions of data used for training and testing, are deceptively simple, yet they constitute sources of randomness that will impact the DNN's performance. Yet other variables that may not be explicitly defined, like the ones defining termination criteria, can have subtle interactions with other variables undermining the implementation of the intended experimental constructs. The key takeaway is that when evaluating the application of a DL technique to a problem through an experiment, the lack of careful consideration of a complex set of variables can dramatically impact the findings.

The **goal** of this paper is to begin understanding the extent to which experiments on DNN4SE techniques are addressing the distinct experimental challenges introduced by DNNs.

In pursuing that goal we make four contributions.

I) We contribute a characterization and analysis of the state of the practice of experimentation with DNN4SE by addressing a fundamental question: **RQ1: To what extent are DNN4SE experiments specified in papers?** To answer that question, in Section 3, we present a systematic mapping study [17] of 55 papers from ICSE, FSE, and TSE from 2018-2021 that apply DL techniques to automate SE tasks. Building on a cause-effect model of the experimental space and the variables relevant to DNNs, we determine the degree to which the variable space in the experiments was specified by each paper. We find that while most experiments clearly identify, for example, the response variables (76%) and training data (69%), none describe their complete space of variables. Furthermore, most experiments lack in critical aspects like the choice for experimental design to control the sources of variability (30%) and the use of even descriptive statistics as part of the results analysis and interpretation (56%). This lack of specificity is not just an under-reporting issue, but it reflects a limited consideration of fundamental experimental aspects that threaten the validity of the findings.

II) Given the community ongoing efforts for sharing artifacts [31], we extend the previous characterization through **RQ2. Do shared artifacts improve the specifications of DNN4SE experiments provided in the papers?** Section 4 contributes an analysis of the artifacts associated with the subset of papers that earned ACM artifact badges, increasing the depth of analysis to include code, data, and documentation. As expected, artifacts complement some but not all aspects presented in the papers, especially the definition of variables and the training and test data, all of which are necessary to operationalize the experiments. However, we also find that 68% of the experiments reported in the artifacts present inconsistencies when compared with the corresponding paper, ranging from the loss function to the testing data being used. This is problematic because the additional effort invested to prepare artifacts to further support the experiments often raises doubts about which portions of the papers and the artifacts are to be trusted.

III) We contribute an analysis of why these findings matter through **RQ3. What are the implications of the previous findings about the under-specification of DNN4SE experiments?** Section 5 summarizes these implications. First, by failing to clearly define factors and treatments in 86% of the experiments, it is unclear whether most experimental results are caused by the intended constructs or by other variables that were not operationalized correctly. In the best of cases, one could argue that those unspecified variables in the papers are controlled when the experiments are performed. However, our analysis of artifacts reveals that that is rarely the case. Second, even when variables are specified it is often unclear how they are controlled to establish causality. We find that 62% of experiments account for sources of randomness related to the dataset, and none controlled for other sources of training randomness by, for example, performing multiple training runs or varying the DNN initial weights. Third, we find that 56% of experiments identify relationships between independent and dependent variables based on single observations which is suspect as it ignores any experimental fluctuation.

IV) **Recommendations.** We are not the first community challenged by the DNN complexity. The Artificial Intelligence (AI) community has developed various checklists to mitigate common ML experimental pitfalls [1, 18, 22]. Similarly, the SE community has developed a body of knowledge to assess and improve the quality of the experiments we conduct (see related work in Section 8). However, as it shall become clear from our RQ1-RQ2-RQ3 findings, there is a distinct and urgent need for the SE community to become much more cognizant of how to manage the space of variables particular to the DNN domain. Towards that end, we recommend actionable practices to manage the challenges in DNN4SE experimentation (Section 7) that, if adopted, can alleviate many of the concerns we encountered. For example, simply conducting multiple DNN training runs to control for randomness could benefit almost all experiments, performing more comparisons over multiple observations to account for experimental variability in DNNs could benefit from 56% to 87% of the experiments, and standardizing a minimal specification of the space of DNN training variables and providing partial automation for synchronizing the paper and artifact content of DNN4SE experiments could benefit 96% of the experiments.

2 DNNS' EXPERIMENTAL VARIABLES

Machine learning (ML) is a subfield of AI that aims to enable computers to learn from experience [11, 19]. ML algorithms build a model based on sample (training) data to make predictions without being explicitly programmed to do so [26]. DL is a type of ML technique supported by neural networks that have a deep architecture as per their constituting layers [11]. The training of these DNNs consists of adjusting its **model parameters**, using a **deep learning algorithm** controlled by a set of **training hyperparameters** and **model hyperparameters**, using a **dataset** [11]. We will later use these 5 groups of variables associated with DNNs, illustrated through a cause-effect diagram in Figure 1, as a basis for the analysis of experiments.

The DNN overall architecture is defined by the **model hyperparameters** and includes 5 variables. DNNs consist of interconnected *neurons* grouped in *layers*. There is always an input layer that accepts inputs and an output layer that provides the output, and hidden layers between them. There are different *layer types*, and how those layers are connected define the higher-level architecture of the DNN (e.g., feed-forward, CNN, RNN, LSTM). Every connection between 2 neurons has a weight which regulates how much of the initial value will be forwarded to a given neuron. Each neuron has an associated value, called the bias. Weights and biases need to be *initialized* prior to training. The sum of the products of the inputs and respective weights, plus the bias, are then provided to an *activation function* to produce a neuron's output. A forward pass is the set of calculations that take place when the input travels through the DNN to the output.

The neuron *weights* and *biases* constitute the two variables defining the DNNs **model parameters**. They are initialized before training and reset during the subsequent training process.

A **dataset** is a collection of inputs and outputs. At least two types of datasets are required: *training* and *test*. The training set is

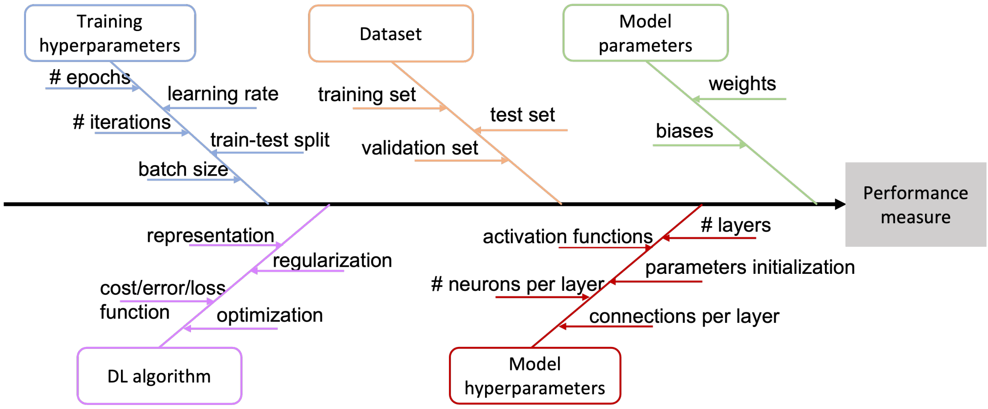

Figure 1: A cause-and-effect diagram for 5 groups of variables in experiments with DNNs.

used to adjust the model parameters during training. The test set is used to check how well the algorithm performs on data that has not seen before, and it is intended to estimate the generalization error. The training set can be further divided into a training and a *validation set*. This validation set can be used to get an estimate of model skill while tuning its hyperparameters.

The **DL algorithm** is defined through 4 variables [11]: a *representation* for encoding the elements in the dataset, a *function measuring the error* between the value predicted by the model and the real value, an *optimization* procedure to minimize the training error (e.g. stochastic gradient descent, Nesterov momentum, Adam), and *regulatization* strategies to reduce the generalization (test) error (e.g. dropout, data augmentation, early stopping).

During training, the DL algorithm's behaviour is controlled through 5 **training hyperparameters** [11]. The *batch size* defines the number of training samples to consider per training *iteration*. Depending on the batch size, multiple iterations will be needed to go through the entire training set. The *number of epochs* defines how many times the algorithm will go through a dataset. The *train-test split* defines on what portion of the data training is performed. Given a batch, the network performance (measured as a function of error/cost/loss) is used to drive the backpropagation (the reverse of a forward pass using gradient descent) to update the network weights and biases to minimize this error. The *learning rate* specifies how much to update the model in response to the estimated error.

Albeit simplified and limited for exposition, this section highlights the vast space of variables involved in training a DL system, where each one can take an increasing number of values. These variables also have many subtle interdependencies (e.g., the batch and epoch size often depend on the parameter initialization, the loss function depends on the architecture, the architecture depends on the data dimensionality). Confounded with the multiple sources of randomness involved in the DL training process (e.g., different train-test partitions, different sample batches being selected, different portions of the network being targeted for regularization, different supported hardware), defining and conducting robust experiments is intrinsically challenging.

3 ANALYSIS OF PAPERS

In this section we answer **RQ1** by providing an overview of the state of the practice in performing experiments where DNNs are utilized to address SE challenges (DNN4SE). We characterize the growing number of experiments being carried out in this domain and identify some overarching limitations across those experiments.

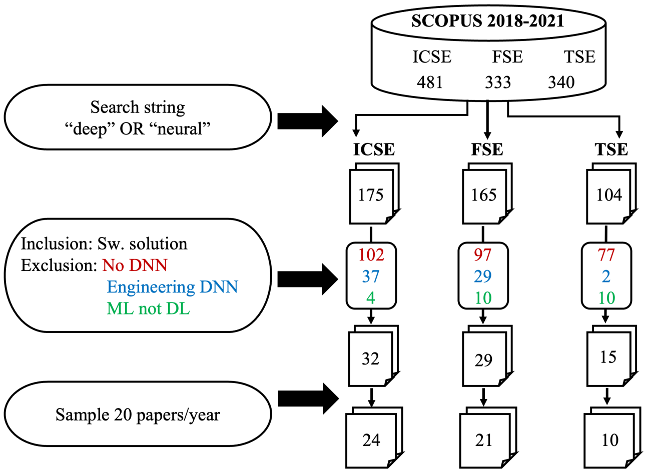

Figure 2: Paper search and selection process.

3.1 Scope of Analysis

We have performed a semi-automated search of papers reporting experiments with DNNs developed to solve SE tasks. Figure 2 summarizes the search and selection process. We have shared in the paper repository the outputs of each step of the selection process.

In a first automated step, during January 2022, we searched SCOPUS^{TM} using the string "deep OR neural" in all fields. The search was limited to full papers from the technical track of the International Conference on Software Engineering (ICSE) and the Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), and papers published in IEEE Transactions on Software Engineering (TSE), covering the period 2018-2021. We decided to favor the flagship conferences ICSE and FSE because we believe they include the latest work in DL and appear at the top of various ranks[1]. Similarly, we selected TSE because it has the highest impact factor among SE journals[2]. The search resulted in 444 out of 1154 published papers.

[1] https://csrankings.org, http://portal.core.edu.au/conf-ranks
[2] https://www.scimagojr.com, https://jcr.clarivate.com/jcr/home

Table 1: Number of papers within scope analyzed / published, and experiments analyzed (in parenthesis)

	ICSE	ESEC/FSE	TSE	Total
2018	2/2 (5)	4/4 (11)	0 (0)	6/6 (16)
2019	8/8 (27)	4/4 (9)	1/1 (6)	13/13 (42)
2020	7/7 (25)	6/6 (21)	3/3 (14)	16/16 (60)
2021	7/15 (21)	7/15 (32)	6/11 (23)	20/41 (76)
Total	24/32 (78)	21/29 (73)	10/15 (43)	55/76 (194)

Next, we excluded the papers that did not cover techniques using DNNs to address software engineering challenges. The examination was conducted by one of the senior researchers authoring this paper with expertise in empirical software engineering and DNN development. This process led to the exclusion of 276 papers that did not include a DNN (e.g. a DNN-solution is part of the related work described in a paper), 68 papers that focus on improving the engineering of DNN-solutions (e.g. testing of DNNs), and 24 papers that use ML mechanisms but not DNNs (e.g., shallow networks). When papers that did not clearly fit in existing categories were found, they were examined and discussed jointly by both authors. The remaining 76 papers report experiments with DL-based software to address SE challenges.

Table 1 shows the paper count distribution over the years and venues. We can see that the number of papers in this area is steadily increasing over the last few years, from 6 papers in 2018 to 41 papers in 2021. For the subsequent analysis we selected every paper identified as within scope from 2018 to 2020, and given the larger number of relevant papers published in 2021 (from 16 in 2020 to 41 in 2021), we randomly sampled 20 papers from 2021 across all venues. This sampling was necessary to control the cost of the study given that just data extraction time per paper was approximately 4 hours per person (we later describe the analysis costs per paper and artifact). This gave us a total of 55 papers spanning four years to analyze.

3.2 Analysis Process

To have a consistent data extraction process from the papers, we defined a set of scoping and analysis guidelines.

First, for each paper, we initially considered just the contents of the published paper. At this early examination stage we did not peak into artifacts that may be associated with the paper like code repositories as we wanted to have a common baseline of materials among all analyzed papers. This also made the analysis cost more viable at the first stage of the study. In addition, for each experiment in a paper we bounded the analysis to the DNN portions. That is, when we found experiments comparing the performance of DNNs against other type of approaches that employ traditional SE approaches or humans, we deemed those portions of the experiment as already understood by the community and only focused on the portions including DNNs.

Second, we controlled for three common sources of uncertainty we faced when analyzing the papers. To control for different reporting styles, we examined the papers in their totality as we often found portions of the experiments distributed and modified throughout the paper. For example, we found instances where the experimental designs are sprinkled through background, approach, study design, and results. To control for DNN usage types, we only considered papers that use DNNs to perform either complex data encodings or function as a model. Third, to control for different levels of detail across experiments, we decided to account for all experiments mentioned in the paper, even if marginally reported.

Given the previous guidelines, the analysis process started with both authors jointly developing an initial characterization schema for the experiments. This schema is based on the steps of the experimental process [10, 20, 32], although we adapted those steps to account for the types of variables found in DNNs such as the model hyperparameters, the training hyperparameters, the DL algorithm, the dataset(s), and the model parameters (as per Figure 1). Then, both authors conducted a refinement and calibration cycle by extracting the information from all the experiments reported in the 17 ICSE papers from 2018 to 2020 according to the schema. This resulted in a refined schema and a more consistent evaluation process. Finally, each remaining paper was examined by just one author. However, when experiments did not fit the schema, introduced new DNN elements, or had ambiguous specifications, they were examined and discussed jointly by the researchers. There were 8 of such joint examinations, lasting between 1-3 hours, which often triggered the re-examination of previously evaluated papers to ensure their consistent analysis.

Table 2 exemplifies the analysis we performed for an experiment. The goal of this paper [AP31] is to perform log-based anomaly detection. The proposed DNN receives as input a sequence of log events and predicts whether the sequence is an anomaly. The experiment evaluates the performance of the proposed DNN in terms of precision, recall, and F1, comparing it against 4 other approaches (none are DNNs). Column 1 and 2 show the steps and aspects of the experimental process, and column 3 assesses to what extent the information has been identified (Fully, Partially, or Missing). The last column provides an explanation of what is lacking. For this experiment, we have been able to find all information related to research hypotheses, DL algorithm, response variables and test set characteristics. For this reason, their final assessment is "Fully" addressed. We have not been able to find any information related to model parameters nor statistics, and therefore, their final assessment is "Missing". For the rest we have not able to find some information, therefore, the final assessment is "Partially" addressed. A detailed description of the classification criteria and its application to all the 55 analyzed papers is available in the repository (Section 10).

3.3 Findings

Table 3 summarizes the findings for the 194 experiments analyzed across the 55 identified target papers. It is encouraging to find that most experiments specify at least to some extent the response variables, the research hypotheses, and the training and test set data. However, the rest of the experimental aspects tend to be underspecified. We find that 50% (7 out of 14) of the aspects are partially addressed, while another 21% (3 out of 14) of the aspects are missing among the experiments detailed in the papers.

We find that essential aspects are missing in most experiments. For example, for the model parameters to be fully addressed, we

Table 2: Assessing of a sampled experiment [AP31] specification in terms of Fully addressed, Partially addressed, or Missing.

Step	Aspect	Assessment	What is lacking
S1. Hypotheses formulation	Research hypotheses	Fully	
S2. Variables identification	Model hyperparameters	Partially	Missing hyperparameters for initialization
	Model parameters	Missing	Missing a pointer to where they can be found
	DL algorithm	Fully	
	Training hyperparameters	Partially	Missing train-test split and learning rate
	Training data	Partially	No information about a dataset for confidentiality reasons
S3. Operationalization	Factors and treatments	Partially	Some model and training hyperparameters are missing, not all training data available, and model parameters are missing
	Response variables	Fully	
S4. Design	Choice of design	Partially	No analysis of sources of randomness, whether they have been controlled, and if so, the mechanism used
	Instrumentation	Partially	One test set is missing due to confidentiality issues. Software environment is not defined. Measuring instruments and procedure can be deduced but are not defined
S5. Objects selection	Test set chars.	Fully	
S6. Analysis & interpretation	Descriptive statistics	Missing	No descriptive statistics reported
	Inferential statistics	Missing	No inferential statistics reported
S7. Validity evaluation	Validity threats	Partially	Missing internal, construct and conclusion

Table 3: Characterization of 194 experiments with DNNs

Step	Aspect	Full	Partial	Missing
S1	Research hypotheses	**76%**	0%	24%
S2	Model hyperparameters	7%	**85%**	8%
	Model parameters	2%	0%	**98%**
	DL algorithm	26%	**72%**	2%
	Training hyperparameters	19%	**73%**	8%
	Training data	**69%**	27%	4%
S3	Factors and treatments	14%	**82%**	4%
	Response variables	**76%**	18%	6%
S4	Choice of design	0%	**70%**	30%
	Instrumentation	2%	**97%**	1%
S5	Test set characteristics	**59%**	19%	22%
S6	Descriptive statistics	10%	34%	**56%**
	Inferential statistics	12%	1%	**87%**
S7	Validity threats	2%	**79%**	19%

required a pointer to a repository where they could be found. Such pointer was lacking for 98% of the experiments. For the choice of (experimental) design to be fully addressed we required a description of what variables are manipulated or controlled and how, yet 30% of the experiments did not have it. For the analysis and interpretation (S6) to be fully addressed we required descriptive and inferential statistics, yet they were missing for 56% and 87% of the experiments respectively. These results at least raise doubts about whether most of the papers are: 1) implementing the construct they are intending, 2) performing meaningful assessments given the experimental noise that is not accounted for by the analysis and interpretation, and 3) establishing causality given the limited amount of control over the large and complex space of variables to be specified.

4 ANALYSIS OF ARTIFACTS

In Section 3 our analysis of papers revealed that the under-specification of experiments with approaches that use DL to address SE problems is pervasive. Still, given our community growing practice towards artifact sharing [31] and the nature of DL experiments (i.e., large open datasets, common architectures, standard APIs), it seems reasonable to ask whether the missing portions of the experiments specifications appear in the shared artifacts. This is also important as it may let us understand if the problem is just one associated with how experiments are reported or if there is a deeper concern about how the experiments are being conducted.

We begin to answer **RQ2** through an analysis of the artifacts associated with those papers to assess the degree to which the under-specification in the papers is complemented by the associated artifacts, and whether the design and analysis limitations identified are mitigated by the artifacts.

4.1 Scope of Analysis

Forty-eight out of 55 papers point to some kind of external artifact. A cursory analysis of those artifacts reveals that their content (from just readmes plus code to experimental results and even new experiments), availability (from broken links to pointers to private repositories or Zenodo), and quality (from a model dump without any explanation to those including a code base to reproduce the results in the paper) had too much variance to define a standardized analysis that would render meaningful findings. This finding is consistent with recent reports on artifact quality [31].

Thus, to get a more precise estimate of the degree of under-specification when considering artifacts, we reduce the scope of analysis to the artifacts associated with the 9 papers (including a total of 44 experiments) that earned at least one of the ACM artifact badges[3] [8]. This reduced scope allows us to focus more deeply on papers vetted (to various degrees) by a conference committee according to established guidelines regarding their completeness and quality.

[3] ACM defines three badges: Artifacts Evaluated (successfully completed an independent audit, with two levels: Functional and Reusable), Artifacts Available (available for retrieval), and Results Validated (results obtained by a team other than the original, with two levels: Results Reproduced and Results Replicated). The 9 papers we analyzed earned the Artifacts Available badge, and three of them also earned the Artifacts Evaluated (two Reusable [AP5, AP39] and one Reusable and Functional [AP36]).

4.2 Analysis Process

We analyzed all artifacts with the following process. First, we examined the readme files and other introductory documentation to get a broad sense of what the artifact was meant to provide. Second, we systematically explored the artifact directories and their contents to identify the resources of information to collect the data required for Table 2. Third, we analyzed the code broadly construed to include Python or C code, configuration files, and batch scripts. The analysis was first meant to map each experiment reported in the paper to the items in the artifact. Although conceptually simple, this analysis process was anything but straight-forward as the artifact structure rarely matched that of the paper (where the experiments are reported). In most cases, we had to recover portions of one or multiple experiments from undocumented code. This required multiple inspections of the code, running portions of it to confirm what was learned through the code inspections, and referencing back the findings to the information in the paper. Fourth, for each experiment identified in the artifact, we collected metadata such as the one reported in Table 2 (more details about the information collected are provided in the repository described in Section 10). During this step we also determined whether the artifact improved or complemented the information provided in the paper, and recorded any inconsistencies we found between them. These steps required approximately 8 hours per paper ([AP39] was an exception given the number of experiments reported). The difficulties in this process, particularly in the third and fourth steps, and the time allocated per paper, forced us to be conservative in our assessment, only judging an artifact experiment to be incomplete or inconsistent with the paper when we had a high certainty that that was the case. Still, these sources of uncertainty in our analysis constitute a threat to the validity of our findings (further discussed in Section 6) that we mitigate by sharing our data (Section 10).

4.3 Findings

Table 4 summarizes our findings for the 44 experiments from papers that earned ACM badges. The columns under 'Improvements' contain the % of experiments exhibiting gains across the specification levels (i.e., $PA \rightarrow FA$ means improvement from partially addressed in the paper to fully addressed when accounting for the materials in the repository), while the columns under Constant show the aspects of the experiments that remained unchanged.

Overall, we find that considering the artifact consistently improves the specification of some portions of the experiments but not others. The improvement is particularly noticeable in the variable identification step (S2) where many experiments that were Partially Addressed (PA) become Fully Addressed (FA). More specifically, the DL algorithm, model and training hyperparameters and the training data become fully addressed in 87%, 95%, 86% and 94% of the experiments, respectively[4]. The model parameters (also part of S2) show a modest 9% gain caused by the artifact for just one of the papers ([AP5]). Under operationalization (S3), the response variables also improve, becoming full for 95% of the experiments, while factors and treatments show some improvement for 59% of the experiments but still remains partially addressed for 84% of the experiments. These operationalization improvements were also expected as the code must assign values to the independent variables and measure the dependent variables to assess the experimental outcome. The rest of the aspects, which are more closely associated with the experimental design and analysis than the implementation, showed slight or no improvement. The instrumentation showed an improvement for 14% of the experiments, test set characteristics for 7%, descriptive statistics for 2%, and research hypotheses, choice of design, inferential statistics, and validity threats showed no improvement. In summary, considering the artifacts improved the aspects associated with S2, but the rest of weak spots identified in the papers remain.

Table 4: Characterization of (44) experiments that earned ACM Artifact Badges.

		Improvements				Constant		
Step	Aspect	PA→FA	M→PA	M→FA	PA→PA	M	PA	FA
S1	Research hypotheses	0%	0%	0%	0%	18%	0%	82%
S2	Model hyperparam.	68%	0%	7%	0%	0%	5%	20%
	Model parameters	0%	0%	9%	0%	82%	0%	9%
	DL algorithm	39%	0%	7%	0%	0%	13%	41%
	Training hyperparam.	59%	0%	7%	0%	0%	14%	20%
	Training data	7%	0%	7%	0%	2%	4%	80%
S3	Factors & treatments	0%	2%	5%	52%	0%	30%	11%
	Response variables	2%	3%	9%	0%	2%	0%	84%
S4	Choice of design	0%	0%	0%	0%	39%	59%	2%
	Instrumentation	2%	7%	0%	5%	2%	75%	9%
S5	Test set chars.	0%	0%	7%	0%	43%	30%	20%
S6	Descriptive statistics	0%	2%	0%	0%	64%	23%	11%
	Inferential statistics	0%	0%	0%	0%	95%	0%	5%
S7	Validity threats	0%	0%	0%	0%	27%	71%	2%

Our inspection also reveled several incomplete artifacts. We found that papers pointing to a piece of information that is not accessible in the artifact, either because it is missing from the artifact (e.g., paper [AP8] mentions that the artifact includes "all model information", but the model parameters are missing) or because it requires special permissions or has broken links (e.g., paper [AP39] contains dropbox links to training data that need permission).

More problematic, however, the inspection of the artifacts revealed many cases where *the experiments in the artifact and the experiments reported in the paper are inconsistent.* We found that most artifacts contained pieces of code representing variations of the experiments reported in the paper. This in itself is not a major source of concern as one may conjecture that these variations corresponded to different configurations explored during the investigation and development of the proposed techniques, configurations that perhaps were not properly labeled or cleaned from the shared code base. What is concerning, however, are the cases where the artifact does not have a single experiment variant that matches the experiment reported in the paper.

When comparing papers and artifact content, we find that 78% of the papers and 68% of the experiments show inconsistencies. For example, [AP29] mentions that the loss function used is binary cross-entropy, while the sigmoidal cross-entropy function is used in the artifact code. Paper [AP36] mentions the programs used as test sets for the paper, but the artifact contains a different set of

[4] The exceptions are two optimization experiments missing from the artifact's code (E4 [AP10] and E1 [AP41]), and 4 experiments in a paper that are missing the training code [AP5].

programs. Paper [AP15] makes a reference to grid search, which is absent in the artifact. Paper [AP40] mentions that the Adam optimizer is used, but the code also contains AdamW. Again, our analysis was conservative and the time dedicated to explore the artifacts was bounded, so it is reasonable to expect the inconsistencies found are likely an underestimate of the ones present. We also found artifacts that were at times inconsistent with themselves. For example, [AP39] provides generous supplementary information in the form of an online appendix that contains information related to experiments that are not reported in the paper, but these show the same inconsistencies with the code that the paper has regarding model hyperparameters and training data. Similarly, [AP41] does not mention in the paper the number of epochs used, and there are two values for it in the configuration file contained in the artifact.

It is important to emphasize that the analysis of the artifacts provides further evidence that the limitations we have identified in these experiments go beyond under-reporting problems. The lack of specificity in fundamental experimental design and implementation details reflect deficiencies that can have severe implications for the findings. We delve into these implications next.

5 IMPLICATIONS

The previous sections characterized the degree of under-specification in DL experiments to address SE problems when considering papers and artifacts. We found that the most affected experimental aspects are the analysis and interpretation of results, the design, and the operationalization of factors and treatments. In this section we answer **RQ3** by deriving the implications of under-specifying those aspects from the perspective of conclusion, internal, and construct validity of the experimental findings [27].

5.1 Is there a Relationship between the Response Variable and the Factor(s)? (Conclusion Validity)

The experiments assessed include 3 types of analysis to determine if there is a relation between the factors and the response variable.

We have found that 56% of the reviewed experiments resort to **comparing single data points**. This is problematic because it assumes that a single observation on the effect of the treatment will be a good estimate of the mean effect of that treatment, basically ignoring fluctuations due to experimental errors (this will be further discussed in Section 5.2). For example, [AP27] proposes a DNN that given as input a set of *may* links between communicating objects in two Android applications, outputs the probability that such links exists. The proposed approach is compared against 3 simpler DNN architectures as baselines. Based on the comparison of the values obtained from the test set for the four treatments, the paper concludes that the best option is the proposed model (the most complex one), with response variables values of 0.931 (F1), 0.991 (AUC) and 0.992 (Kruskal's γ). However, the results of the second best option are 0.920, 0.988 and 0.989 respectively. Note that a mere standard deviation of 0.0155, 0.0045 and 0.0045 in the response variables (assuming a sample size of 30) will invalidate the conclusion. This reflects a known weaknesses with single point comparisons underlined by a problem with the design of this experiment, which does not control, for example, for random sources of variation that would have required multiple runs and hence resulted in multiple values to perform a statistical comparison that accounts for variability. A variant of this problem is manifested in [AP55], which proposes a DNN that receives a code function as input and predicts whether it is vulnerable. The paper computes the performance of three techniques over multiple Android applications in terms of precision, recall, F-measure, and AUC. However, it then resorts to count the number of projects in which each technique has shown better results and compares those single values losing an opportunity to perform a more meaningful comparison.

We find that 31% of the experiments perform a **comparison of means**. This is stronger than using single data points, but still insufficient to guarantee that the differences found in the sample can be extrapolated to the population the sample represents. For example, [AP13] proposes a DNN that takes as input color pictures of source code files to predict whether they contain a fault. It uses 10 test sets corresponding to open source projects to assess the proposed approach against 4 existing techniques as per their mean F-measure for the different projects, and concludes that "the proposed DTL-DP shows significant improvements on the state of the art in cross-project defect prediction". Yet, there is no analysis that considers the variability observed on the collected measures, even though the F1 values showed large variability. To better understand the implications of this oversight we perform a statistical analysis with the data reported in Table 4 of the paper. Let's assume that the statistical null hypothesis (H0) is: "There is no difference in F-measure between the different approaches examined", and that the design is a 1-factor 5-levels experiment (inferred from the design description). The 1-way repeated measures ANOVA shows that we can reject the null hypothesis ($p<0.01$). The follow-up Bonferroni multiple comparisons test shows that the proposed approach has a better performance than three of the competing ones, but similar to one of them (DBN-CP, Cohen's d=0.3). This example illustrates that relationships identified through means may not necessarily be generalizable to the population.

Only 13% of the papers we reviewed identify the potential relationship through **inferential statistics**, meaning that the obtained results can be generalized from the experiment sample to the population it represents. For example, [AP53] proposes a DNN that given as input a code snippet that needs to be logged, suggests which variables should be logged. Their proposed approach is compared against 5 baselines for 9 different projects in terms of accuracy, mean reciprocal rank and mean average precision. Data is analyzed with a Wilcoxon signed-rank test (considering the 9 scores, one per project), and Cliff's Delta effect size is computed. In all cases, the improvement of the proposed approach is statistically significant, with a large effect size.

5.2 Is the Relationship Causal? (Internal Validity)

If a causal relationship exists, the improvement observed in the response variable measured (effect) can be attributed to the application of the technique (cause) being used and not due to other variables. In an experiment, the extent to which extraneous variables are accounted for in the design will define the strength of the

Table 5: Extraneous variables and how to deal with them

	Characteristics			Mechanism
Case	Known	Measurable	Controllable	
I	No	-	-	Randomization
II	Yes	No	-	Case I + Replication
III	Yes	Yes	No	Case II + Statistical adjustment
IV	Yes	Yes	Partially	Case III + Blocking
V	Yes	Yes	Yes	Case IV + Held-constant Incorporate as factor

causality link [15]. Table 5 shows some of the established recommended mechanisms to deal with extraneous variables [2, 15, 20, 32], which depend on the nature of the extraneous variable being controlled. For example, when the variable is known, measurable, and controllable, then we can address it either holding it constant or by incorporating it an experimental factor (e.g. dataset); and when the variable is known and measurable but not controllable we can use blocking to control its impact (e.g. random training/test split). Such mechanisms naturally apply to DL.

However, DL systems can be particularly challenging in that they have variables that use sources of randomness to improve the performance of the model [9]. In some cases, these variables are easy to identify and set (e.g. random weights initialization, batch size), in others they are easy to identify but difficult to anticipate their impact (e.g., data shuffling, dropout), and in other cases they are not even easily identifiable (e.g. more obscure options of core libraries).[5] Traditionally, the ML community has focused on classical notions of variance associated to the dataset variables, mostly ignoring the other types [5]. We now analyze whether such trend also applies to the DNN4SE experiments we analyzed. Since all experiments we have studied neither explicitly analyze the sources of randomness present in the experiment, discussing how they have incorporated them into the design, nor provide the experimental design and its rationale in the paper, the results presented here are deduced from the papers.

Our findings confirm that most experiments (62%) acknowledge the **classical** ML random variables related to the dataset. For example papers [AP46], [AP14], and [AP12] use several test sets. While paper [AP50] uses k-fold-cross-validation. However, this leaves free other sources of randomness. We also find that some experiments (38%) neglect to mention **any kind of source of randomness**, approaching their experimental design with the assumption that all variables can be held constant. All these papers train the DL algorithm once, measuring the response variable(s) for the test set. An example is [AP27] mentioned in the previous section and also [AP4] which proposes a DNN that automatically applies code changes implemented by developers during pull requests (PRs). None of the papers we analyzed deal with **random variables extrinsic to the dataset**. One example of this deficiency is how all experiments train the DNN only once for a given combination of hyperparameters. For example, in the optimization experiment reported in [AP10], Xavier initialization of parameters is used. However, since the DNN is trained just once, it is impossible to know if the best configuration is due to the combination of levels of factors or just a fortuitous (random) selection of initial weights.

It is important to note that all previous instances deal with *known* sources of randomness (cases II-V from Table 5). However, there might be *unknown* sources of randomness in an experiment (case I). The ML community has not fully acknowledged the existence of these variables, but it would be valuable for the experiments designs to safeguard against them. These variables are typically addressed by randomly assigning the order in which the experimental runs will take place. Imagine a situation where caching is in effect for the non-initial runs. If the runs are not randomly executed, and there is not enough of them, the first runs could behave differently from the rest. If we are comparing 2 DNNs and we plan all the runs for one of them first, this could be affecting the results.

5.3 Does the (Cause) Operationalization Accurately Represent its Construct? (Construct Validity)

A construct validity is an assessment of how well researchers translate their ideas into specific factors and treatments, and response variables [32]. Since the experiments we have analyzed operationalize well their response variables (76% fully address it), we will focus on factors and treatments.

The positive news is that only 4% of experiments have a **definition of factors that is incomplete**. This is the case for several hyperparameter optimization experiments, which are often not fully acknowledged in the papers. For example, in [AP25], the hyperparameters fine-tuning optimization experiment is mostly absent. The paper briefly mentions the range of hyperparameters, and gives some examples, but the listing is not exhaustive so in the end it is not known what factors were explored.

On the negative side, 82% of experiments define their factors properly, but their **treatment definitions are incomplete**. For example, the optimization experiment in [AP16] mentions that the hyperparemeters to be fine-tuned are embedding size, number of hidden states, batch size, maximum number of iterations, optimizer, learning rate, beam size and lambda. However, it does not specify the range of values that have been explored. In [AP8], the regularization term, the number of iterations, or the topology of the proposed DNN are not reported. In the experiment in [AP13], the treatments are defined at the architectural level (a deep adaptation network is compared against a deep belief network, a LSTM, and a CNN). However, specific implementations of these architectures are being compared, overlooking the fact that other non-specified variables like the model hyperparameters, the DL algorithm or the data representation could be the underlying causes for the performance gain, and not the architecture.

This issue gets magnified as the limitations propagate across papers. For example, [AP25] has an ambitious agenda to compare the proposed technique against three other state-of-the-art approaches, but none of them are easily and reliably available. For one of them, the stable version in Github is available, but it may be different from the one in the paper (according to our results from Section 4). For another, the authors of the paper had to resort to reimplement the approach following the original paper where it is proposed, which may implement a different technique. For the third one, the performance numbers reported in the original paper are used, but even if the training has ben imitated, those may have suffered from

[5] For a detailed analysis see [23, 30, 33].

extraneous variables that are unstated. The limited availability of high-quality artifacts remains an ongoing challenge.

Finally, 14% of experiments had all **factors and treatments fully operationalized**. For example, paper [AP14] specifies that the factors are: Word2vec vector length (with values 100, 50, 120), learning rate (with values 0.001, 0.005, 0.01) and epoch size (with values 100, 200, 300).

5.4 Characterization of Experiments and Implications

We now proceed to analyze the distribution of experiments' implications to better understand how often they occur and what combinations are the most common. We utilize the parallel categories diagram in Figure 3 to facilitate the exposition. The sets of nodes being considered are associated with the three implication types analyzed in the previous sections. That is, for relationship exploration (conclusion validity) we consider comparisons among single values, means, and inferential; for causality (internal validity) we consider when none, classical, and other sources of randomness are controlled; for construct validity we consider when none, factors, or both treatments and factors are defined. In the figure, the nodes on the left correspond to comparisons, the ones on the center to causality, and the ones on the right to constructs issues.

We have not found any experiment that properly addresses all types of validity threats discussed under the implications. The best conducted experiments, 11% of the ones examined, perform inferential analysis, control classical sources of randomness, and specify factors. The majority of experiments (61%), however, have at least one critical issue (either compare single values, do not control any single source of randomness, or specify neither factors nor treatments). Even though most experiments specify at least factors, they perform comparisons based on single values and/or do not control any variables (45%).

6 VALIDITY THREATS

We briefly discuss the main limitations arising from the scope, design, and implementation of our study [17].

The **external validity** of our study is determined by the *eligibility criteria* we chose. We concentrated on the 'top' conferences and journals (Section 3.1) with the expectation that the findings would constitute an upper bound for the average quality of experiments appearing in other venues. The time period covered (2018–2021) allowed us to determine the status of recent research in the topic (2022 papers were not examined as the search was performed in 1/2022, and the analysis was conducted during 2022). Given the number of relevant papers in 2021, we randomly selected a subset to be examined. The same quality-driven and cost-control reasoning applies for us to target the artifacts with an ACM badge (Section 4.1).

Internal validity. Our *source of information* (Section 3.1) to identify the papers—SCOPUS—included the chosen venues in the time period covered; this reduced the possibility of omitting potentially relevant studies. The *search strategy* we followed is also repeatable. The paper *selection process* required for each paper to be examined by one of the two authors; however, joint checks and discussion of papers that did not fit in existing filtering criteria reduced the chances of missing potentially relevant papers.

Due to the high data extraction costs from papers, the *data collection process* (Section 3.2 and Section 4.2) considered all 17 ICSE papers 2018–2020, which were jointly examined by both authors to ensure that the collection strategies and results were aligned, while the remaining papers were examined by just one author. Again, joint checks and discussions of studies that did not fit the schema, introduced new ML elements, or had ambiguous specifications reduced possible researcher bias. Finally, doing a *critical appraisal of individual sources of evidence*, we note that analyzing papers was challenging given the diversity of presentation styles, the number and complexity of the variables to check, and the increasing richness of the DNN domain. Furthermore, analyzing artifacts was a consistently arduous re-engineering process. The nature and magnitude of these analyses may have introduced errors in our measures. We attempted to control these internal threats by sharing all the intermediate results of the study with the community.

Construct validity. The characterization schema (*data items* defined in Section 3.2) was specifically developed for this research. We created it starting from the steps of the experimental process and the aspects of the experiments that have to be covered during each step. Beginning with the generic definitions given by the experimental software engineering literature, the authors iteratively and systematically tailored it to the DL domain. We believe this provides a reasonable operationalization, one that is transparent as well for others to assess, refine, and reuse. A simpler assessment would just analyze the validity threats reported by the papers. However, the description of threats is typically ad-hoc and often incomplete [3, 29]. For this reason, we decided to assess the validity of the experiments from their description (and code artifacts in some cases). The *syntheses of results* made in Section 3.3 and Section 4.3 allow identifying the validity level of the results reported in the studies.

7 RECOMMENDATIONS

Failing to address the limitations we identified in the state of the practice could undermine much of the research devoted to DNN4SE. Thus, we propose three actionable recommendations that have the potential to address most of the pressing concerns we discovered. These recommendations correspond to well-established practices in experimental studies adapted for the DNN4SE domain where authors may not be aware of the impact of not following them. As researchers become aware of these practices and adopt them, the quality of the studies will likely improve, as authors will start incorporating them in their papers, and reviewers will start considering them into their reviewing process.

Rec#1: Perform Multiple DNN Training Runs to Control for Randomness. Experiments must strive to control the randomness of the DNN training process. This process can introduce various sources of randomness, and a fundamental one is the random data selection and shuffling that occurs iteratively to compute the gradient over the DNN, which means that the resulting values may change over different runs. Yet, none of the papers reported to make multiple training runs to control for this intrinsic source of DNN randomness. This raises questions about whether most results are caused by just a fortuitous or unfortunate sample selection while searching for the gradient. There are other sources of randomness

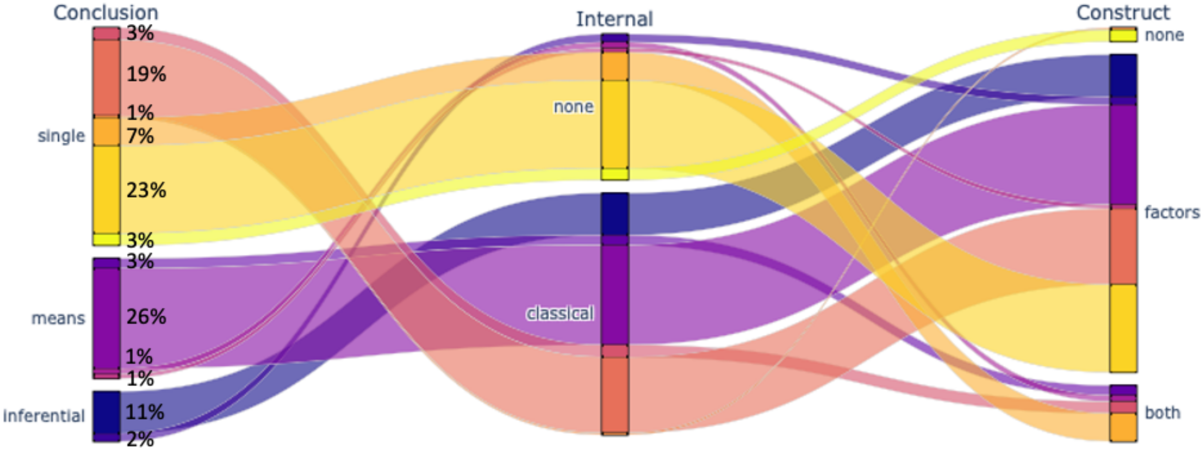

Figure 3: Distribution of experiments' implications.

to consider (e.g., the initialization weights, the data splitting) but based on our findings we argue that simply conducting multiple runs of the DNN training process would enable the control of a sizable portion of the randomness we observed. Furthermore, given the size of the experiments we analyzed and the magnitude of free computing resources available, we found no compelling argument for not running an experiment multiple times to account for the randomness in the DNN training process. This recommendation could benefit almost all experiments we analyzed.

Rec#2: Compute Statistics Over Multiple Runs and Data Partitions. Experiments are meant to establish relationships between factors and response variables. Our analysis, however, found that 56% of experiments identified a relationship based on single observations. Some of those studies had multiple observations gathered over multiple units of analysis (i.e., projects, releases, apps); in such cases it is difficult to justify why select a single data point to compare treatments. For the rest of the cases, however, there are plenty of opportunities to collect multiple observations. For instance, recommendation Rec#1 for conducting multiple training runs will enable the collection of multiple observations. A second easily accessible source of observations for most of the papers we analyzed are the multiple partitions of the dataset used as part of the training. Given the number of sources of randomness in DNN training rendering multiple observations, we find no compelling reason not to require at least a comparison of means from such observations, and if there are enough observations computer inferential statistics to judge whether the results generalize from the sample to the population. This recommendations could benefit between 56% and 87% of the experiments we analyzed.

Rec#3: Specify DNN Training Parameters Treatment Space and Check for Paper Consistency Against Artifact. We have already described the large DNN configuration space and how different instantiations of it can dramatically impact the performance of DNN4SE techniques. Yet, most papers fail to provide a specification of even some of the basic DNN parameters in that space. That lack can be mitigated by artifacts with code implementing the DNN training process. However, re-engineering the experimental design from such artifacts puts an undue load on the reader and it is fault prone (we have done 44 of them to attest to that!). Furthermore, some experiments are often missing in the artifacts and it is common to find inconsistencies between papers and their corresponding artifacts. We recommend that papers shall provide a tabular description of the DNN configuration space explored for each experiment (as we have done for each experiment analyzed – see Appendix for samples). We also recommend for the adoption of ML experiment management tools (e.g., jupiter, mlflow, DVC) to track the DNN experiments, how they evolve, and also to control how they are shared in the papers and in the artifacts to facilitate the detection of inconsistencies. This recommendation could benefit 86% of the analyzed experiments.

Deploying Mediums. The previous recommendations can be implemented through different mediums. They can go directly to authors as part of a call for papers checklists [1, 18, 22], be integrated as a part of the artifact verification process, be provided to reviewers to help them judge a paper soundness and verifiability, become part of broader guidelines such as the recently introduced empirical processes guidelines [24], or serve as instructions for newcomers to the area. Given the increasing number of DNN4SE papers (the trend from Table 1 indicates that they are likely to become a dominant research thrust in the venues we studied for years to come) and the pitfalls we observed and quantified, pursuing several of these mediums seems warranted.

Periodic Checks of DNN4SE Paper Experiments. Quantifications and reflections of where we stand as a community, like we have completed here, are an essential measurement stick to judge progress. Given the issues we found and the nature of DNN4SE that includes rapidly evolving technology, researchers, and methodologies, follow up checks seem required to at least determine the trends over the concerns. To reduce the cost of such checks, the framework we have defined in our evaluation could be reused and a smaller sample of the yearly experiments could be analyzed.

8 RELATED WORK

8.1 In the SE Field

A series of studies have analyzed the **quality of SE experiments**. Table 6 shows the number of papers examined, the period covered, whether all or just a subset of papers are examined, the population

Table 6: Studies analyzing the quality of SE experiments.

Study	Size	Period	Population	Sample	Type	Aspect
[7]	103	93–02	Selected Js&Cs	E	Both	Statistical power
[12]	103	93–02	Selected Js&Cs	E	Both	Theory
[14]	150	02–12	All	R	Both	Researcher and publication bias
[16]	103	93–02	Selected Js&Cs	E	Both	Effect size
[25]	51	06–15	ICSE	R	Both	Correctness of analysis
[28]	49	00-18	ML4DP[6]	E	Technology	Statistical errors
[29]	83	15–19	Selected Js	E	Human	Construct validity

the papers belong to (selected journals and/or conferences, particular conferences, or all), the sampling performed (Exhaustive or Random), the type of experiments included (human-oriented, technology-oriented, or both), and the quality aspect under examination for each of those studies. These studies differ primarily from ours in that they focus mostly on a single quality aspect at a high-level of abstraction that is common across multiple software engineering domains, while we performed a deeper specialized analysis on more quality aspects but focused on a single domain.

To improve the quality of experiments, the SE community has developed an extensive body of knowledge, some of which has resulted in **guidelines** for running and reporting experiments. Some of the guidelines are **general** enough to apply to any SE experiment [13, 15, 32], and therefore served as a starting point to characterize our experiments (Section 3). However, such general guidelines do not address the specific challenges associated with experiments in the DNN4SE domain which is rapidly evolving and acquiring a critical momentum in the SE community. Other guidelines are **specific**. For example, there are domain-specific guidelines for the analysis of randomized testing algorithms [4], for addressing the diversity of the projects from which to get the dataset to be used in MSR studies [21], and there are guidelines that are specific to conducting human-based experiments [24, Experiments] or to perform benchmarking [24, Benchmarking]. Again, although helpful, they are not addressing specific concerns raised when conducting experiments in the DNN domain. This is the first paper that characterizes the state of the practice in DNN4SE experimentation.

8.2 In the AI Field

Similarly, the reproducibility guidelines from the AI community [1, 18, 22] are also relevant and applicable to experiments conducted in the DNN4SE domain since they cover the possibility of performing experiments. However, we have found these guidelines to be limited in several ways. First, they are too descriptive and general as they try to encompass a broad range of models that go beyond DNNs (e.g., decision trees, random forests, support vector machines, etc.), and they address the possibility of making theoretical contributions (which is not of interest in SE when using DNNs). Second, while they are broad in terms of the models covered, they seem too narrow in other aspects. For instance, they do not allow for accommodating experiments beyond those involved in the development of the DNN, such as those conducted when comparing a new approach against state-of-the-art models. Third, they lack a comprehensive description of the experimental design (a crucial aspect, as it is the only feature of experiments that allows them to identify causality). Finally, they strictly focus on enabling the reproduction of the results obtained with the ML learning model presented. Concerning the reported experiments, this implies using the artifacts made available by the authors. As a consequence, they lack sufficient details to enable the replication of the experiments using different contexts or datasets.

9 CONCLUSIONS

The SE community is increasingly developing techniques based on DNNs to solve software engineering problems. Performing experiments to assess such techniques is challenging given DNNs' inherent complexity involving many subtle and interdependent training variables, sources of randomness, and rapid technological evolution. Our examination of 194 experiments in 55 papers is the first to quantify these challenges. We find that 87% of experiments are missing inferential statistics and 56% are missing even basic descriptive statistics, 4% are not stating the experimental factors and 82% only do so partially, and 38% do not specify even the basic elements of the experimental design to control any source of randomness while the rest only control for the classical sources of randomness. These findings' trends mildly improve when artifacts are provided as part of such experiments, and what is more concerning is that that most artifacts are not fully consistent with their corresponding paper.

These findings are problematic because they imply that: 1) there is weak support to determine that there is a true relationship between independent and dependent variables that did not take place by happenstance, 2) there is limited control over the space of DNN relevant variables, which can render a relationship between dependent variables and treatments that may not be causal but rather correlational, and 3) there is a lack of specificity in terms of what are the DNN variables and their values utilized in the experiments to define the treatments being applied, which makes it unclear whether the techniques designed are the ones being assessed. We have proposed a series of actionable recommendations addressing the most critical findings we uncovered and will push forward to have them become a part of our community practices.

10 DATA AVAILABILITY

The data of our analyses is available at https://github.com/GRISE-UPM/Pitfalls_Experiments_DNN4SE

ACKNOWLEDGMENTS

The authors gratefully acknowledge the Universidad Politécnica de Madrid (www.upm.es) for providing computing resources on Magerit Supercomputer, the reviewers for their feedback, and the authors of the papers and artifacts analyzed in the paper.

REFERENCES

[1] AAAI 2022. *The 37th AAAI Conference on Artificial Intelligence Reproducibility Checklist.* accessed August 26, 2022.

[2] Naomi Altman and Martin Krzywinski. 2021. Sources of variation. *Nature Methods* 12 (2021), 5–6.

[3] Apostolos Ampatzoglou, Stamatia Bibi, Paris Avgeriou, Marijn Verbeek, and Alexander Chatzigeorgiou. 2019. Identifying, categorizing and mitigating threats

[6] Machine Learning for Defect Prediction papers.

to validity in software engineering secondary studies. *Information and Software Technology* 106 (2019), 201–230.

[4] Andrea Arcuri and Lionel Briand. 2011. A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE'11, May.* 1–10.

[5] Xavier Bouthillier, Pierre Delaunay, Mirko Bronzi, Assya Trofimov, Brennan Nichyporuk, Justin Szeto, Nazanin Mohammadi Sepahvand, Edward Raff, Kanika Madan, Vikram Voleti, Samira Ebrahimi Kahou, Vincent Michalski, Tal Arbel, Chris Pal, Gael Varoquaux, and Pascal Vincent. 2021. Accounting for Variance in Machine Learning Benchmarks. In *Proceedings of Machine Learning and Systems.* 747–769.

[6] Prem Devanbu, Matthew Dwyer, Sebastian Elbaum, Michael Lowry, Kevin Moran, Denys Poshyvanyk, Baishakhi Ray, Rishabh Singh, and Xiangyu Zhang. 2020. Deep Learning & Software Engineering: State of Research and Future Directions. arXiv:cs.SE/2009.08525

[7] Tore Dybå, Vigdis By Kampenes, and Dag I. K. Sjøberg. 2006. A systematic review of statistical power in software engineering experiments. *Information and Software Technology* 48, 8 (2006), 745–755.

[8] Association for Computing Machinery. 2020. *Artifact Review and Badging.* https://www.acm.org/publications/policies/artifact-review-and-badging-current

[9] Claudio Gallicchio, José Martín-Guerrero, Alessio Micheli, and Emilio Olivas. 2017. Randomized Machine Learning Approaches: Recent Developments and Challenges. In *Proceedings of the 25th European Symposium on Artificial Neural Networks (ESANN).*

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

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning.* The MIT Press.

[12] Jo Erskine Hannay, Dag I. K. Sjøberg, and Tore Dybå. 2007. A Systematic Review of Theory Use in Software Engineering Experiments. *IEEE Trans. Software Eng.* 33, 2 (2007), 87–107.

[13] Andreas Jedlitschka, Marcus Ciolkowski, and Dietmar Pfahl. 2008. Reporting Experiments in Software Engineering. In *Guide to Advanced Empirical Software Engineering*, Forrest Shull, Janice Singer, and Dag I.K. Sjøberg (Eds.). Springer, Chapter 8, 201–228.

[14] Magne Jørgensen, Tore Dybå, Knut Liestøl, and Dag I. K. Sjøberg. 2016. Incorrect results in software engineering experiments: How to improve research practices. *J. Syst. Softw.* 116 (2016), 133–145.

[15] Natalia Juristo and Ana M Moreno. 2011. *Basics of software engineering experimentation.* Springer Science & Business Media.

[16] Vigdis By Kampenes, Tore Dybå, Jo Erskine Hannay, and Dag I. K. Sjøberg. 2007. A systematic review of effect size in software engineering experiments. *Inf. Softw. Technol.* 49, 11-12 (2007), 1073–1086.

[17] B. Kitchenham, L. Madeyski, and D. Budgen. Early Access. SEGRESS: Software Engineering Guidelines for REporting Secondary Studies. *IEEE Transactions on Software Engineering* (Early Access). https://doi.org/10.1109/TSE.2022.3174092

[18] machine 2020. *The Machine Learning Reproducibility Checklist v2.0.* accessed August 26, 2022.

[19] Tom Mitchell. 2019. *Machine Learning.* McGraw-Hill Education.

[20] Douglas C Montgomery. 2019. *Design and Analysis of Experiments.* John Wiley & Sons Inc.

[21] Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. 2013. Diversity in software engineering research. In *9th joint meeting on foundations of software engineering.* 466–476.

[22] neurips 2022. *The 36th Conference on Neural Information Processing Systems PaperChecklist Guidelines.* accessed August 26, 2022.

[23] Hung Viet Pham, Shangshu Qian, Jiannan Wang, Thibaud Lutellier, Jonathan Rosenthal, Lin Tan, Yaoliang Yu, and Nachiappan Nagappan. 2020. Problems and Opportunities in Training Deep Learning Software Systems: An Analysis of Variance. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering.* 771–783.

[24] Paul Ralph, Nauman bin Ali, Sebastian Baltes, Domenico Bianculli, Jessica Diaz, Yvonne Dittrich, Neil Ernst, Michael Felderer, Robert Feldt, Antonio Filieri, Breno Bernard Nicolau de França, Carlo Alberto Furia, Greg Gay, Nicolas Gold, Daniel Graziotin, Pinjia He, Rashina Hoda, Natalia Juristo, Barbara Kitchenham, Valentina Lenarduzzi, Jorge Martínez, Jorge Melegati, Daniel Mendez, Tim Menzies, Jefferson Molleri, Dietmar Pfahl, Romain Robbes, Daniel Russo, Nyyti Saarimäki, Federica Sarro, Davide Taibi, Janet Siegmund, Diomidis Spinellis, Miroslaw Staron, Klaas Stol, Margaret-Anne Storey, Damian Tamburri, Marco Torchiano, Christoph Treude, Burak Turhan, Xiaofeng Wang, and Sira Vegas. 2021. Empirical Standards for Software Engineering Research. arXiv:cs.SE/2010.03525v2

[25] Rolando Reyes, Óscar Dieste, Efraín R. Fonseca, and Natalia Juristo. 2018. Statistical errors in software engineering experiments: a preliminary literature review. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE).* 1195–1206.

[26] Arthur Samuel. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of research and development* 3, 3 (1959), 210–229.

[27] William R. Shadish, Thomas D. Cook, and Donald T. Campbell. 2002. *Experimental and Quasi-Experimental Designs for Generalized Causal Inference.* Wadsworth, Cengage Learning.

[28] Martin Shepperd, Yuchen Guo, Ning Li, Mahir Arzoky, Andrea Capiluppi, Steve Counsell, Giuseppe Destefanis, Stephen Swift, Allan Tucker, and Leila Yousefi. 2019. The Prevalence of Errors in Machine Learning Experiments. In *Intelligent Data Engineering and Automated Learning – IDEAL 2019*, Hujun Yin, David Camacho, Peter Tino, Antonio J. Tallón-Ballesteros, Ronaldo Menezes, and Richard Allmendinger (Eds.). 102–109.

[29] Dag I.K. Sjoberg and Gunnar R. Bergersen. 2022. Construct Validity in Software Engineering. *IEEE Transactions on Software Engineering* (2022), Early Access. https://doi.org/10.1109/TSE.2022.3176725

[30] Cecilia Summers and Michael J. Dinneen. 2021. Nondeterminism and Instability in Neural Network Optimization. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event.* 9913–9922.

[31] Christopher S. Timperley, Lauren Herckis, Claire Le Goues, and Michael Hilton. 2021. Understanding and Improving Artifact Sharing in Software Engineering Research. *Empirical Software Engineering* 26, 4 (2021).

[32] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering.* Springer Science & Business Media.

[33] Donglin Zhuang, Xingyao Zhang, Shuaiwen Leon Song, and Sara Hooker. 2022. Randomness In Neural Network Training: Characterizing The Impact of Tooling. In *Proceedings of the 5th Conference on Machine Learning and Systems.*

ANALYZED PAPERS

[AP1] Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. From ui design image to gui skeleton: A neural machine translator to bootstrap mobile gui implementation. In *Proceedings of the 40th ICSE*, 2018.

[AP2] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. Deep code search. In *Proceedings of the 40th ICSE*, 2018.

[AP3] Kui Liu, Dongsun Kim, Tegawendé F. Bissyandé, Taeyoung Kim, Kisub Kim, Anil Koyuncu, Suntae Kim, and Yves Le Traon. Learning to spot and refactor inconsistent method names. In *Proceedings of the 41st ICSE*, 2019.

[AP4] Michele Tufano, Jevgenija Pantiuchina, Cody Watson, Gabriele Bavota, and Denys Poshyvanyk. On learning meaningful code changes via neural machine translation. In *Proceedings of the 41st ICSE*, 2019.

[AP5] Rabee Sohail Malik, Jibesh Patra, and Michael Pradel. Nl2type: Inferring javascript function types from natural language information. In *Proceedings of the 41st ICSE*, 2019.

[AP6] Dehai Zhao, Zhenchang Xing, Chunyang Chen, Xin Xia, and Guoqiang Li. Actionnet: Vision-based workflow action recognition from programming screencasts. In *Proceedings of the 41st ICSE*, 2019.

[AP7] Facundo Molina, Renzo Degiovanni, Pablo Ponzio, Germán Regis, Nazareno Aguirre, and Marcelo Frias. Training binary classifiers as data structure invariants. In *Proceedings of the 41st ICSE*, 2019.

[AP8] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. A novel neural source code representation based on abstract syntax tree. In *Proceedings of the 41st ICSE*, 2019.

[AP9] Alexander LeClair, Siyuan Jiang, and Collin McMillan. A neural model for generating natural language summaries of program subroutines. In *Proceedings of the 41st ICSE*, 2019.

[AP10] Huong Ha and Hongyu Zhang. Deepperf: Performance prediction for configurable software with deep sparse neural network. In *Proceedings of the 41st ICSE*, 2019.

[AP11] Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xiwei Xu, Liming Zhu, Guoqiang Li, and Jinshui Wang. Unblind your apps: Predicting natural-language labels for mobile gui components by deep learning. In *Proceedings of the 42nd ICSE*, 2020.

[AP12] Thong Hoang, Hong Jin Kang, David Lo, and Julia Lawall. Cc2vec: Distributed representations of code changes. In *Proceedings of the 42nd ICSE*, 2020.

[AP13] Jinyin Chen, Keke Hu, Yue Yu, Zhuangzhi Chen, Qi Xuan, Yi Liu, and Vladimir Filkov. Software visualization and deep transfer learning for effective software defect prediction. In *Proceedings of the 42nd ICSE*, 2020.

[AP14] Yi Li, Shaohua Wang, and Tien N. Nguyen. Dlfix: Context-based code transformation learning for automated program repair. In *Proceedings of the 42nd ICSE*, 2020.

[AP15] Lin Shi, Mingzhe Xing, Mingyang Li, Yawen Wang, Shoubin Li, and Qing Wang. Detection of hidden feature requests from massive chat messages via deep siamese network. In *Proceedings of the 42nd ICSE*, 2020.

[AP16] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, and Xudong Liu. Retrieval-based neural source code summarization. In *Proceedings of the 42nd ICSE*, 2020.

[AP17] Cody Watson, Michele Tufano, Kevin Moran, Gabriele Bavota, and Denys Poshyvanyk. On learning meaningful assert statements for unit test cases. In *Proceedings of the 42nd ICSE*, 2020.

[AP18] Preetha Chatterjee, Kostadin Damevski, and Lori Pollock. Automatic extraction of opinion-based q&a from online developer chats. In *Proceedings of the 43rd ICSE*, 2021.

[AP19] Marlo Haering, Christoph Stanik, and Walid Maalej. Automatically matching bug reports with related app reviews. In *Proceedings of the 43rd ICSE*, 2021.

[AP20] Nan Jiang, Thibaud Lutellier, and Lin Tan. Cure: Code-aware neural machine translation for automatic program repair. In *Proceedings of the 43rd ICSE*, 2021.

[AP21] Kaibo Cao, Chunyang Chen, Sebastian Baltes, Christoph Treude, and Xiang Chen. Automated query reformulation for efficient search based on query logs from stack overflow. In *Proceedings of the 43rd ICSE*, 2021.

[AP22] Yi Li, Shaohua Wang, and Tien N. Nguyen. Fault localization with code coverage representation learning. In *Proceedings of the 43rd ICSE*, 2021.

[AP23] Seohyun Kim, Jinman Zhao, Yuchi Tian, and Satish Chandra. Code prediction by feeding trees to transformers. In *Proceedings of the 43rd ICSE*, 2021.

[AP24] Yi Li, Shaohua Wang, and Tien N. Nguyen. A context-based automated approach for method name consistency checking and suggestion. In *Proceedings of the 43rd ICSE*, 2021.

[AP25] Gang Zhao and Jeff Huang. Deepsim: Deep learning code functional similarity. In *Proceedings of the 26th ESEC/FSE*, 2018.

[AP26] Vincent J. Hellendoorn, Christian Bird, Earl T. Barr, and Miltiadis Allamanis. Deep learning type inference. In *Proceedings of the 26th ESEC/FSE*, 2018.

[AP27] Jinman Zhao, Aws Albarghouthi, Vaibhav Rastogi, Somesh Jha, and Damien Octeau. Neural-augmented static analysis of android communication. In *Proceedings of the 26th ESEC/FSE*, 2018.

[AP28] Thanh Nguyen, Ngoc Tran, Hung Phan, Trong Nguyen, Linh Truong, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen. Complementing global and local contexts in representing api descriptions to improve api retrieval tasks. In *Proceedings of the 26th ESEC/FSE*, 2018.

[AP29] Davide Fucci, Alireza Mollaalizadehbahnemiri, and Walid Maalej. On using machine learning to identify knowledge in api reference documentation. In *Proceedings of the 27th ESEC/FSE*, 2019.

[AP30] Yanju Chen, Ruben Martins, and Yu Feng. Maximal multi-layer specification synthesis. In *Proceedings of the 27th ESEC/FSE*, 2019.

[AP31] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 27th ESEC/FSE*, 2019.

[AP32] Zhenpeng Chen, Yanbin Cao, Xuan Lu, Qiaozhu Mei, and Xuanzhe Liu. Sentimoji: An emoji-powered learning approach for sentiment analysis in software engineering. In *Proceedings of the 27th ESEC/FSE*, 2019.

[AP33] Michael Pradel, Georgios Gousios, Jason Liu, and Satish Chandra. Typewriter: Neural type prediction with search-based validation. In *Proceedings of the 28th ESEC/FSE*, 2020.

[AP34] Yujun Chen, Xian Yang, Hang Dong, Xiaoting He, Hongyu Zhang, Qingwei Lin, Junjie Chen, Pu Zhao, Yu Kang, Feng Gao, Zhangwei Xu, and Dongmei Zhang. Identifying linked incidents in large-scale online service systems. In *Proceedings of the 28th ESEC/FSE*, 2020.

[AP35] Jaeseong Lee, Pengyu Nie, Junyi Jessy Li, and Milos Gligoric. On the naturalness of hardware descriptions. In *Proceedings of the 28th ESEC/FSE*, 2020.

[AP36] Dongdong She, Rahul Krishna, Lu Yan, Suman Jana, and Baishakhi Ray. Mtfuzz: Fuzzing with a multi-task neural network. In *Proceedings of the 28th ESEC/FSE*, 2020.

[AP37] Reyhaneh Jabbarvand, Forough Mehralian, and Sam Malek. Automated construction of energy test oracles for android. In *Proceedings of the 28th ESEC/FSE*, 2020.

[AP38] Jieshan Chen, Mulong Xie, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, and Guoqiang Li. Object detection for graphical user interface: Old fashioned or deep learning or a combination? In *Proceedings of the 28th ESEC/FSE*, 2020.

[AP39] Kexin Pei, Jonas Guan, Matthew Broughton, Zhongtian Chen, Songchen Yao, David Williams-King, Vikas Ummadisetty, Junfeng Yang, Baishakhi Ray, and Suman Jana. Stateformer: Fine-grained type recovery from binaries using generative state modeling. In *Proceedings of the 29th ESEC/FSE*, 2021.

[AP40] Qihao Zhu, Zeyu Sun, Yuan-an Xiao, Wenjie Zhang, Kang Yuan, Yingfei Xiong, and Lu Zhang. A syntax-guided edit decoder for neural program repair. In *Proceedings of the 29th ESEC/FSE*, 2021.

[AP41] Shangwen Wang, Ming Wen, Bo Lin, and Xiaoguang Mao. Lightweight global and local contexts guided method name recommendation with prior knowledge. In *Proceedings of the 29th ESEC/FSE*, 2021.

[AP42] Zhipeng Gao, Xin Xia, David Lo, John Grundy, and Thomas Zimmermann. Automating the removal of obsolete todo comments. In *Proceedings of the 29th ESEC/FSE*, 2021.

[AP43] Yi Li, Shaohua Wang, and Tien N. Nguyen. Vulnerability detection with fine-grained interpretations. In *Proceedings of the 29th ESEC/FSE*, 2021.

[AP44] Forough Mehralian, Navid Salehnamadi, and Sam Malek. Data-driven accessibility repair revisited: On the effectiveness of generating labels for icons in android apps. In *Proceedings of the 29th ESEC/FSE*, 2021.

[AP45] Yiling Lou, Qihao Zhu, Jinhao Dong, Xia Li, Zeyu Sun, Dan Hao, Lu Zhang, and Lingming Zhang. Boosting coverage-based fault localization via graph-based representation learning. In *Proceedings of the 29th ESEC/FSE*, 2021.

[AP46] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Trang Pham, Aditya Ghose, and Tim Menzies. A deep learning model for estimating story points. *IEEE Transactions on Software Engineering*, 45(7):637–656, 2019.

[AP47] Qiao Huang, Xin Xia, David Lo, and Gail C. Murphy. Automating intention mining. *IEEE Transactions on Software Engineering*, 46(10):1098–1119, 2020.

[AP48] Song Wang, Taiyue Liu, Jaechang Nam, and Lin Tan. Deep semantic feature learning for software defect prediction. *IEEE Transactions on Software Engineering*, 46(12):1267–1293, 2020.

[AP49] Kevin Moran, Carlos Bernal-Cárdenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk. Machine learning-based prototyping of graphical user interfaces for mobile apps. *IEEE Transactions on Software Engineering*, 46(2):196–221, 2020.

[AP50] Jian Gao, Yu Jiang, Zhe Liu, Xin Yang, Cong Wang, Xun Jiao, Zijiang Yang, and Jiaguang Sun. Semantic learning and emulation based cross-platform binary vulnerability seeker. *IEEE Transactions on Software Engineering*, 47(11):2575–2589, 2021.

[AP51] Suyu Ma, Zhenchang Xing, Chunyang Chen, Cheng Chen, Lizhen Qu, and Guoqiang Li. Easy-to-deploy api extraction by multi-level feature embedding and transfer learning. *IEEE Transactions on Software Engineering*, 47(10):2296–2311, 2021.

[AP52] Hui Liu, Jiahao Jin, Zhifeng Xu, Yanzhen Zou, Yifan Bu, and Lu Zhang. Deep learning based code smell detection. *IEEE Transactions on Software Engineering*, 47(9):1811–1837, 2021.

[AP53] Zhongxin Liu, Xin Xia, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. Which variables should i log? *IEEE Transactions on Software Engineering*, 47(9):2012–2031, 2021.

[AP54] Kui Liu, Dongsun Kim, Tegawendé F. Bissyandé, Shin Yoo, and Yves Le Traon. Mining fix patterns for findbugs violations. *IEEE Transactions on Software Engineering*, 47(1):165–188, 2021.

[AP55] Hoa Khanh Dam, Truyen Tran, Trang Pham, Shien Wee Ng, John Grundy, and Aditya Ghose. Automatic feature learning for predicting vulnerable software components. *IEEE Transactions on Software Engineering*, 47(1):67–85, 2021.