ICIC Express Letters
Part B: Applications
Volume **12**, Number **11**, November **2021**

ICIC International ©2021 ISSN 2185-2766
pp. **1019–1026**

OPEN-SOURCE MACHINE LEARNING SOFTWARE SYSTEMS: ARCHITECTURAL ANALYSIS

Mamdouh Alenezi[1] and Mohammed Akour[1,2,*]

[1]Computer Science Department
Prince Sultan University
Riyadh 11586, Saudi Arabia
malenezi@psu.edu.sa

[2]Information Systems Department
Yarmouk University
Irbid 21163, Jordan
*Corresponding author: mohammed.akour@yu.edu.jo

Received March 2021; accepted May 2021

Abstract. *Machine learning is becoming popular gradually due to the widespread use of AI in different applications ranging from user-specific applications to scientific applications. Machine learning models are usually employed in application programs with the help of open-source machine learning software systems. There is a need to analyze the architecture of such systems. In this work, we chose four different popular open-source machine learning software systems to analyze their architecture. These systems are TensorFlow, Microsoft Cognitive Toolkit, Caffe, Mxnet. We recovered their architecture through a re-engineering approach to building their UML class diagram, Package diagram using the StarUml tool. We analyzed and studied the characteristics of the architecture by using software engineering metrics from the literature. We compared and discussed the open-source machine learning software systems. It is very clear that the architecture quality was not valued heavily in the envision of these systems. In the future, we will investigate more software artifacts and software engineering metrics of these open-source machine learning software systems.*

Keywords: Machine learning, Software architecture quality, Reverse engineering, Open-source machine learning software systems

1. **Introduction.** Machine learning is one of the branches of Artificial Intelligence (AI). Usually, in this branch of AI, applications and systems are constructed by giving input to a machine learning module that provides prediction as output. The field of machine learning has accelerated the development of many fields, such as speech recognition, video classification, object recognition, and fatigue estimation [1]. Such development has resulted due to the advancements of machine learning models and developments of software platforms, which help in training sophisticated advanced models that use a huge amount of computational resources. To facilitate application for the developers and people involved in sophisticated AI applications, many open-source machine learning software systems can be found on the Internet. These applications are easy to download and use [2,3]. Due to the widespread usage of machine learning software systems, there is a growing need to understand software architecture to support tasks like re-engineering. Software architecture recovery is a technique to understand the structural characteristics and layout of the software, usually from the source code [4]. There are various techniques of software architecture recovery approaches and these are broadly classified as clustering-based techniques and pattern-based techniques. Therefore, software architecture recovery consists of methods to extract architectural information from source code.

DOI: 10.24507/icicelb.12.11.1019

The rise of artificial intelligence resulted in huge demand for AI and machine learning skills. Machine Learning (ML) based technology is now being used in almost the industries, i.e., finance, and health care. Demand for machine learning and artificial intelligence is growing exponentially. Machine learning techniques and systems are very commonly used in diverse researches [5-10].

As a result, the community has increased due to this ever-increasing demand, and this led to the evolution of AI frameworks which makes learning AI much easier. ML developers are looking for an appropriate framework for a variety of projects for ML application development. ML frameworks assist ML developers to define ML models. ML frameworks enable developers to select the pre-built model and optimize components which usually help in model building. Most of the ML frameworks are open-source. Like any other open-source software, open-source machine learning frameworks are vulnerable; hence, they can be subjected to various security attacks. Therefore, it becomes imminent to study the architecture of the machine learning framework. Therefore, in this paper, we selected popular open-source machine learning frameworks used nowadays. Furthermore, we studied the architectural characteristics of this open-source machine learning framework.

To understand the structural characteristics of famous open-source machine learning software systems, we compiled a technical report of 4 frameworks, i.e., TensorFlow, Microsoft Cognitive Toolkit, Caffe, and Mxnet. For each of the software systems, we extracted the UML class diagram, package diagram. Also, we studied and analyzed the structural characteristics of the famous software systems by using standard software engineering metrics [11].

This research paper has been organized as follows. Section 2 summarizes the related work. Section 3 presents the methodology and tools used for the architecture recovery of the open-source machine learning software systems. Section 4 discusses the experimental evaluation that includes metrics and software architecture analysis of the addressed systems. Moreover, the comparison results are presented in this section. The conclusion of the paper has been outlined in Section 5.

2. **Related Work.**

Software architecture recovery is the most widely applied technique to analyze software architecture. The architecture recovery process recovers system architecture from the artifacts like source code. This helps the software engineers to understand the whole system and evaluate it. It also helps in analyzing the probable changes of the system on the architecture. Another reason to recover the architecture is to get hold of useful information about the architecture whether it fits the desired style or whether the architecture has probable anti-patterns which are often referred to as architectural smells [12]. Such a process usually results in rearranging the architecture to fit it in the desired style. Sometimes the issues might be fixed. Usually, this comes into effect in the upcoming versions of the software system. Architecture recovery helps in analyzing the situation and helps the software engineers to judge whether they are on the right track. The process also assists in maintaining the software system. Numerous research studies have been done on recovering software architecture for analyzing the architecture. We will discuss them in the subsequent paragraphs.

In [13], the authors extracted the software architecture of the popular open-source Linux operating system. The authors think that the Linux kernel, in particular, is an excellent candidate for architectural recovery. It is said that Linux has an interesting structure. Also, Linux is growing rapidly; it is estimated that the source code is doubling every year. The authors in [14] recovered the architecture of Apache web server 1.3 for examining the complexity of the software system. The authors share the experiences of conducting the architecture recovery process that was performed by the students of the course. In [15], the authors recovered the software architecture of web applications. Moreover, the

authors developed tools that can parse and extract relations among the components of web applications. The tools use the source code to extract the artifacts.

The authors in [16] reverse engineer an application that has a significant dynamic linking within its implementation. Nautilus file manager is selected to recover the architecture. The lack of design documents makes this file manager an excellent candidate for architecture recovery. The increasing trend of using dynamic linking in open-source software leads to a wide range of opportunities for architectural analysis. Other such efforts of software architecture recovery can be found in [17] where the authors recovered the architecture of real-time train control systems. The recovered architecture helps in assessing the quality of the software architecture. In a similar work [18], the authors recovered the architecture of the subsystem of the train control system. Microservice architectural style is adopted by most of the current software applications. The authors in [19] proposed an architectural recovery tool for microservice-based applications. The authors study the complexity of the architecture. The authors in [20] recovered the software architecture of automotive embedded systems. Software architecture is recovered from the source code.

From the studies discussed above, it is evident that software architecture recovery has been performed by the software engineers to analyze the architecture. We presented architecture recovery of various heterogeneous software applications. Furthermore, in the literature, a comparative analysis of software architecture recovery techniques can be found in [21]. A set of eight architectures recovered from open-source systems was taken as model architecture to compare six state-of-the-art architectures. Metrics were used to analyze each technique with the ability to recognize the systems' architectural components and the whole architectural structure. The systems analyzed are Algorithm for Comprehension-Driven Clustering (ACDC) [22], Architecture Recovery using Concerns (ARC) [23], Bunch [24], scaLableInforMationBOttleneck (LIMBO) [25], and Weighted Combined Algorithm (WCA) [26]. The authors in [15] analyzed the source code and binaries of web applications. With the information extracted the authors performed algebraic manipulations to generate architecture diagrams highlighting the main components of web application and interactions among the components. Pattern matching and data mining techniques have been used for software architecture recovery which can be found in literature in [27,28]. With this contextual premise as a reference, the authors of this research paper have discussed the framework of software architecture recovery in the next section.

3. **Research Methodology.** In this section, we will discuss the methodology and tools used for the architecture recovery of the open-source machine learning software system presented in this paper. We used StarUML for recovering the software artifacts from the source code of the machine learning software systems. StarUML is a UML tool put forward by MKLab. It supports most of the diagrams in UML 2.0. The tool supports MDA (Model Driven Architecture) with the support of the UM profile concept, and at the same time, it enables the generation of code in multiple languages.

To study the architecture of the open-source machine learning software systems, we recovered the package diagram from source code. Apart from these artifacts, we analyzed the architecture using the software engineering metrics, see Figure 1, to describe the characteristics of the architecture [28-34]. We focus on four main aspects of the architecture, namely, size, coupling, cohesion, and complexity. For analyzing the architecture, the selected metrics are shown in Table 1.

4. **Experimental Results.** In this section, we discuss in detail the experimental evaluation and results of the addressed systems.

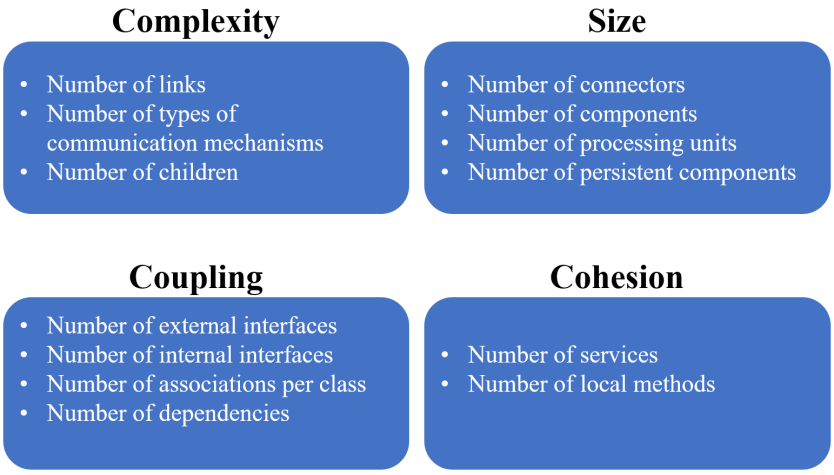

FIGURE 1. Software architecture metrics

TABLE 1. Software architecture analysis: TensorFlow

Complexity		**Size**	
Number of links	27	Number of connectors	27
Number of types of communication mechanisms	2	Number of components	11
Number of children	118	Number of processing units	0
		Number of persistent components	0
Coupling		**Cohesion**	
Number of external interfaces	0	Number of services	0
Number of internal interfaces	9	Number of local methods	544
Number of associations per class	16		
Number of dependencies	0		

4.1. **TensorFlow.** We analyzed the software architecture by using standard software engineering metrics. From the software architecture metrics, it can be observed that the architecture is not big in size, maintainable, and good. Also, the class diagram contains more than 100 children, local methods in which the class diagram is not too complex. Table 1 summarized the results of extracted software architecture metrics.

4.2. **Microsoft Cognitive Toolkit.** Under the Microsoft package, the children are four-level deep. We analyzed the software architecture of the cognitive toolkit. It can be observed from the metrics that the software architecture is not maintainable as there are less cohesion and coupling. Also, the class diagram is complex with 1577 methods, 494 children, and 62 attributes. Table 2 summarized the results of extracted software architecture metrics.

4.3. **Caffe.** There is one child under the Caffe folder. We have analyzed the software architecture. The architecture is not maintainable with no coupling and cohesion. The class diagram is also not complex with 19 methods and 11 generalizations. Table 3 summarizes the results of extracted software architecture metrics.

4.4. **Mxnet.** Under the Mxnet folder, the children are three levels deep. We analyzed the software architecture. It is observed that the architecture is less maintainable. The class diagram is not that complex with 176 methods, 76 children. Table 4 summarized the results of extracted software architecture metrics.

Table 2. Software architecture analysis: Microsoft Cognitive Toolkit

Complexity		**Size**	
Number of links	0	Number of connectors	16
Number of types of communication mechanisms	1	Number of components	0
Number of children	494	Number of processing units	0
		Number of persistent components	0
Coupling		**Cohesion**	
Number of external interfaces	0	Number of services	0
Number of internal interfaces	0	Number of local methods	1577
Number of associations per class	16		
Number of dependencies	0		

Table 3. Software architecture analysis: Caffe

Complexity		**Size**	
Number of links	0	Number of connectors	11
Number of types of communication mechanisms	0	Number of components	0
Number of children	0	Number of processing units	0
		Number of persistent components	0
Coupling		**Cohesion**	
Number of external interfaces	0	Number of services	0
Number of internal interfaces	0	Number of local methods	19
Number of associations per class	0		
Number of dependencies	0		

Table 4. Software architecture analysis: Mxnet

Complexity		**Size**	
Number of links	16	Number of connectors	16
Number of types of communication mechanisms	2	Number of components	0
Number of children	76	Number of processing units	0
		Number of persistent components	0
Coupling		**Cohesion**	
Number of external interfaces	0	Number of services	0
Number of internal interfaces	0	Number of local methods	185
Number of associations per class	16		
Number of dependencies	0		

4.5. **Systems comparisons.** The summary of the machine learning software systems is shown in Table 5. Among the open-source machine learning, software systems analyzed TensorFlow, Microsoft Cognitive Toolkit, and Mxnet are more complex within the architecture. At the same time, the architecture of the three open sources of machine learning software systems is well organized and follows object-oriented principles. TensorFlow has the highest number of lines of code with 4.5 KLOC as compared to the other open-source machine learning software systems.

Package diagrams of Mxnet, Microsoft Cognitive Toolkit, and TensorFlow are comparable as the architecture has a two-level hierarchy and it is not deeper than 2 to 3

TABLE 5. Summary of open-source machine learning software systems

Machine learning software systems	Version	Number of classes	KLOC
TensorFlow	1.14.0	117	4.5
Cognitive Toolkit	2.7	40	3.8
Caffe	1.0	11	3.6
Mxnet	1.5.0	112	3.3

levels. In addition, the number of packages in the architecture mentioned above is almost comparable. In Caffe, there are two packages.

When comparing the complexity of the open-source machine learning software systems TensorFlow, Microsoft Cognitive Toolkit, and Mxnet have the most number of local methods. Several associations per class of TensorFlow, Microsoft Cognitive Toolkit, and Mxnetis are almost similar. Both the number of local methods and associations indicate the complexity of the class diagram. This means that TensorFlow, Microsoft Cognitive Toolkit, and Mxnet have complex UML class diagrams. The number of child classes is the most in TensorFlow, Microsoft Cognitive Toolkit, and Mxnet UML class diagram. Child classes indicate the level of hierarchy within the UML class diagram. This means that the TensorFlow, Microsoft Cognitive Toolkit, and Mxnet have a deep hierarchy which shows the complex nature of the UML class diagram. Also, the number of generalizations is the most in Microsoft Cognitive Toolkit. The generalization relationship is based on a link in which one model element (child) is based on another element (parent). Hence, the number of generalizations depicts the structural complexity of the UML class diagram. This means that Microsoft Cognitive Toolkit is even more complicated in the structure besides having a high number of child classes; it also has a high number of generalizations.

By looking at the UML class diagram of all four open-source machine learning software systems, we can see that the UML class diagram of Microsoft Cognitive Toolkit and Mxnet is quite complex in structure. Furthermore, it can easily be observed from the diagram that the two have the most number of links among the classes within the architecture. And the two open-source machine learning software systems have the most number of classes within the architecture. Among the open-source, machine learning software systems Caffe has unnamed classes within the architecture. It is easy to observe that the UML class diagram is also not complex as compared to other open-source machine learning software systems. The UML class diagram of Microsoft Cognitive Toolkit has no links within the architecture. The package diagram of Mxnet has a package with the most dependencies with other packages of the architecture. Mxnet package has the most numbers of dependent packages in the architecture also op package within the architecture has most dependencies with other packages in the architecture. The package diagram of Microsoft Cognitive Toolkit has a high number of levels within the package architecture. Msra package has the most number of dependencies within the architecture. Table 6 shows a class diagram complexity summary of open source machine learning software systems.

5. **Conclusions.** In this paper, we have analyzed the software architecture of the popular open-source machine learning software systems. Star UML tools were used to recover the software artifacts from the source code. Also, we analyzed the characteristics of software architectures by using software engineering metrics from the literature. Four famous open-source machine learning software systems were selected for recovering the software artifacts. These are TensorFlow, Microsoft Cognitive Toolkit, Caffe, and Mxnet. For each of the architectures, we extracted the UML class diagram, package diagram and studied the architectural characteristics by using the proposed software engineering metrics. At the end of the study, we compared the recovered open-source machine learning software

TABLE 6. Class diagram complexity summary of open-source machine learning software systems

Metric	TensorFlow	Microsoft Cognitive Toolkit	Caffe	Mxnet
Number of connectors	27	16	11	16
Number of children	118	494	0	76
Number of local methods	544	1577	19	185
Number of associations per class	16	16	0	16

systems and found out that TensorFlow, Mxnet, and Microsoft Cognitive Toolkit are more complicated than the other architecture and more organized following the object-oriented principles.

As future work, we plan to investigate more artifacts extracted from the source code of the open-source machine learning software systems. Also, we plan to look for more metrics to describe the characteristics of software architecture. Software stability metrics are proposed in [23] for the class, architecture, and system level. It is possible to extract architectural parameters by using tools like SD metric. The methodology of extracting metrics from the tools can be found in the literature, such as in [25]. The authors used SD metric tools to assess the internal quality of the structural and behavior diagram of UML. Measurement of metrics early in the development phase leads to good quality software. Inspired by such work, we plan to extract as many characteristics as we can from the architecture by using the source code of the open-source machine learning software systems. If required, we may use tools to gain a better understanding of the architectural metrics.

REFERENCES

[1] Y. G. Gordienko, S. Stirenko, Y. P. Kochura, O. Alienin, M. Novotarskiy and N. Gordienko, Deep learning for fatigue estimation on the basis of multimodal human-machine interactions, *arXiv.org*, arXiv: 1801.06048, 2017.

[2] E. Frank and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, 2011.

[3] S. Sonnenburg, M. L. Braun, C. S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K.-R. Müller, F. Pereira and C. E. Rasmussen, The need for open source software in machine learning, *Journal of Machine Learning Research*, vol.8, no.10, pp.2443-2466, 2007.

[4] D. A. Tamburri and R. Kazman, General methods for software architecture recovery: A potential approach and its evaluation, *Empirical Software Engineering*, vol.23, no.3, pp.1457-1489, 2018.

[5] M. Akour, H. Al Sghaier and O. Al Qasem, The effectiveness of using deep learning algorithms in predicting students achievements, *Indonesian Journal of Electrical Engineering and Computer Science*, vol.19, no.1, pp.387-393, 2020.

[6] M. Akour and M. Alenezi, Test suites effectiveness evolution in open source systems: Empirical study, *Indonesian Journal of Electrical Engineering and Computer Science*, vol.19, no.2, pp.1085-1092, 2020.

[7] O. Al Qasem, M. Akour and M. Alenezi, The influence of deep learning algorithms factors in software fault prediction, *IEEE Access*, vol.8, pp.63945-63960, 2020.

[8] H. Alsghaier and M. Akour, Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier, *Software: Practice and Experience*, vol.50, no.4, pp.407-427, 2020.

[9] O. Al Qasem and M. Akour, Software fault prediction using deep learning algorithms, *International Journal of Open Source Software and Processes (IJOSSP)*, vol.10, no.4, 2019.

[10] M. Akour, O. Al Qasem, H. Alsghaier and K. Al-Radaideh, The effectiveness of using deep learning algorithms in predicting daily activities, *International Journal of Advanced Trends in Computer Science and Engineering*, vol.8, no.5, pp.2231-2235, 2019.

[11] M. Alenezi and I. Abunadi, Quality of open source systems from product metrics perspective, *International Journal of Computer Science Issues (IJCSI)*, vol.12, no.5, p.143, 2015.

[12] F. A. Fontana, V. Lenarduzzi, R. Roveda and D. Taibi, Are architectural smells independent from code smells? An empirical study, *Journal of Systems and Software*, vol.154, pp.139-156, 2019.
[13] I. T. Bowman, R. C. Holt and N. V. Brewster, Linux as a case study: Its extracted software architecture, *Proc. of the 1999 International Conference on Software Engineering*, pp.555-563, 1999.
[14] B. Grone, A. Knöpfel and R. Kugel, Architecture recovery of Apache 1.3 – A case study, *International Conference on Software Engineering Research and Practice*, 2002.
[15] A. E. Hassan and R. C. Holt, Architecture recovery of web applications, *Proc. of the 24th International Conference on Software Engineering*, pp.349-359, 2002.
[16] I. Ivkovic and M. W. Godfrey, Architecture recovery of dynamically linked applications: A case study, *Proc. of the 10th International Workshop on Program Comprehension*, pp.178-184, 2002.
[17] W. Eixelsberger, M. Kalan, M. Ogris, H. Beckman, B. Bellay and H. Gall, Recovery of architectural structure: A case study, *International Workshop on Architectural Reasoning for Embedded Systems*, pp.89-96, 1998.
[18] W. Eixelsberger, Recovery of a reference architecture: A case study, *Proc. of the 3rd International Workshop on Software Architecture*, pp.33-36, 1998.
[19] G. Granchelli, M. Cardarelli, P. Di Francesco, I. Malavolta, L. Iovino and A. Di Salle, Towards recovering the software architecture of microservice-based systems, *IEEE International Conference on Software Architecture Workshops (ICSAW)*, pp.46-53, 2017.
[20] X. Zhang, M. Persson, M. Nyberg, B. Mokhtari, A. Einarson, H. Linder, J. Westman, D. Chen and M. Törngren, Experience on applying software architecture recovery to automotive embedded systems, *Software Evolution Week – IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pp.379-382, 2014.
[21] J. Garcia, I. Ivkovic and N. Medvidovic, A comparative analysis of software architecture recovery techniques, *The 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp.486-496, 2013.
[22] V. Tzerpos and R. C. Holt, ACCD: An algorithm for comprehension-driven clustering, *Proc. of the 7th Working Conference on Reverse Engineering*, pp.258-267, 2000.
[23] J. Garcia, D. Popescu, C. Mattmann, N. Medvidovic and Y. Cai, Enhancing architectural recovery using concerns, *The 26th IEEE/ACM International Conference on Automated Software Engineering (ASE2011)*, pp.552-555, 2011.
[24] B. S. Mitchell and S. Mancoridis, On the automatic modularization of software systems using the Bunch tool, *IEEE Trans. Software Engineering*, vol.32, no.3, pp.193-208, 2006.
[25] P. Andritsos and V. Tzerpos, Information-theoretic software clustering, *IEEE Trans. Software Engineering*, vol.31, no.2, pp.150-165, 2005.
[26] O. Maqbool and H. Babri, Hierarchical clustering for software architecture recovery, *IEEE Trans. Software Engineering*, vol.33, no.11, pp.759-780, 2007.
[27] K. Sartipi, K. Kontogiannis and F. Mavaddat, A pattern matching framework for software architecture recovery and restructuring, *Proc. of the 8th International Workshop on Program Comprehension (IWPC2000)*, pp.37-47, 2000.
[28] S. Kalyanasundaram, K. Ponnambalam, A. Singh, B. J. Stacey and R. Munikoti, Metrics for software architecture: A case study in the telecommunication domain, *Proc. of IEEE Canadian Conference on Electrical and Computer Engineering*, vol.2, pp.715-718, 1998.
[29] M. Alenezi, Software architecture quality measurement stability and understandability, *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol.7, no.7, pp.550-559, 2016.
[30] M. Staron and W. Meding, A portfolio of internal quality metrics for software architects, *International Conference on Software Quality*, pp.57-69, 2017.
[31] A. Abu Hassan and M. Alshayeb, A metrics suite for UML model stability, *Software & Systems Modeling*, vol.18, no.1, pp.557-583, 2019.
[32] T. Al Hamed and M. Alenezi, Business continuity management & disaster recovery capabilities in Saudi Arabia ICT businesses, *International Journal of Hybrid Information Technology*, vol.9, no.11, pp.99-126, 2016.
[33] M. Alenezi and M. Zarour, Modularity measurement and evolution in object-oriented open-source projects, *Proc. of the International Conference on Engineering & MIS*, pp.1-7, 2015.
[34] D. Singh, An optimizating the software metrics for UML structural and behaviourl diagrams using metrics tool, *INFOCOMP Journal of Computer Science*, vol.18, no.1, pp.9-19, 2019.