Software Engineering Meets Deep Learning: A Mapping Study

Fabio Ferreira
Center of Informatics
Federal Institute of the Southeast of Minas Gerais (IF Sudeste MG)
Barbacena, Brazil
fabio.ferreira@ifsudestemg.edu.br

Luciana Lourdes Silva
Department of Computing
Federal Institute of Minas Gerais (IFMG)
Ouro Branco, Brazil
luciana.lourdes.silva@ifmg.edu.br

Marco Tulio Valente
ASERG Group - Department of Computer Science - Federal University of Minas Gerais (UFMG)
Belo Horizonte, Brazil
mtov@dcc.ufmg.br

ABSTRACT

Deep Learning (DL) is being used nowadays in many traditional Software Engineering (SE) problems and tasks. However, since the renaissance of DL techniques is still very recent, we lack works that summarize and condense the most recent and relevant research conducted at the intersection of DL and SE. Therefore, in this paper, we describe the first results of a mapping study covering 81 papers about DL & SE. Our results confirm that DL is gaining momentum among SE researchers over the years and that the top-3 research problems tackled by the analyzed papers are documentation, defect prediction, and testing.

CCS CONCEPTS

• **Software and its engineering**; • **Software organization and properties**;

KEYWORDS

Software Engineering, Deep Learning

ACM Reference Format:
Fabio Ferreira, Luciana Lourdes Silva, and Marco Tulio Valente. 2021. Software Engineering Meets Deep Learning: A Mapping Study. In *The 36th ACM/SIGAPP Symposium on Applied Computing (SAC '21), March 22–26, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3412841.3442029

1 INTRODUCTION

Deep Learning (DL) applications are increasingly crucial in many areas, such as automatic text translation [1], image recognition [2], self-driving cars [3], and smart cities[4]. Moreover, various frameworks are available nowadays to facilitate the implementation of DL applications, such as TensorFlow[1] and PyTorch[2]. Recently, Software Engineering (SE) researchers are also starting to explore the application of DL in typical SE problems, such documentation [5–7], defect prediction [8–11], and testing [12–14].

[1]https://www.tensorflow.org
[2]https://pytorch.org

SAC '21, March 22–26, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8104-8/21/03…$15.00
https://doi.org/10.1145/3412841.3442029

However, since the cross-pollination between DL & SE is recent, we do not have a clear map of the research conducted by combining these two areas. This map can help other researchers interested in starting to work on the application of DL in SE. It can also help researchers that already work with DL & SE to have a clear picture of similar research in the area. Finally, mapping the research conducted in the intersection of DL & SE might help practitioners and industrial organizations better understand the problems, solutions, and opportunities in this area.

In this paper, we provide the results of an effort to review and summarize the most recent and relevant literature about DL & SE. To this purpose, we collect and analyze 81 papers recently published in major SE conferences and journals. We show the growth of the number of papers on DL & SE over the years. We also reveal the most common recent problems tackled by such papers and provide data on the most common DL techniques used by SE researchers. Next, we highlight papers that achieved the same results using only traditional Machine Learning-based techniques. Finally, we discuss the main drawbacks and strengths of using DL techniques to solve SE-related problems.

2 DEEP LEARNING IN A NUTSHELL

DL is a subfield of Machine Learning (ML) that relies on multiple layers of Neural Networks (NN) to model high-level representations [15]. Similar to traditional ML, DL techniques are suitable for classification, clustering, and regression problems. The key difference between traditional ML and DL techniques is that while in traditional ML approaches the features are handcrafted, in DL they are selected by neural networks automatically [16]. Currently, there are many types of NNs [15] and below, we outline four common classes of NNs that are useful in SE problems:

Multilayer Perceptrons (MLP): They are suitable for classification and regression prediction problems and can be used with different types of data, such as image and text. Besides, when evaluating the performance of different algorithms, we can use MLP results as a baseline of comparison. Basically, MLPs consist of one or more layers of neurons. The input layer receives the data, the hidden layers provide abstraction levels, and the output layer makes predictions.

Convolutional Neural Networks (CNN): Although they were designed for image recognition, we can use CNN for other classification and regression problems. They also can be adapted to different types of data, such as text and sequences of input data. In summary, the input layer in CNNs receives the data, and the hidden layers are responsible for feature extraction. There are three types of layers in CNNs, such as convolution layers (which filter an input multiple times to build a feature map), pooling layers (responsible

for reducing the spatial size of the feature map), and fully-connected layers. Then, the CNN output can feed a fully connected layer to create the model.

Recurrent Neural Networks (RNN): They are a specialized type of NN for sequence prediction problems, i.e., they are designed to receive historical sequence data and predict the next output value(s) in the sequence. The main difference regarding the traditional MLP can be thought as loops on the MLP architecture. The hidden layers do not use only the current input but also the previously received inputs. Conceptually, this feedback loop adds memory to the network. The Long Short-Term Memory (LSTM) is a particular type of RNN able to learn long-term dependencies. LSTM is one of the most used RNNs in many different applications with outstanding results [17].

Hybrid Neural Network Architectures (HNN): They refer to architectures using two or more types of NNs. Usually, CNNs and RNNs are used as layers in a wider model. As an example from the industry, Google's translate service uses LSTM RNN architectures [1].

3 STUDY DESIGN

We conducted the mapping study following four process steps: (1) definition of research questions, (2) search process, (3) selection of studies, and (4) quality evaluation.

3.1 Research Questions

This study aims to identify and analyze DL techniques from the purpose of understanding their application in the context of software engineering. To achieve this goal, we established three research questions:

- **(RQ1)** *What SE problems are solved by DL?*
- **(RQ2)** *What DL techniques are used in SE problems?*
- **(RQ3)** *How does DL compare with other ML techniques used in SE problems?*

In RQ1, the expected result is a list of primary studies categorized by SE research problem. In RQ2, we expect to identify the most common DL techniques used by the analyzed papers. Finally, in RQ3, we pretend to clarify whether DL techniques necessarily improve the results of solving SE problems comparable with traditional ML approaches.

3.2 Search process

To collect the papers, we used the search string *"deep learn*"* and applied this query to titles, keywords, and abstracts of articles in the following digital libraries: Scopus, ACM Digital Library, IEEE Xplore, Web of Science, SpringerLink, and Wiley Online Library. However, we only considered papers published in SE conferences and journals indexed by CSIndexbr [18], which is a Computer Science Index system.[3] CSIndexbr is considered a GOTO ranking [19], i.e., an information system that provides Good, Open, Transparent, and Objective data about CS departments and institutions.[4]

The SE venues listed by CSIndexbr are presented in Table 1. As can be observed, the system indexes 15 major conferences and 12 major journals in SE, including top-conferences (ICSE, FSE, and ASE), top-journals (IEEE TSE and ACM TOSEM) and also next-tier conferences (MSR, ICSME, ISSTA, etc) and journals (EMSE, JSS, IST, etc). CSIndexbr follows a quantitative criteria, based on h5-index, number of papers submitted and accepted to index the venues.

Table 1: Venues

Acronym	Name
ICSE	Int. Conference on Software Engineering
FSE	Foundations of Software Engineering
MSR	Mining Software Repositories
ASE	Automated Software Engineering
ISSTA	Int. Symposium on Software Testing and Analysis
ICSME	Int. Conf. on Software Maintenance and Evolution
ICST	Int. Conf. on Software Testing, Validation and Verification
MODELS	Int. Conf. on Model Driven Engineering Languages and Systems
SANER	Int. Conf. on Software Analysis, Evolution and Reengineering
SLPC	Systems and Software Product Line Conference
RE	Int. Requirements Engineering Conference
FASE	Fundamental Approaches to Software Engineering
ICPC	Int. Conf. on Program Comprehension
ESEM	Int. Symp. on Empirical Software Engineering and Measurement
ICSA	Int. Conference on Software Architecture
IEEE TSE	IEEE Transactions on Software Engineering
ACM TOSEM	ACM Transactions on Software Engineering and Methodology
JSS	Journal of Systems and Software
IEEE Software	IEEE Software
EMSE	Empirical Software Engineering
SoSyM	Software and Systems Modeling
IST	Information and Software Technology
SCP	Science of Computer Programming
SPE	Software Practice and Experience
SQJ	Software Quality Journal
JSEP	Journal of Software Evolution and Process
REJ	Requirements Engineering Journal

By searching for *"deep learn*"* from August/2019 to December/2019 we found 141 distinct papers in the conferences and journals listed in Table 1.

3.3 Study selection

In this step, we filtered the studies retrieved from the search process to exclude papers not aligned with the study goals. First, we removed papers with less than 10 pages, due to our decision to focus on full papers only. The only exception are papers published at IEEE Software (magazine). In this case, we defined a threshold of six pages to select the papers. After applying this size threshold, we eliminated 49 papers.

Then, we carefully read the title and abstract of the remaining papers to confirm they indeed qualify as research that uses DL on SE-related problems. We eliminated 11 papers, including 5 papers that are not related to SE (e.g., one paper that evaluates an "achievement-driven methodology to give students more control of their learning with enough flexibility to engage them in *deeper learning*") two papers published in other tracks (one paper at ICSE-SEET and one paper at ICSE-SEIP), two papers that only mention DL in the abstract, and two papers that were supersed by a journal version, i.e., we discarded the conference version and considered the extended version of the work. Our final dataset has **81 papers**. Figure 1 summarizes the steps we followed for selecting the papers.

4 RESULTS

4.1 Overview

The study selection resulted in 81 primary studies. We did not define an initial publication date for the candidate papers. Despite that,

[3] https://csindexbr.org
[4] http://gotorankings.org

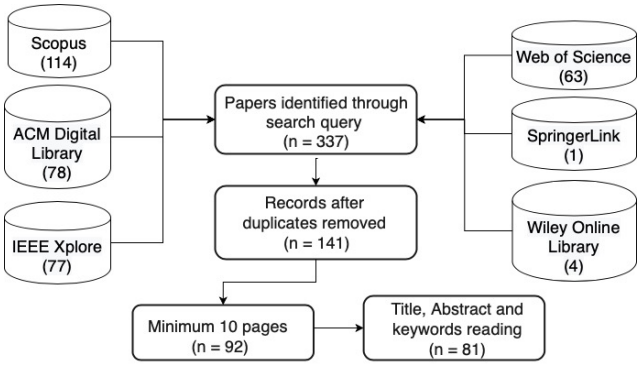

Figure 1: Steps for selecting papers

we found a single paper published in 2015. All other papers are from subsequent years, as illustrated in Figure 2.

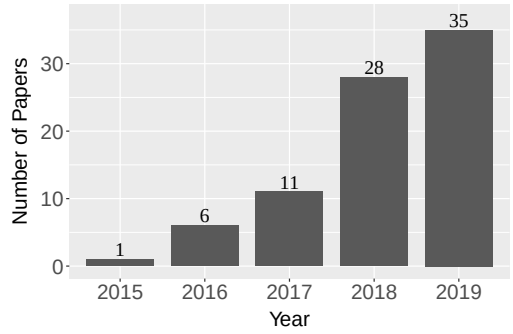

Figure 2: Papers by year

While we only consider nine months from 2019, we have more papers published in 2019 than in 2018, which shows an increasing interest in applying DL in SE.

Figure 3 shows the number of papers by publication venue. Among our primary studies, we identified that 61 studies are from conferences (75.3%) and 20 papers from journals (24.7%). ICSE and FSE concentrate most papers (24 papers or 29.6%). IEEE TSE is the journal with the highest number of papers (7 papers, 8.6%). We did not find papers about DL & SE in nine venues: MODELS, SLPC, RE, FASE, ICSA, ACM TOSEM, SoSyM, SCP, and SQJ.

We found 12 papers (14.8%) with at least one author associated to industry. Microsoft Research has the highest number of papers (3 papers), followed by Clova AI, Facebook, Grammatech, Nvidia, Accenture, Fiat Chrysler, IBM, and Codeplay (each one with a single paper).

(RQ1) What SE problems are solved by DL?

As illustrated in Figure 4, we classified the papers in three principal groups: (1) papers that investigate the usage of SE tools and techniques in the development of DL-based systems; (2) papers that propose the usage of DL-based techniques to solve SE-related problems; and (3) position papers or tutorials. The following subsections describe the papers in each group.

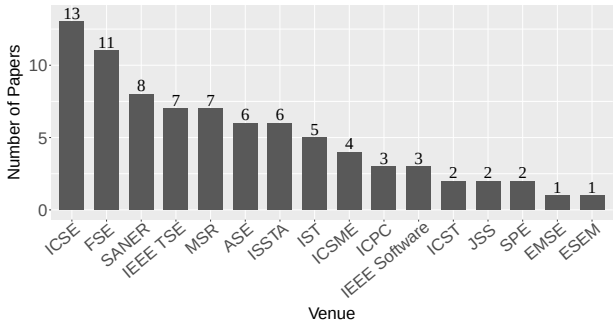

Figure 3: Papers by venue

4.1.1 Using Software Engineering Techniques in DL-based Software. We classified 10 papers in this category (12.3%), including papers that adapt SE tools and techniques to DL-based software (8 papers) and papers that describe empirical studies of DL-based software (2 papers). Papers that apply SE to DL are mostly focused on solving particular problems for testing DL-based software [20–25]. We also found papers that describe quantitative metrics to assess DL-based software [26] and to support the deployment of DL-based software systems [27]. Finally, we found two empirical studies of DL-based software, both investigating the characteristics of the bugs reported in such systems [28, 29].

4.1.2 Using DL Techniques in SE Problems. The usage of DL in SE concentrates in three main problems: documentation, testing, and defect prediction. We provide more details in the following paragraphs:

Documentation: This category has the highest number of papers (13 papers, 16%). Seven papers study problems associated with Stack Overflow (SO) questions and answers. For example, in a paper at ASE16, Xu *et al.* [30] were one the first to use word embedding and CNN to link semantically related knowledge units (KU) at Stack Overflow They claim that traditional word representations miss many cases of potentially linkable KUs at Stack Overflow. Interestingly, at FSE17, Fu *et al.* [31] showed that a simple optimizer used together with the SVM algorithm can achieve similar results, but 84x faster than Xu's runtime performance. Next, at ESEM18, Xu *et al.* [32] replicate the two previous studies using a large dataset. They report that (1) the effectiveness of both approaches declined sharply in the new dataset; (2) despite that, SVM continues to outperform by a small margin DNN approaches in this new dataset; (3) however, both approaches are outperformed by another lightweight SVM method, called SimBow. Finally, at MSR18, Majumder *et al.* [6] report significant runtime improvements to Fu *et al.* [31] results. First, they cluster the data (with KMeans) and then build local models on the produced clusters.

At ASE16, Chen *et al.* [33] rely on word embedding and CNN to translate Chinese queries to English before searching in Stack Overflow. The authors trained the CNN model with large amount of duplicate questions (0.3 million) in Stack Overflow and the corresponding Chinese translations of these questions translated by machine (Google Translate).

At MSR19, Wang *et al.* [34] proposed DeepTip, a DL based approach that uses different CNN architectures to extract small, practical, and useful tips from developer answers.

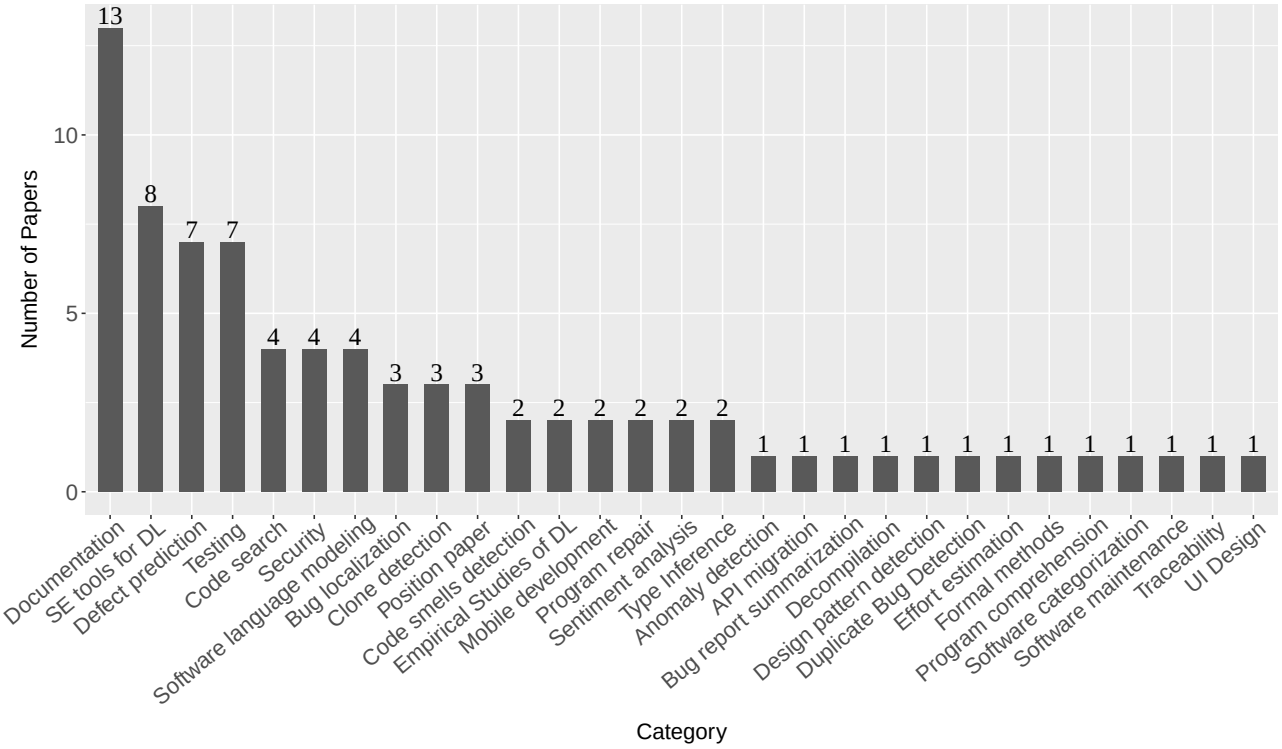

Figure 4: Papers by research problem

At ICPC18, XingHu *et al.* [35] proposed a new approach named DeepCom to generate code comments for Java methods automatically. The authors use a DNN that analyzes structural information of Java methods for better comments generation.

We also found two papers [7, 36] about the automatic identification of source code fragments in videos. At ICSE19, Zhao *et al.* [36] present a DL-based computer vision technique to recognize developer actions from programming screencasts automatically. The authors detect screen changes by image differencing and then recognize developer actions with a CNN model. At MSR18, Ott *et al.* [7] also proposed a DL solution based on CNN and autoencoders to identify source code in images and videos. They identified the presence of typeset and handwritten source code in thousands of video images with 85.6%-98.6% of accuracy.

Testing: We found seven papers (8.6%) using DL in software testing, covering fuzzing tests [37–39], fault localization [14, 40], mutation testing [12], and testing of mobile apps [13]. For example, the three papers related to fuzzing tests adopt the LSTM model to automate the generation of test cases. Specifically, at ASE17, Godefroid *et al.* [38] introduce Learn&fuzz, a first attempt to automatically generate test data for the Microsoft Edge renderer. Their tool uses a corpus of PDF files and LSTM network as the encoder and decoder of seq2seq model. The authors claim that grammar-based fuzzing is the most effective fuzzing technique despite the grammar being written manually, making it a laborious and time-consuming process. Later, at ISSTA18, Cummins *et al.* [37] also claim that traditional approaches based on compiler fuzzing are time-consuming (e.g., they can take nine months to port from a generator to another) and require a huge effort. Then, they introduce DeepSmith [37], an approach that encodes a corpus of handwritten code and uses LSTM to speed up their random program generator (requiring less than a day to train). The authors also state that their framework requires low effort, detects a similar number of bugs compared to the state-of-the art, and reveal bugs not detected by traditional approaches.

The last paper we analyzed concerning fuzzing tests is SeqFuzzer, by Zhao *et al.* [39], published at ICST19 . In this paper, the authors argue that the increasing diversity and complexity of industrial communication protocols is a major challenge for fuzz testing, especially when dealing with stateful protocols. In addition, they also assert that current approaches have a significant drawback, since the process to model the protocol for generating test cases is laborious and time-consuming. To tackle this problem, SeqFuzzer also uses a seq2seq model based on a LSTM network to detect protocol formats automatically and to deal with the temporal features of stateful protocols. The authors report that SeqFuzzer automatically generates fuzzing information with high receiving rates and has successfully identified various security vulnerabilities in the EtherCAT protocol.

In a paper published at SANER19, Zhang *et al.* [40] proposed CNN-FL, an approach to improve fault localization effectiveness. The authors claim that preliminary works found promising insights to fault localization but the current results are still preliminary. They use execution data from test cases to train the CNN network

and the results show that CNN-FL notably improves the fault localization effectiveness. After CNN-FL, at ISSTA19, Li *et al.* [14] claim that traditional learning-to-rank algorithms are very effective on fault localization problems, but they fail to automatically select important features and discover new advanced ones. To tackle this problem, they propose DeepFL to automatically detect or generate complex features for fault location prediction. DeepFL uses LSTM and MLP networks on different feature dimensions, such as code-complexity metrics and mutation-based suspiciousness. The authors showed that DeepFL outperforms traditional algorithms on the same set of features (detects 50+ more faults within Top-1) and during prediction phase (1200x faster).

Finally, we found a paper related to mutation test [12] and another about testing of mobile apps [13]. At ICST19, Mao *et al.* [12] evaluate the effectiveness and efficiency of a predictive mutation testing technique under cross-project on a large dataset. They use 11 classification algorithms including traditional ML (e.g. Random Forest) and DL algorithms (MLP, CNN, and Cascading Forest). They report that Random Forest achieves similar performance than the evaluated DL algorithms. At ICSE17, Liu *et al.* [13] train a RNN to learn from existing UI-level tests to generate textual inputs for mobile apps. The authors claim that despite the improvements on automated mobile testing, relevant text input values in a context remains a significant challenge, which makes hard the large-scale usage of automated testing approaches. They report that RNN models combined with a Word2Vec result in app-independent approach.

Defect Prediction: We also found seven papers (8.6%) that rely on DL for defect prediction. At ICSE16, Wang *et al.* [10] argue that traditional features used by defect predictors fail to capture semantic differences in programs. Then, they use a Deep Belief Network (DBN) to automatically learn semantic features from token vectors extracted from ASTs. For within-project defect prediction, they report an average improvement of 14.2% in F-score. An extended version of their work appeared at TSE18. Among other improvements, the authors included four open-source commercial projects in their evaluation setup (e.g., google/guava and facebook/buck). At MSR19, Dam *et al.* [41] also argue that traditional features do not capture the multiple levels of semantics in a source code file. Then, they use LSTM to extract both syntactic and structural information from ASTs. They evaluate the model using real projects provided by an industrial partner (Samsung). They report a recall of 0.86 (23% of improvement from Wang previous results), but at a lower precision. However, the authors claim that recall is more important when predicting defects, because the costs of missing defects might be much higher than the ones of reporting false defects.

At TSE18, Wen *et al.* [42] use change sequences to predict defects, e.g., a conventional metric used by standard models might be the number of distinct developers who changed a particular file. Instead of using this single value, they use vectors with the name of developers responsible for each change performed in the file. The authors report F-measure of 0.657, which represents an improvement of 31.6% over prediction models based on traditional metrics. At MSR19, Hoang *et al.* [11] propose DeepJIT to automatically extract features from code changes and commit messages. The goal is to use DL to extract features that represent the semantics of commits. The authors report improvements of around 10% when DeepJIT is compared with a state-of-the-art benchmark.

At IST18, Tong *et al.* [8] argue that Stacked Denoising Autoencoders (SDAEs) were never used in the field of software defect prediction. Finally, at IST19, Zhou *et al.* [43] propose a new deep forest model to predict defects that relies on a cascade strategy to transform random forest classifiers into a layer-by-layer structure.

Other research problems: Other important research problems handled using DL are code search [44–47], security [48–51], and software language modelling [52–55]. The next most investigated research problem, with three papers each, are bug localization [56–58] and clone detection [59–61]. We also found two papers on each of the following problems: code smell detection [62, 63], mobile development [64, 65], program repair [66, 67], sentiment analysis [68, 69], and type inference [70, 71].

Finally, we found one paper related to the following problems: anomaly detection [72], API migration [73], bug report summarization [74], decompilation [75], design patterns detection [76], duplicate bug detection [77], effort estimation [78], formal methods [79], program comprehension [80], software categorization [81], software maintenance [82], traceability [83], and UI design [84].

4.1.3 Position Papers. We classified three papers (3.7%) in this category, all published at IEEE Software. They describe the challenges and opportunities of using DL in automotive software [85, 86] or provide a short tutorial on machine learning and DL [87].

(RQ2) What DL techniques are used in SE problems?

To answer this RQ, we identify the DL technique that each paper uses for supporting the SE research problem. Figure 5 shows the most common DL techniques used by the analyzed papers. The most common techniques are CNN (18 papers, 22.2%), RNN (17 papers, 20.9%), and HNN (12 papers, 14.8%).

Table 2 shows the distribution of the DL techniques by research problem. As we can observe, RNNs are used in all problems, except security and bug localization. Although CNN is used in more papers

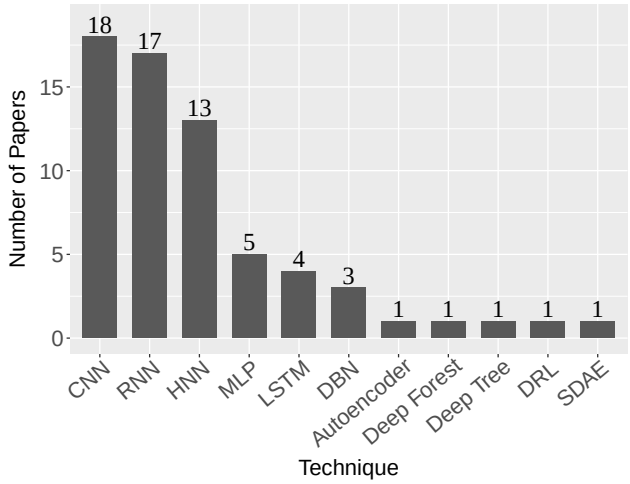

Figure 5: Papers by DL technique

Table 2: Neural networks techniques by research problem

	Neural Networks					
Problem	**CNN**	**RNN**	**HNN**	**LSTM**	**DBN**	**MLP**
Documentation	•	•	•			
Defect prediction		•	•		•	
Testing	•	•	•	•		
Code search		•			•	
Security			•	•		
Software language modeling		•				
Bug localization	•		•			•
Clone detection	•	•				•

(18 papers), they have a focus on four problems (documentation, testing, bug localization, and clone detection).

(RQ3) How does DL compare with other machine learning techniques used in SE problems?

The use of DL techniques to solve SE problems has increased significantly. This popularity may lead researchers to adopt DL because they believe it will always be the best choice, or because of its fame. However, sometimes applying DL to a SE problem may not be the best approach and could even be a wrong choice considering its drawbacks. For example, Majumder *et al.* [6] extend recent results for text mining Stack Overflow by clustering the dataset, then tuning every learner within each cluster. They obtained results over 500 times faster than DL (and over 900 times faster if they use all the cores on a standard laptop computer). Fakhoury *et al.* [62] also compare traditional ML with DL techniques. The authors build and validate an oracle of around 1,700 instances and create binary classification models using traditional ML approaches and CNN. Their results show that traditional ML algorithms outperform deep neural networks in all evaluation metrics and resources (time and memory).

Mao *et al.* [12] use ML and DL algorithms to evaluate the effectiveness and efficiency of a predictive mutation testing technique under cross-project dataset. The results show that Random Forest (ML algorithm) achieves nearly the same effectiveness for Predictive Mutation Testing compared to advanced DL models.

In some cases, adaptations in traditional machine learning techniques are sufficient to improve the results. For example, Fu and Menzies [31] use deep learning to find which questions in Stack Overflow can be linked together. They show that applying a very simple differential evolution optimizer to fine-tune SVM can achieve similar (and sometimes better) results. Their approach terminated in 10 minutes, i.e., 84 times faster than the deep learning method.

Zhou *et al.* [43] investigate whether DL is better than traditional approaches in tag recommendation for software information sites. Their findings show that their model, with some engineering, can achieve better results than DL models.

5 DISCUSSION

As discussed in Section 4, the use of DL in SE problems has increased significantly. To analyze whether Deep Learning benefits the software engineering community, we focused on papers that use DL techniques to solve SE problems to identify the strengths and drawbacks of these techniques.

5.1 Strengths

To understand why a significant number of papers are frequently adopting DL, we looked at papers that use DL techniques to solve SE-related problems aiming to identify their strengths. In what follows, we present the key strengths pointed out by the studies:

Best results with unstructured data: most ML algorithms face difficulties to analyze unstructured data like pictures, pdf files, audios, and more. In contrast, it is possible to use different data formats to train DL algorithms and obtain insights that are relevant to the training purpose. For example, Zhou *et al.* [36] recognize developers actions from programming screencasts. Using programming screencasts from Youtube, they demonstrate the usefulness of their technique in a practical scenario of action-aware extraction of key-code frames in developers' work. Ott *et al.* [7] use a similar DL approach to identifying source code in images and videos.

No need for Feature Engineering (FE): FE is an essential task in ML, once it improves model accuracy. However, it often requires domain knowledge in a specific problem. The main advantage of DL techniques is their ability to apply FE by themselves. DL algorithms scan the data to find correlated features and combine them through multi-layers to enable faster learning. According to Zhou *et al.* [36], e.g., recognizing developers' actions from programming screencasts is a challenging task due to the diversity of developer actions, working environments, and programming languages. Thus, they use the CNN's ability to automatically extract image features from screen changes resulting from developer actions.

No need for labeling data: labeling process gets a set of unlabeled data and adds meaningful tags for each piece of these data. For example, labels may indicate whether an image contains a dog or a cat, what is the sentiment of a post, or what is the action performed in a video. ML requires labeled data for training, which may be expensive since these algorithms usually require thousands of data. By contrast, DL supports unsupervised learning techniques that allow the learning with no guidelines. For example, Dam *et al.* [48] use LSTM to detect the most likely locations of code vulnerabilities in an unlabeled code base. According to the authors, LSTM offers a powerful representation of source code and can automatically learn both syntactic and semantic features that represent long-term source code dependencies.

Best results with sequence prediction: RNNs are designed to receive historical sequence data and predict the next output value in the sequence. Unlike traditional ML techniques, RNNs can embed sequential inputs, such as sentences in their internal memory. This step allows them to achieve better results for tasks that deal with sequential data, e.g., text and speech [31].

High-quality results: once trained correctly, DL techniques can perform thousands of repetitive tasks with better performance compared to traditional ML techniques [42].

We summarize these strengths and the papers that cited each of them in Table 3.

5.2 Drawbacks

Despite their benefits, DL-based techniques also have some drawbacks (which are summarized in Table 4).

Table 3: Strengths

Strength	#	References
Best results with unstructured data	5	[84], [7], [36], [62], [75]
No need for feature engineering	12	[7], [81], [36], [48], [35], [76], [11], [42], [9], [41], [50], [80]
No need for labeling data	6	[44], [48], [69], [74], [9], [41]
Best results with sequential data	6	[13], [70], [47], [31], [75], [42]
High-quality Results	10	[78], [40], [35], [69], [52], [77], [74], [59], [45], [63]

Table 4: Drawbacks

Drawback	#	References
Computational cost	22	[6], [7], [44], [31], [47], [53], [40], [45], [70], [79], [84], [77], [75], [56], [62], [11], [43], [41], [61], [63], [80], [50]
Dataset size and quality	7	[55], [63], [62], [42], [43], [50], [80]
Replication	3	[6], [62], [42]
Number of parameters	11	[6], [78], [53], [76], [10], [63], [62], [43], [68], [41], [51]
Implementation difficulty	4	[6], [62], [80], [43]

Computational cost: the process of training is the most challenging part of using DL-based techniques, and by far the most time consuming since DL algorithms learn progressively. Consequently, DL-based techniques require high-performance hardware, such as hundreds of machines equipped with expensive multi-core GPUs. According to Fu and Menzies [31], even with advanced hardware, training DL models requires from hours to weeks. We found references to high computational cost in at least 22 papers.

Size and quality of the dataset: DL algorithms require extensive amounts of data for training. While companies like Google, Facebook, and Microsoft have abundant data, most researchers do not have access to a massive amount of data. For example, the effectiveness of DeepSims, a DL approach proposed by Zhao and Huang [55], is limited by the size and quality of the training dataset. The authors state that *"building such a large and representative dataset is challenging"*.

Replication: once trained with data, DL models are very efficient in formulating an adequate solution to a particular problem. However, they are unable to do so for a similar problem and require retraining. Moreover, reproducing DL results is also a significant problem [6]. Majumder [6] state that: *"it is not yet common practice for deep learning researchers to share their implementations and data, where a tiny difference may lead to a huge difference in the results"*.

Number of parameters: DL techniques are multi-layered neural networks. The number of parameters grows with the number of layers, which might lead to overfitting, a tendency to "learn" characteristics of the training data that does not generalize to the population as a whole. At least 11 papers reference this problem.

6 CONCLUSION

In this work, we analyzed 81 recent papers that apply DL techniques to SE problems or vice-versa. Our main findings are as follows: i) DL is gaining momentum among SE researchers. For example, 35 papers (43.2%) are from 2019 and only one paper from 2015; ii) The top-3 research problems tackled by the analyzed papers are documentation (13 papers), defect prediction (7 papers), and testing (7 papers), and iii) The most common neural network type used in the analyzed papers is CNN and RNN.

The list of papers and the data analyzed in this work are available at: https://doi.org/10.5281/zenodo.4302713

ACKNOWLEDGMENTS

Our research is supported by FAPEMIG and CNPq.

REFERENCES

[1] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," *CoRR*, vol. abs/1609.08144, 2016.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.
[3] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. A. Mujica, A. Coates, and A. Y. Ng, "An empirical evaluation of deep learning on highway driving," *CoRR*, vol. abs/1504.01716, 2015.
[4] M. Mohammadi, A. Al-Fuqaha, M. Guizani, and J. Oh, "Semisupervised deep reinforcement learning in support of iot and smart city services," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 624–635, 2018.
[5] P. Zhou, J. Liu, X. Liu, Z. Yang, and J. Grundy, "Is deep learning better than traditional approaches in tag recommendation for software information sites?" *IST*, vol. 109, no. 1, pp. 1 – 13, 2019.
[6] S. Majumder, N. Balaji, K. Brey, W. Fu, and T. Menzies, "500+ times faster than deep learning: A case study exploring faster methods for text mining StackOverflow," in *MSR*, 2018, pp. 554–563.
[7] J. Ott, A. Atchison, P. Harnack, A. Bergh, and E. Linstead, "A deep learning approach to identifying source code in images and video," in *MSR*, 2018, pp. 376–386.
[8] H. Tong, B. Liu, and S. Wang, "Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning," *IST*, vol. 96, no. 1, pp. 94–111, 2018.
[9] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep semantic feature learning for software defect prediction," *IEEE TSE*, pp. 1–26, 2018.
[10] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *ICSE*, 2016, pp. 297–308.
[11] T. Hoang, H. K. Dam, Y. Kamei, D. Lo, and N. Ubayashi, "DeepJIT: An end-to-end deep learning framework for just-in-time defect prediction," in *MSR*, 2019, pp. 34–45.
[12] D. Mao, L. Chen, and L. Zhang, "An extensive study on cross-project predictive mutation testing," in *ICST*, 2019, pp. 160–171.
[13] P. Liu, X. Zhang, M. Pistoia, Y. Zheng, M. Marques, and L. Zeng, "Automatic text input generation for mobile testing," in *ICSE*, 2017, pp. 643–653.
[14] X. Li, W. Li, Y. Zhang, and L. Zhang, "DeepFL: Integrating multiple fault diagnosis dimensions for deep fault localization," in *ISSTA*, 2019, pp. 169–180.
[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
[16] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.
[17] Yu Wang, "A new concept using lstm neural networks for dynamic system identification," in *2017 American Control Conference (ACC)*, 2017, pp. 5324–5329.
[18] M. T. Valente and K. Paixao, "CSIndexbr: Exploring the Brazilian scientific production in Computer Science," *arXiv*, vol. abs/1807.09266, 2018.
[19] E. Berger, S. M. Blackburn, C. E. Brodley, H. V. Jagadish, K. S. McKinley, M. Nascimento, M. Shin, K. Wang, and L. Xie, "GOTO rankings considered helpful," *Communications of the ACM*, vol. 62, no. 7, pp. 29–30, 2019.
[20] H. V. Pham, T. Lutellier, W. Qi, and L. Tan, "CRADLE: Cross-backend validation to detect and localize bugs in deep learning libraries," in *ICSE*, 2019, pp. 1027–1038.
[21] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang, "DeepGauge: Multi-granularity testing criteria for deep learning systems," in *ASE*, 2018, pp. 120–131.
[22] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "DeepHunter: A coverage-guided fuzz testing framework for deep neural networks," in *ISSTA*, 2019, pp. 146–157.
[23] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," in *ICSE*, 2018, pp. 303–314.
[24] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *ICSE*, 2019, pp. 1039–1049.
[25] A. Dwarakanath, M. Ahuja, S. Sikand, R. M. Rao, R. P. J. C. Bose, N. Dubash, and S. Podder, "Identifying implementation bugs in machine learning based image classifiers using metamorphic testing," in *ISSTA*, 2018, pp. 118–128.

[26] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, and J. Zhao, "DeepStellar: Model-based quantitative analysis of stateful deep learning systems," in *FSE*, 2019, pp. 477–487.

[27] E. D. Coninck, S. Bohez, S. Leroux, T. Verbelen, B. Vankeirsbilck, P. Simoens, and B. Dhoedt, "DIANNE: a modular framework for designing, training and deploying deep neural networks on heterogeneous distributed infrastructure," *JSS*, vol. 141, no. 1, pp. 52 – 65, 2018.

[28] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan, "A comprehensive study on deep learning bug characteristics," in *FSE*, 2019, pp. 510–520.

[29] Y. Zhang, Y. Chen, S.-C. Cheung, Y. Xiong, and L. Zhang, "An empirical study on TensorFlow program bugs," in *ISSTA*, 2018, pp. 129–140.

[30] B. Xu, D. Ye, Z. Xing, X. Xia, G. Chen, and S. Li, "Predicting semantically linkable knowledge in developer online forums via convolutional neural network," in *ASE*, 2016, pp. 51–62.

[31] W. Fu and T. Menzies, "Easy over hard: A case study on deep learning," in *FSE*, 2017, pp. 49–60.

[32] B. Xu, A. Shirani, D. Lo, and M. A. Alipour, "Prediction of relatedness in Stack Overflow: Deep learning vs. svm: A reproducibility study," in *ESEM*, 2018, pp. 21:1–21:10.

[33] G. Chen, C. Chen, Z. Xing, and B. Xu, "Learning a dual-language vector space for domain-specific cross-lingual question retrieval," in *ASE*, 2016, pp. 744–755.

[34] S. Wang, N. Phan, Y. Wang, and Y. Zhao, "Extracting API tips from developer question and answer websites," in *MSR*, 2019, pp. 321–332.

[35] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *ICPC*, 2018, pp. 200–210.

[36] D. Zhao, Z. Xing, C. Chen, X. Xia, and G. Li, "ActionNet: Vision-based workflow action recognition from programming screencasts," in *ICSE*, 2019, pp. 350–361.

[37] C. Cummins, P. Petoumenos, A. Murray, and H. Leather, "Compiler fuzzing through deep learning," in *ISSTA*, 2018, pp. 95–105.

[38] P. Godefroid, H. Peleg, and R. Singh, "Learn&Fuzz: Machine learning for input fuzzing," in *ASE*, 2017, pp. 50–59.

[39] H. Zhao, Z. Li, H. Wei, J. Shi, and Y. Huang, "SeqFuzzer: An industrial protocol fuzzing framework from a deep learning perspective," in *ICST*, 2019, pp. 59–67.

[40] Z. Zhang, Y. Lei, X. Mao, and P. Li, "CNN-FL: An effective approach for localizing faults using convolutional neural networks," in *SANER*, 2019, pp. 445–455.

[41] H. K. Dam, T. Pham, S. W. Ng, T. Tran, J. Grundy, A. Ghose, T. Kim, and C.-J. Kim, "Lessons learned from using a deep tree-based model for software defect prediction in practice," in *MSR*, 2019, pp. 46–57.

[42] M. Wen, R. Wu, and S. C. Cheung, "How well do change sequences predict defects? sequence learning from software changes," *IEEE TSE*, pp. 1–20, 2018.

[43] T. Zhou, X. Sun, X. Xia, B. Li, and X. Chen, "Improving defect prediction with deep forest," *IST*, vol. 114, no. 1, pp. 204 – 216, 2019.

[44] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep API learning," in *FSE*, 2016, pp. 631–642.

[45] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *ICSE*, 2018, pp. 933–944.

[46] Q. Huang, Y. Yang, and M. Cheng, "Deep learning the semantics of change sequences for query expansion," *Software: Practice and Experience*, pp. 1–18, 2019.

[47] J. Cambronero, H. Li, S. Kim, K. Sen, and S. Chandra, "When deep learning met code search," in *FSE*, 2019, pp. 964–974.

[48] H. K. Dam, T. Tran, T. T. M. Pham, S. W. Ng, J. Grundy, and A. Ghose, "Automatic feature learning for predicting vulnerable software components," *IEEE TSE*, vol. 14, no. 8, pp. 1–19, 2018.

[49] C. Chen, W. Diao, Y. Zeng, S. Guo, and C. Hu, "DRLgencert: Deep learning-based automated testing of certificate verification in SSL/TLS implementations," in *ICSME*, 2018, pp. 48–58.

[50] Z. Han, X. Li, Z. Xing, H. Liu, and Z. Feng, "Learning to predict severity of software vulnerability using only vulnerability description," in *ICSME*, 2017, pp. 125–136.

[51] R. Yan, X. Xiao, G. Hu, S. Peng, and Y. Jiang, "New deep learning method to detect code injection attacks on hybrid applications," *JSS*, vol. 137, no. 1, pp. 67 – 77, 2018.

[52] M. White, C. Vendome, M. Linares-Vásquez, and D. Poshyvanyk, "Toward deep learning software repositories," in *MSR*, 2015, pp. 334–345.

[53] V. J. Hellendoorn and P. Devanbu, "Are deep neural networks the best choice for modeling source code?" in *FSE*, 2017, pp. 763–773.

[54] M. Tufano, C. Watson, G. Bavota, M. D. Penta, M. White, and D. Poshyvanyk, "Deep learning similarities from different representations of source code," in *MSR*, 2018, pp. 542–553.

[55] G. Zhao and J. Huang, "DeepSim: Deep learning code functional similarity," in *FSE*, 2018, pp. 141–151.

[56] A. N. Lam, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Bug localization with combination of deep learning and information retrieval," in *ICPC*, 2017, pp. 218–229.

[57] X. Huo, F. Thung, M. Li, D. Lo, and S. Shi, "Deep transfer bug localization," *IEEE TSE*, pp. 1–12, 2019.

[58] Y. Xiao, J. Keung, K. E. Bennin, and Q. Mi, "Improving bug localization with word embedding and enhanced convolutional neural networks," *IST*, vol. 105, no. 1, pp. 17 – 29, 2019.

[59] L. Li, H. Feng, W. Zhuang, N. Meng, and B. Ryder, "CCLearner: A deep learning-based clone detection approach," in *ICSME*, 2017, pp. 249–260.

[60] H. Yu, W. Lam, L. Chen, G. Li, T. Xie, and Q. Wang, "Neural detection of semantic code clones via tree-based convolution," in *ICPC*, 2019, pp. 70–80.

[61] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," in *ASE*, 2016, pp. 87–98.

[62] S. Fakhoury, V. Arnaoudova, C. Noiseux, F. Khomh, and G. Antoniol, "Keep it simple: Is deep learning good for linguistic smell detection?" in *SANER*, 2018, pp. 602–611.

[63] H. Liu, J. Jin, Z. Xu, Y. Bu, Y. Zou, and L. Zhang, "Deep learning based code smell detection," *IEEE TSE*, pp. 1–28, 2019.

[64] C. Guo, D. Huang, N. Dong, Q. Ye, J. Xu, Y. Fan, H. Yang, and Y. Xu, "Deep review sharing," in *SANER*, 2019, pp. 61–72.

[65] C. Guo, W. Wang, Y. Wu, N. Dong, Q. Ye, J. Xu, and S. Zhang, "Systematic comprehension for developer reply in mobile system forum," *SANER*, pp. 242–252, 2019.

[66] M. Tufano, J. Pantiuchina, C. Watson, G. Bavota, and D. Poshyvanyk, "On learning meaningful code changes via neural machine translation," in *ICSE*, 2019, pp. 25–36.

[67] M. White, M. Tufano, M. Martinez, M. Monperrus, and D. Poshyvanyk, "Sorting and transforming program repair ingredients via deep learning code similarities," in *SANER*, 2019, pp. 479–490.

[68] H. Sankar, V. Subramaniyaswamy, V. Vijayakumar, S. A. Kumar, R. Logesh, and A. Umamakeswari, "Intelligent sentiment analysis approach using edge computing-based deep learning technique," *Software: Practice and Experience*, vol. 0, no. 0, pp. 1–13, 2019.

[69] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto, "Sentiment analysis for software engineering: How far can we go?" in *ICSE*, 2018, pp. 94–104.

[70] V. J. Hellendoorn, C. Bird, E. T. Barr, and M. Allamanis, "Deep learning type inference," in *FSE*, 2018, pp. 152–162.

[71] R. S. Malik, J. Patra, and M. Pradel, "NL2Type: Inferring JavaScript function types from natural language information," in *ICSE*, 2019, pp. 304–315.

[72] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *FSE*, 2019, pp. 807–817.

[73] C. Chen, Z. Xing, Y. Liu, and K. L. X. Ong, "Mining likely analogical apis across third-party libraries via large-scale unsupervised API semantics embedding," *IEEE TSE*, vol. 14, no. 8, pp. 1–1, 2019.

[74] X. Li, H. Jiang, D. Liu, Z. Ren, and G. Li, "Unsupervised deep bug report summarization," in *ICPC*, 2018, pp. 144–155.

[75] D. S. Katz, J. Ruchti, and E. Schulte, "Using recurrent neural networks for decompilation," in *SANER*, 2018, pp. 346–356.

[76] H. Thaller, L. Linsbauer, and A. Egyed, "Feature maps: A comprehensible software representation for design pattern detection," in *SANER*, 2019, pp. 207–217.

[77] J. Deshmukh, A. K. M, S. Podder, S. Sengupta, and N. Dubash, "Towards accurate duplicate bug retrieval using deep learning techniques," in *ICSME*, 2017, pp. 115–124.

[78] M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, A. Ghose, and T. Menzies, "A deep learning model for estimating story points," *IEEE TSE*, vol. 45, no. 7, pp. 637–656, 2019.

[79] T.-D. B. Le and D. Lo, "Deep specification mining," in *ISSTA*, 2018, pp. 106–117.

[80] Q. Mi, J. Keung, Y. Xiao, S. Mensah, and Y. Gao, "Improving code readability classification using convolutional neural networks," *IST*, vol. 104, no. 1, pp. 60–71, 2018.

[81] C. Wang, X. Peng, M. Liu, Z. Xing, X. Bai, B. Xie, and T. Wang, "A learning-based approach for automatic construction of domain glossary from source code and documentation," in *FSE*, 2019, pp. 97–108.

[82] R. Xie, L. Chen, W. Ye, Z. Li, T. Hu, D. Du, and S. Zhang, "DeepLink: A code knowledge graph based deep learning approach for issue-commit link recovery," in *SANER*, 2019, pp. 434–444.

[83] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *ICSE*, 2017, pp. 3–14.

[84] C. Chen, T. Su, G. Meng, Z. Xing, and Y. Liu, "From UI design image to gui skeleton: A neural machine translator to bootstrap mobile GUI implementation," in *ICSE*, 2018, pp. 665–676.

[85] F. Falcini, G. Lami, and A. M. Costanza, "Deep learning in automotive software," *IEEE Software*, vol. 34, no. 3, pp. 56–63, 2017.

[86] F. Falcini, G. Lami, and A. Mitidieri, "Yet another challenge for the automotive software: Deep learning," *IEEE Software*, pp. 1–13, 2017.

[87] P. Louridas and C. Ebert, "Machine learning," *IEEE Software*, vol. 33, no. 5, pp. 110–115, 2016.