American University of Beirut
ECE Department
EECE 796 Special Project Proposal
Summer term 2019

Title: Machine Learning Meets Software Engineering

Student: Ahmad Mustapha

ID: 201924287

Supervisor: Prof. Wassim Masri

Abstract:

Machine Learning has been successfully applied to different domains. And with the rise of Deep Learning, the applicability and efficiency of Machine Learning have increased. However, these technologies have so far a minimal impact on the domain of Software Engineering. We argue that things are changing and the relevance of machine learning in Software Engineering is increasing. In this project, we aim to study the interaction between Machine Learning including Deep Learning from one side and Software Engineering including Software Testing from the other side. To do this we will distill a collected set of 1516 research papers from five top Software Engineering Conferences published between 2017 and 2019 inclusive and study the papers that apply Machine Learning in Software Engineering and vice versa.

Outline:

Introduction:

Software Engineering research aims to make software development cycle efficient in time to delivery, reliability, and costs. Machine learning as automation components has intuitively the potential to push the aforementioned targets to new records. To understand this interaction between the two domains we scraped ~1500 research papers from top software engineering conferences throughout multiple years and filtered out the papers that have used machine learning.

Our study has revealed different insights related to the community dataset practices, the machine learning models utilized, the phases of software engineering that attracts machine learning, and most importantly the application domain of using machine learning in software engineering.

The application of Artificial Intelligence on Software Engineer have been conducted since the mid-'80s [1]–[3]. However, and as ML is a subfield of AI, efforts to summarize the interaction between ML and SE on the Application levels are scarce. A notable effort is done by [4], [5], however, those are based on very old research that goes back to the '80s and '90s and it is well known that the domain of machine learning have considerably changed. New publications related to the topic are very broad [6], [7]. The most comprehensive effort has been done by [8], their work directly related to our expected work where they studied around 100 papers that applies Deep Learning on Software Engineering. However, they only targeted Deep Learning and their focus was on models used rather than the types of applications. From this short review, one can see that this niche is not yet fully investigated.

The rest of the report is divided as follow: (1) Methodology, describes our procedure to find papers that combines software engineering and machine learning. (2) Machine Learning Insights, summarize insights about the bolts and nuts of the machine learning practices of the community. (3) Software Engineering Insights, summarizes insights about the community for software engineering point of view. Where machine learning is mostly applied? What are the application domain or themes of applying machine learning? (4) The Rise of SE 2.0, a concluding section of the report.

Methodology:

To get insights about the interaction between machine learning and software engineering we followed the following methodology. We first scraped all publications from 6 major Software engineering conferences throughout the years 2017-2019. The conferences we considered are:

- IEEE International Conference on Software Testing, Verification and Validation (ICST)
- International Conference on Software Engineering (ICSE)
- The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)
- IEEE/ACM International Conference on Automated Software Engineering (ASE)
- The ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)
- International Symposium on Software Reliability Engineering (ISSRE)

The total number of collected papers was 1516 paper, distributed on the conferences as visible in Figure 1:

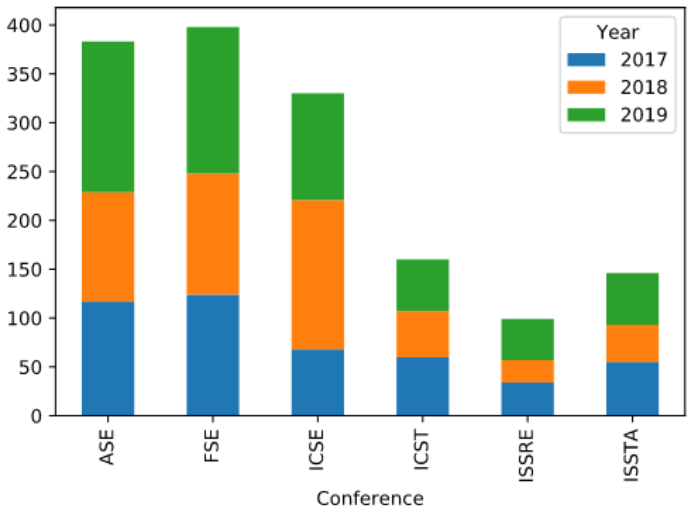


Figure 1 The number of papers scraped

To filter out the papers that are related to machine learning we proposed the following NLP based approach. We manually defined a bag of words that encapsulates machine learning concepts. The *ML_tokens* are ["train", "training", "learning", "classify", "classifier", "cluster", "clustering", "regression", "machine", "deep", "neural", "network", "networks"]. We counted the number of ML_tokens appeared in the **abstract** and **title** of each paper. Note that we only counted whether each token appeared or not discarding the number of times it appeared. We used the sum of the appeared tokens as a metric to score papers relatedness to machine learning. Table 1 shows the distribution of machine learning score.

score	0	1	2	3	4	5	6	7	8	9
count	988	264	139	51	39	19	8	6	1	1

Table 1 Count Distribution of Machine Learning Score

According to our heuristic 34% of the papers contained at least 1 machine learning term. However, this might not be insightful. As some tokens might appear in a different context to machine learning. For example, “deep” token can appear as “deep understanding” and “network” can appear in the context of computer networks. Using bi-grams might increase the precision of the filtering process, though the naïve approach we used is enough to take a sample to study. The sample we studied for this manuscript is the top 100 papers sorted by machine learning score. The score for those papers ranged from 3 to 9. While studying the papers we found that they are related to machine learning and the number of false-positive is negligible. Figure 2 shows the count distribution and the relative percentage of the studied papers (machine learning papers) over conferences.

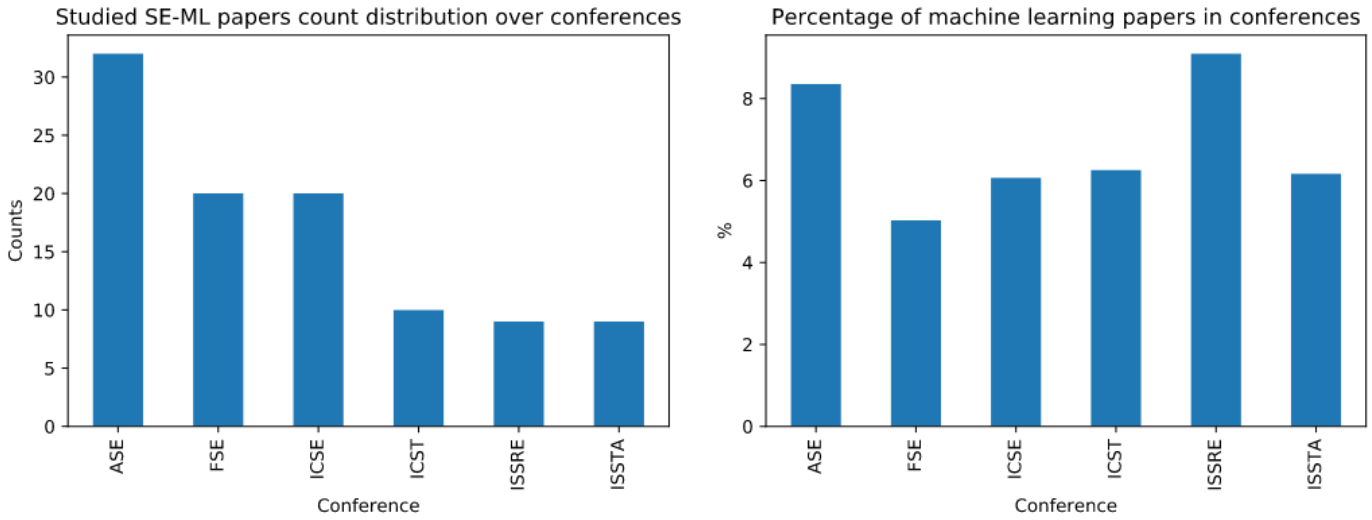


Figure 2 Count Distribution and percentage of studied papers w.r.t conferences

On Average machine learning papers represent 6.8% of published papers in each conference throughout 3 years. Interestingly ISSRE achieved the highest percent (9%) of machine learning papers relative to the number of published papers (approx. 100) in the case of ISSRE.

In the following sections, we will discuss different insights we abstracted from the studied papers on different dimensions. Note that for time restrictions and the sake of reporting we only studied 40 papers out of the selected 100. We plan to complete our insights based on all the 100 papers.

Machine Learning Insights:

Machine Learning has a recurrent pipeline. The pipeline can be abstracted as follows:

1. Dataset Collection and Preparation
2. Feature Selection
3. Model Selection
4. Model Training
5. Model Evaluation
6. Model Deployment

In this section, we will study the unique characteristics of applying Machine Learning in the scope of Software Engineering. Model training, evaluation, and

deployment are fine-grained details that depend on the selected model and thus will not be used to retrieve insights.

Datasets:

Dataset collection and the quality of data is very essential and critical for any machine learning application. The development of the machine learning domain (including deep learning) dependent on a special practice performed by the community which is *dataset benchmarks*. The practice is defined by creating different well-developed datasets for every sub-domain and make them easily accessible for the community to work on and use as benchmarks. Most notable is the ImageNet dataset which contained about 1 million images distributed over 1000 class and was fueling traditional computer vision and recently deep learning research. Does the Software Engineering community follow a similar practice?

Out of the studied papers, 50% of the dataset used was collected by the papers' authors and 50% have been based on previous community datasets. These results can be linked to the fact that software engineering has a wide set of micro-domains. However, although 50% of studied papers used previous community datasets, the referenced datasets don't stand by themselves. In other words, the datasets are not identified by a name but through references to previous research. This hinders the research pace and scatter research efforts that have similar goals. For example: [9] referenced their dataset as "*three ground truth datasets provided by Genome project (ref), Drebin (ref), and Fanet al. (ref)*"; [10] referenced the dataset as "*CODEnn-Java-Train is the dataset publicly released by the authors of (ref)*" where the dataset is hosted on an online drive.; [11] stated "*For the experiments, we use eleven real-world configurable software systems … These software systems were measured and published online (ref). More information about these systems and how they were measured can be found in (ref)*".

On the other hand, there exist some standalone datasets although the sample size wasn't large enough to prove their recurrent usage. Some of those are: **OWASP**[1] which is a fully runnable open source web application that contains thousands of exploitable test cases, each mapped to specific CWEs, which can be analyzed by any

[1] https://owasp.org/www-project-benchmark/

type of Application Security Testing (AST) tool. **CaDO**[2] Dataset which stands for Content Analysis for API Documentation and contains a list of Java and .Net documentations along with its API knowledge type. **LinuxFlaw**[3] which is a repository of Linux software vulnerabilities and how to trigger them. **Qualitas Corpus** [4] contains the source/compiled code of 100 java systems while some of them have different versions. … Note that some of these datasets are accessible only per request.

Research efforts that solely aim to create well-developed large scale open datasets for Software Engineering would be of great importance for the field.

Model Selection:

The authors in [8] have done considerable work in specifying the deep learning models used in Software Engineering research. In this section, we introduce our observations of the usage of these models along with traditional machine learning models. Figure 3 summarizes the papers counts of each machine learning model. Note that some papers used multiple models. Also, some papers were related to testing deep learning models we omit those from counting as the choice of the model in these papers can be arbitrary.

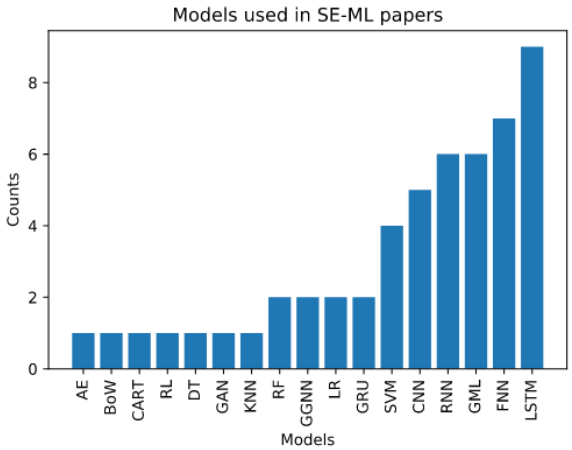


Figure 3 Machine Learning Models Used

[2] https://cado.informatik.uni-hamburg.de/data-sets/
[3] https://github.com/VulnReproduction/LinuxFlaw
[4] http://qualitascorpus.com/

The list of models found are:

- Autoencoder (AE)
- BoW (Bag of Words)
- CART (Classification and Regression Tree)
- RL (Reinforcement Learning)[5]
- Decision Tree (DT)
- Generative Adversarial Network (GAN)
- K-Nearest Neighbor (KNN)
- Random Forest (RF)
- Gated Graph Neural Network (GGNN)
- Logistic Regression (LG)
- Gated Recurrent Units (GRU)
- SVM (Support Vector Machine)
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN)
- General Machine Learning (GML) [6]
- Feed Forward Neural Network (FNN)
- Long Short Term Memory (LSTM)

Note that models with variations are grouped in the same category. For example [12] used SVM with graph kernels and [13] used RNNs with Abstract Syntax Tree-based Neural Network. The results show that LSTM is the most used model preceded by FNNs. This is expected as the sequential nature of source code requires models that operate on sequences and this where LSTM shines.

The papers that used deep learning models outnumber those that used traditional machine learning. This can be related to our sampling procedure. The *ML-tokens* we defined have 4 tokens solely related to deep learning, hence the higher the ML score the more the paper is related to deep learning. And because we selected the top 100 papers with highest scores, we naturally favored those that are more related to deep learning. We only studied the first 40 papers out of those 100. Studying the rest of the papers will lead to a more representative view of the actual usage of machine learning models.

It is insightful to mention also that 20% of the studied papers applied both traditional machine learning along with Deep Learning for comparison purposes.

[5] Not a model obviously but we went with this because it appeared only one time.

[6] When the authors used some machine learning techniques: evolutionary algorithm, graph centrality, sampling, concept drift resolution, …

Software Engineering Insights:

Software Engineering research is concerned with increasing the efficiency of the software engineering process in terms of time, quality, reliability, and costs. In this section, we will present insights related to the software engineering aspects of the studied papers.

Software Engineering Process:

The Software Engineering process is composed of the following major phases: Requirements, Design, Development, Testing, Documentation, Deployment, and Maintenance. In which of these phases machine learning is mostly utilized? Figure 4 summarizes the distribution of machine learning papers over software engineering phases. The testing phase was the phase that mostly attracted machine learning techniques. Note that we designated security and automatic repair as belonging to the testing phase. Note that the categorization of papers was done manually by one researcher which increases the possibility of bias. This should be resolved by asking multiple researchers to do the categorization and then resolve categorization conflicts. Concerning the Extra category, it represents research that doesn't fit in any part of the SE cycle but the cycle as an all. For example [14] developed a system to profile open source developers by aggregating data from different open source development platforms and apply learning on it. [13] proposed a code representation that utilizes Abstract Syntax Trees. [15] proposed a machine learning approach to identify the functional similarity between code fragments. All these papers don't fit any SE phase per se but can be applied to aid in different phases.

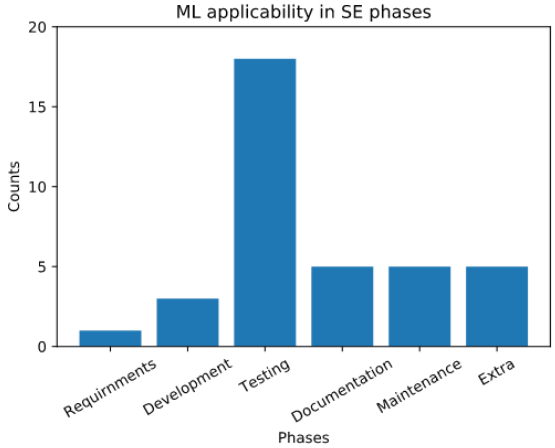


Figure 4 Count distribution of machine learning papers over SE phases

Targeted Languages:

What are the languages targeted by the studied papers? For each studied paper we manually designated the language used to demonstrate the approach. By "used" we don't mean the language used to implement the machine learning pipeline but the language used in the form of source code to build datasets used for training and evaluation. Figure 5 summarizes the count distribution of the studied papers over programming languages. Java has attracted the most number of publications. Java seems the preferred language to apply research on in the community.

This fact calls the research community to consider different languages. Machine Learning research is dependent on datasets and what works on one dataset representing one language might not necessarily work on other datasets that represent other languages.

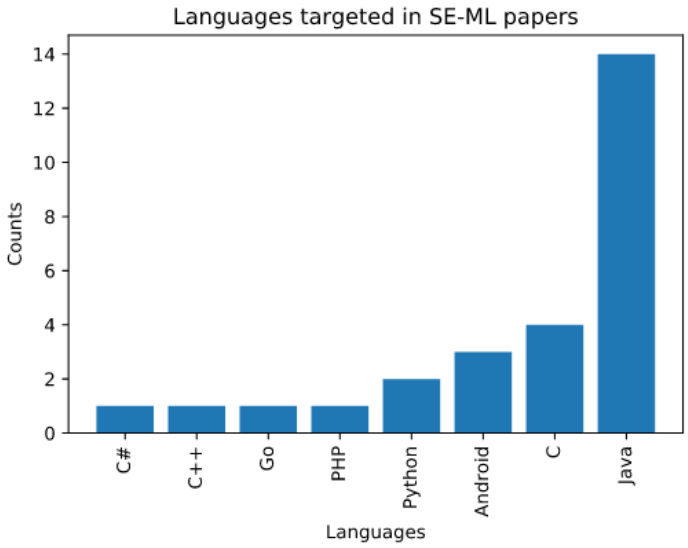


Figure 5 Count distribution of studied papers over programming languages

Applications:

It is of importance to understand the state of the art applications of machine learning in software engineering. This would allow the community to find what is working out and focus on it or find what has not been tried and try it. To approach this requirement, we tried to abstract the papers research targets as much as possible to find high-level patterns of machine learning usage in software engineering. Because

of the small number of papers studied so far, it was not easy to find abstraction for all applications. Figure 6 summarizes the formulated abstract themes and their counts distribution. The other category contains a set of diverse applications that didn't fill in the other categories.

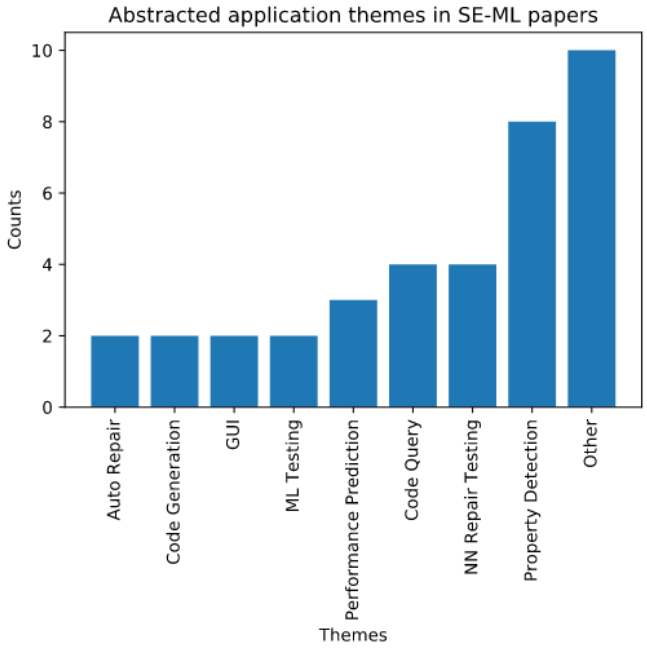


Figure 6 Count distribution of machine learning over application themes

Property Detection

Property prediction is the most recurring application theme throughout the studied papers. The theme is defined by developing a machine learning model to detect a certain property regarding a code fragment (Module, Class, Method, ...) or a documentation piece. For example [16] proposed to detect smelly code segments (i.e. need refactoring) and considered the feature-envy smelly code as a demonstration. [17] proposed to detect valid data structures - named class invariants in the community. [18] proposed to detect the type of documentation (directive, functional, non-information, …). [12] proposed to detect whether a class is thread-safe or not. [19] proposed to detect SQL vulnerabilities in PHP applications. [20] proposed to detect whether an update (git commit) is defect inducing or not. Finally [21] proposed also vulnerability detection for Java.

Neural Network Testing and Repair

The fact that deep learning models are being deployed in safety-critical applications have called to more testing for neural networks. As the software engineering community has experience in traditional software testing, new efforts were founded to apply this experience for testing deep learning models. Note that the deep learning community has used to test a deep learning model on the testing and validation dataset, but this type of testing is no more sufficient. [22] proposed a coverage criteria for testing neural networks, [23] also shared efforts on this topic by providing a more adequate test coverage criteria.

On the other side, some efforts focused on debugging deep learning models. Unlike traditional software, deep learning models bugs are latent, and fixing them is not trivial. The deep learning community used to increase the quality and the size of the training datasets to fix model errors (i.e. increase accuracy over testing dataset), however, the software engineering community is proposing new ways that can fix model errors efficiently. [24] proposed a model weight adaptation schema using an ensemble of models trained on subsets of the original dataset. [25] proposed differential analysis to find the training samples that can fix model bugs, based on computing the difference between model activation during correct classification and activation during incorrect classification.

Machine Learning Testing

The software engineering community is being also involved in the domain of testing traditional machine learning models. This theme is strictly related to the previous theme i.e. testing deep learning models, though we separated them because the earlier represents a subdomain that will grow separately than this one. Still, they share the same underlying goals. [26] proposed a directed search-based test generation methodology that reveals machine learning models discrimination against protected attributes. [27] proposed a new property of machine learning models to adhere to so that they are not discriminatory. The property is balanced data usage. To test the property, they proposed to measure a model output invariance against training dataset transformations such as permutation, variables shifting, etc.

Code Query

Code query theme is concerned with representing code or documentation (it is better to be called software artifact query) fragments in a way that makes it possible to perform some querying on them. [28] trained a custom deep learning model to detect whether 2 Go language code fragments are cloned or not. [10] performed empirical experiments to derive key design considerations in neural code search systems. Such systems take a natural language query and return a code snippet. [29] trained an RNN model that takes two software artifacts and output the probability of them being related. The goal of the model is to automatically generate software tractability matrices. [15] trained a model to measure the functional similarity between two code fragments.

Performance Prediction

Many software systems provide a set of configuration for users. However, choosing the right combination of configurations to achieve the best performance requires the internal experience of the software being used. To resolve the problem researchers are building models to predict the performance of a software system given a set of configurations. Another related problem is detecting rare events in cloud environments based on performance predictions. The small size of datasets (because generating training dataset is expensive) is one of the main barriers in the field. [11] propose the usage of deep learning models to tackle the problem and he added sparsity penalty to resolve the problem of small dataset size. [30] developed a system to predict web response time of cloud-based web applications. They preferred traditional machine learning algorithms because they generate interpretable models. Rather than focusing on increasing the accuracy of performance prediction models, authors in [31] proposed to find a high-performance configuration using Generative Adversarial Networks (GANs).

Graphical User Interface

One of the most interesting applications of machine learning in software engineering is in the field of graphical user interfaces. Thanks to deep learning high performance in computer vision tasks new ideas can be tried, which were not approachable a decade ago. [32] proposed to use CNNs with RNNs to generate a graphical user interface skeleton from raw design images. Their model achieved outstanding

accuracy in generating sound and working skeletons. [33] also used convolutional neural networks to detect text-layout bugs on mobile applications. Finding such types of bugs was not possible before convolution neural networks. This line of research is fresh and promising.

Code Generation

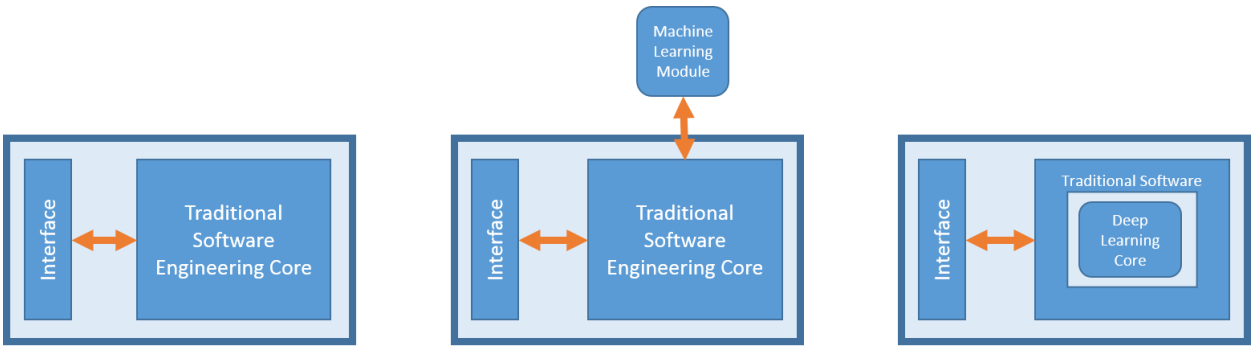
Automatic generation of code or documentation requires knowledge in the underlying structure of the targeted software artifacts. This knowledge can be tough to machines through machine learning techniques. [34] proposed a scoped N-grams approach for code completion. They have shown that carefully adapting N-grams for source code can outperform deep learning models in the code completion task. [35] proposed a combined approach of deep learning and reinforcement learning to generate sound documentation of source code.

Automatic Repair

Similar to code generation, automatic repair requires a deep understanding of code underlying structure along with error localization. Automatic repair is part of the code generation theme, however, we separated them because the automatic repair is on a higher level of hardness compared to other code generation techniques. [36] proposed a well-crafted neural machine translation model to repair recurring compilation error. They used a Google proprietary fine-grained source code snapshots dataset. Their model generates the correct repair changes for 19,314 out of 38,788 (50%) of unseen compilation errors. The correct changes are in the top three suggested fixes 86% of the time on average. They motivated their work by the empirical fact that during 2 months developers spend approximately 21 months fixing build errors. Their model is trained separately for each diagnosis. [37] proposed a combined model of deep learning and constrained based reasoning. An RNN network fixes the program syntactical errors, and the constrained based reasoning part changes the syntactically correct code based on a set of predefined rules until it achieves functional correctness. The model was applied to students introductory programming assignments. And it is trained on each assignment separately. Their model was able to repair syntax errors in 60% (8689) of submissions, and finds functionally correct repairs for 23.8% (3455) submissions.

The Rise of SE 2.0:

The initial goal of this report was to study the interaction between two assumingly separate domains Software Engineering and Machine Learning. However, this point of view to the two concepts assume that they interact externally in which each domain aids the other domains in some specific regions. This view is no more right as before. In the coming years deep learning as a subdomain of machine learning is no more an external domain from a software engineering perspective, it is becoming a core part of the software engineering practice which requires its separate research line that targets defining its different development cycle and to increase its efficiency from time, costs, and reliability as par to traditional software.



SE 0.1 SE with data products SE 0.2

Figure 7 visualizes the concept we are trying to convey. Earlier on software engineering was the practice of handcrafting algorithms and composes systems of subsystems or decompose complex ones into smaller parts. As a result, datafication of different aspects of systems have emerged. The leading software service companies in Silicon Valley have realized the revenue potentials of the collected data by utilizing machine learning techniques that generate *data products*[7]. Data products such as e-commerce recommendations, newsfeed recommendations, Ad personification, …. Machine Learning modules have become an external module that interacts with traditional software components, although being of a different nature. And finally with the rise of deep learning techniques many applications rise on them as core parts. Deep learning applications are not data products they are *full*

[7] Rachel Schutt and Cathy O'Neil. 2013. Doing Data Science: Straight Talk from the Frontline. O'Reilly Media, Inc.

products that stand alone and fulfill the entire client requirements except for some traditional components that behave as a glue to compose different components. Not that traditional systems won't exist but that a new way of creating software has emerged. In a keynote at FSE'18, Erik Meijer [38] articulated this concept as following "*In many areas, such as image recognition, natural language processing, search, recommendation, autonomous cars, systems software and infrastructure, and even Software Engineering tools themselves, Software 2.0) is quickly swallowing Software 1.0 … Instead of code as the artifact of interest, in Software 2.0 it is all about the data where compilation of source code is replaced by training models with data. This new style of programming has far-reaching consequences for traditional software engineering practices. Everything we have learned about life cycle models, project planning and estimation, requirements analysis, program design, construction, debugging, testing, maintenance, and implementation, . . . runs the danger of becoming obsolete."* Obviously, he is doing a stretch here, but the reader should get the idea.

Reference:

[1] J. Mostow, "Foreword What is AI? And What Does It Have to Do with Software Engineering?," *IEEE Trans. Softw. Eng.*, vol. SE-11, no. 11, pp. 1253–1256, Nov. 1985, doi: 10.1109/TSE.1985.231876.

[2] D. Barstow, "Artificial intelligence and software engineering," in *Proceedings of the 9th international conference on Software Engineering*, Monterey, California, USA, Mar. 1987, pp. 200–211, Accessed: May 01, 2020. [Online].

[3] K. Srinivasan and D. Fisher, "Machine Learning Approaches to Estimating Software Development Effort," *IEEE Trans. Softw. Eng.*, vol. 21, no. 2, pp. 126–137, Feb. 1995, doi: 10.1109/32.345828.

[4] D. Partridge, "Artificial intelligence and software engineering: a survey of possibilities," *Inf. Softw. Technol.*, vol. 30, no. 3, pp. 146–152, Apr. 1988, doi: 10.1016/0950-5849(88)90061-4.

[5] D. Zhang and J. J. P. Tsai, "Machine learning and software engineering," Feb. 2002, vol. 11, pp. 22–29, doi: 10.1109/TAI.2002.1180784.

[6] P. Louridas and C. Ebert, "Machine Learning," *IEEE Softw.*, vol. 33, no. 5, pp. 110–115, Sep. 2016, doi: 10.1109/MS.2016.114.

[7] K. Meinke and A. Bennaceur, "Machine Learning for Software Engineering Models, Methods, and Applications," Dec. 2017, doi: 10.1145/3183440.3183461.

[8] X. Li, H. Jiang, Z. Ren, G. Li, and J. Zhang, "Deep Learning in Software Engineering," *ArXiv180504825 Cs*, May 2018, Accessed: May 01, 2020. [Online]. Available: http://arxiv.org/abs/1805.04825.

[9] M. Fan *et al.*, "Graph Embedding Based Familial Analysis of Android Malware using Unsupervised Learning," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, May 2019, pp. 771–782, doi: 10.1109/ICSE.2019.00085.

[10] J. Cambronero, H. Li, S. Kim, K. Sen, and S. Chandra, "When deep learning met code search," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2019*, Tallinn, Estonia, 2019, pp. 964–974, doi: 10.1145/3338906.3340458.

[11] H. Ha and H. Zhang, "DeepPerf: Performance Prediction for Configurable Software with Deep Sparse Neural Network," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, May 2019, pp. 1095–1106, doi: 10.1109/ICSE.2019.00113.

[12] A. Habib and M. Pradel, "Is this class thread-safe? inferring documentation using graph-based learning," in *Proceedings of the 33rd ACM/IEEE*

International Conference on Automated Software Engineering - ASE 2018, Montpellier, France, 2018, pp. 41–52, doi: 10.1145/3238147.3238212.

[13] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstract syntax tree," in *Proceedings of the 41st International Conference on Software Engineering*, Montreal, Quebec, Canada, May 2019, pp. 783–794, doi: 10.1109/ICSE.2019.00086.

[14] S. Amreen, A. Karnauch, and A. Mockus, "Developer Reputation Estimator (DRE)," in *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, San Diego, California, Nov. 2019, pp. 1082–1085, doi: 10.1109/ASE.2019.00107.

[15] G. Zhao and J. Huang, "DeepSim: deep learning code functional similarity," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Lake Buena Vista, FL, USA, Oct. 2018, pp. 141–151, doi: 10.1145/3236024.3236068.

[16] H. Liu, Z. Xu, and Y. Zou, "Deep learning based feature envy detection," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering - ASE 2018*, Montpellier, France, 2018, pp. 385–396, doi: 10.1145/3238147.3238166.

[17] F. Molina, R. Degiovanni, P. Ponzio, G. Regis, N. Aguirre, and M. Frias, "Training Binary Classifiers as Data Structure Invariants," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, May 2019, pp. 759–770, doi: 10.1109/ICSE.2019.00084.

[18] D. Fucci, A. Mollaalizadehbahnemiri, and W. Maalej, "On using machine learning to identify knowledge in API reference documentation," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Tallinn, Estonia, Aug. 2019, pp. 109–119, doi: 10.1145/3338906.3338943.

[19] K. Zhang, "A Machine Learning Based Approach to Identify SQL Injection Vulnerabilities," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov. 2019, pp. 1286–1288, doi: 10.1109/ASE.2019.00164.

[20] G. G. Cabral, L. L. Minku, E. Shihab, and S. Mujahid, "Class imbalance evolution and verification latency in just-in-time software defect prediction," in *Proceedings of the 41st International Conference on Software Engineering*, Montreal, Quebec, Canada, May 2019, pp. 666–676, doi: 10.1109/ICSE.2019.00076.

[21] U. Koc, S. Wei, J. S. Foster, M. Carpuat, and A. A. Porter, “An Empirical Assessment of Machine Learning Approaches for Triaging Reports of a Java Static Analysis Tool,” in *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, Apr. 2019, pp. 288–299, doi: 10.1109/ICST.2019.00036.

[22] L. Ma *et al.*, “DeepGauge: multi-granularity testing criteria for deep learning systems,” in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, Montpellier, France, Sep. 2018, pp. 120–131, doi: 10.1145/3238147.3238202.

[23] J. Kim, R. Feldt, and S. Yoo, “Guiding deep learning system testing using surprise adequacy,” in *Proceedings of the 41st International Conference on Software Engineering*, Montreal, Quebec, Canada, May 2019, pp. 1039–1049, doi: 10.1109/ICSE.2019.00108.

[24] H. Zhang and W. K. Chan, “Apricot: A Weight-Adaptation Approach to Fixing Deep Learning Models,” in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov. 2019, pp. 376–387, doi: 10.1109/ASE.2019.00043.

[25] S. Ma, Y. Liu, W.-C. Lee, X. Zhang, and A. Grama, “MODE: automated neural network model debugging via state differential analysis and input selection,” in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Lake Buena Vista, FL, USA, Oct. 2018, pp. 175–186, doi: 10.1145/3236024.3236082.

[26] S. Udeshi, P. Arora, and S. Chattopadhyay, “Automated Directed Fairness Testing,” *Proc. 33rd ACMIEEE Int. Conf. Autom. Softw. Eng. - ASE 2018*, pp. 98–108, 2018, doi: 10.1145/3238147.3238165.

[27] A. Sharma and H. Wehrheim, “Testing Machine Learning Algorithms for Balanced Data Usage,” 2019, Accessed: Jul. 12, 2020. [Online]. Available: https://ris.uni-paderborn.de/publication/7635.

[28] C. Wang *et al.*, “Go-clone: graph-embedding based clone detector for Golang,” in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis - ISSTA 2019*, Beijing, China, 2019, pp. 374–377, doi: 10.1145/3293882.3338996.

[29] J. Guo, J. Cheng, and J. Cleland-Huang, “Semantically Enhanced Software Traceability Using Deep Learning Techniques,” in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, May 2017, pp. 3–14, doi: 10.1109/ICSE.2017.9.

[30] J. Chen, J. Chakraborty, P. Clark, K. Haverlock, S. Cherian, and T. Menzies, “Predicting Breakdowns in Cloud Services (with SPIKE),” *Proc. 2019 27th*

ACM Jt. Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng. - ESECFSE 2019, pp. 916–924, 2019, doi: 10.1145/3338906.3340450.

[31] L. Bao, X. Liu, F. Wang, and B. Fang, "ACTGAN: automatic configuration tuning for software systems with generative adversarial networks," in *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, San Diego, California, Nov. 2019, pp. 465–476, doi: 10.1109/ASE.2019.00051.

[32] 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 *Proceedings of the 40th International Conference on Software Engineering*, Gothenburg, Sweden, May 2018, pp. 665–676, doi: 10.1145/3180155.3180240.

[33] Y. Wang, H. Xu, Y. Zhou, M. R. Lyu, and X. Wang, "Textout: Detecting Text-Layout Bugs in Mobile Apps via Visualization-Oriented Learning," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, Oct. 2019, pp. 239–249, doi: 10.1109/ISSRE.2019.00032.

[34] V. J. Hellendoorn and P. Devanbu, "Are deep neural networks the best choice for modeling source code?," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, Paderborn, Germany, Aug. 2017, pp. 763–773, doi: 10.1145/3106237.3106290.

[35] Y. Wan *et al.*, "Improving Automatic Source Code Summarization via Deep Reinforcement Learning," *ArXiv181107234 Cs*, Nov. 2018, Accessed: Jul. 12, 2020. [Online]. Available: http://arxiv.org/abs/1811.07234.

[36] A. Mesbah, A. Rice, E. Johnston, N. Glorioso, and E. Aftandilian, "DeepDelta: learning to repair compilation errors," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2019*, Tallinn, Estonia, 2019, pp. 925–936, doi: 10.1145/3338906.3340455.

[37] S. Bhatia, P. Kohli, and R. Singh, "Neuro-symbolic program corrector for introductory programming assignments," in *Proceedings of the 40th International Conference on Software Engineering*, Gothenburg, Sweden, May 2018, pp. 60–70, doi: 10.1145/3180155.3180219.

[38] E. Meijer, "Behind every great deep learning framework is an even greater programming languages concept (keynote)," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Lake Buena Vista, FL, USA, Oct. 2018, p. 1, doi: 10.1145/3236024.3280855.