Studying Software Engineering Patterns for Designing Machine Learning Systems

Hironori Washizaki
Waseda University /
National Institute of Informatics /
SYSTEM INFORMATION /
eXmotion, Tokyo, Japan
washizaki@waseda.jp

Hiromu Uchida
Waseda University
Tokyo, Japan
eagle_h.21@toki.waseda.jp

Foutse Khomh
Polytechnique Montréal
Montréal, QC, Canada
foutse.khomh@polymtl.ca

Yann-Gaël Guéhéneuc
Concordia University
Montréal, QC, Canada
yann-gael.gueheneuc@concordia.ca

***Abstract*—Machine-learning (ML) techniques are becoming more prevalent. ML techniques rely on mathematics and software engineering. Researchers and practitioners studying best practices strive to design ML systems and software that address software complexity and quality issues. Such design practices are often formalized as architecture and design patterns by encapsulating reusable solutions to common problems within given contexts. However, a systematic study to collect, classify, and discuss these software-engineering (SE) design patterns for ML techniques have yet to be reported. Our research collects good/bad SE design patterns for ML techniques to provide developers with a comprehensive classification of such patterns. Herein we report the preliminary results of a systematic-literature review (SLR) of good/bad design patterns for ML.**

***Index Terms*—Machine Learning, Architecture Patterns, Design Patterns, Anti-patterns, ML Patterns**

I. Introduction

The popularity of Machine-learning (ML) techniques has increased in recent years. They are being used in many domains, including cyber security, IoT, and autonomous cars. ML techniques rely on mathematics and software engineering. Mathematics is used to generate the algorithms, develop capabilities to learn from input data, and produce representative models. On the other hand, software engineering is employed for implementation and robust performance.

Although many works have investigated the mathematics and computer science on which ML techniques are built, few have examined their implementation, which raises many concerns. The first is the software complexity of ML techniques. The second is the quality of the available implementations, including performance and reliability. The third is the quality of the models, which may be negatively impacted by a software bug. These concerns could be alleviated if developers could demonstrate the software quality of their implementations of the ML techniques. Consequently, researchers and practitioners have been studying best practices to design ML systems and software to address issues with software complexity and quality of ML techniques. Such practices are often formalized as architecture patterns and design patterns by encapsulating reusable solutions to commonly occurring problems within the given contexts in ML systems and software design.

Many resources available on the Internet discuss various ML techniques and their concrete uses from putting together a pipeline to implementing a Markov Decision Process. Such resources are known as gray literature, and are useful to developers putting together ML systems as they provide many examples and discuss good/bad design patterns.

A study has not systematically collected, classified, and discussed these software-engineering (SE) design patterns for ML techniques, although such patterns could greatly help software developers improve ML techniques for their users. To provide developers with a comprehensive classification of such patterns, we aim to collect good/bad SE design patterns for ML techniques. Herein we report the preliminary results of a systematic literature review (SLR) of good/bad design patterns for ML. Specifically, we focus on (1) our method, (2) architecture and design (anti-)patterns, and (3) preliminary and quantitative results. We also report on ML developers' understanding of the implementation of ML techniques by answering the following research questions:

- RQ1. How do ML developers perceive and tackle the design of ML systems and software? We conducted a questionnaire-based survey, which shows that ML engineers have little knowledge of the architecture and design patterns that could assist in developing ML systems and software.
- RQ2. Does academic and gray literature address the design of ML systems and software? In our SLR of both academic and gray literature, we identified 19 academic papers and 19 gray documents. These were analyzed to extract patterns.
- RQ3. Can ML architecture and design patterns be classified? We distinguished SE patterns for ML (such as patterns for designing ML software) and non-SE patterns for ML (such as patterns for designing ML models) by analyzing the contents of the extracted documents. We classified these SE patterns with respect to two processes: the typical ML pipeline process and the typical SE process from ISO/IEC/IEEE 12207.
- RQ4. What software-engineering architecture and design patterns for ML application systems and software exist? From our collection of patterns, we confirmed that SE patterns for ML do exist and are related to different phases of the

software-development process and ML pipeline. We also provide some examples of such patterns.

The rest of this paper is organized as follows. Section II summarizes related works. Section III describes our survey and answers RQ1. Section IV presents our SLR and a subset of the results. Section V discusses our results, and Section VI concludes this paper.

II. Related Work

Previous surveys have examined general architecture and design patterns, e.g., [1]–[3], but focus mainly on object-oriented design. Surveys on architecture and design patterns exist for specific domains, e.g., multi-agent systems [4], IoT [5], or security [6]. For software engineering of ML applications, case studies, practices, and patterns are available as independent documents. However, this is the first survey and the comprehensive literature review on ML architecture and design patterns.

III. Survey

ML techniques are concrete solutions to practical problems. Hence, ML developers may already have built a body of knowledge on the good/bad design practices of ML development. RQ1 aims to validate the above assertion.

To answer RQ1, we asked 760+ developers at Japanese companies through Japan-wide mailing lists of developers and direct contact with developers during a workshop on software engineering. Developers answered three questions about SE patterns for ML systems anonymously. These questions distinguished between "patterns", which are defined as "formal practices" in any structured form, and ad-hoc practices, which are suggestions or lore shared by developers without a formal form, e.g., suggestions extracted from a blog text.

SQ1. Do you refer to existing reference architectures or architectural patterns to design your own ML systems?

SQ2. How do you acquire and elicit requirements for ML systems? Do you use patterns or practices, a template or process, or an ad-hoc method?

SQ3. How do you ensure non-functional features of ML systems? Do you use patterns or practices, a template or process, or an ad-hoc method?

Out of the 760+ contacted participants, 9 answered at least the first question for a response rate of 1%. We expected such a low participation rate of because we contacted mostly developers, who do not work with ML systems.

TABLE I
Summary of the Survey Results

SQ1.	Yes		No
	3 (General architecture, design and cloud patterns)		5
SQ2.	Patterns	Template or Process	Ad-hoc
	0	2	7
SQ3.	Patterns	Process	Ad-hoc
	1	1	6

Table I summarizes the survey results. The number of answers differs by column because some participants did not answer all three questions. Most of the developers used ad-hoc practices. If they used patterns, developers mentioned general patterns like the design patterns by Gamma et al. [7] rather than patterns dedicated to ML development.

Thus, we conclude the survey as follows:

RQ1. How do ML developers perceive and tackle the design of ML systems and software? **ML developers use either general patterns or ad-hoc patterns. They do not use patterns dedicated to SE design of ML systems and software.**

IV. Systematic Literature Review

A. Queries

We performed an SLR of both academic and gray literature to collect software-engineering good/bad design patterns for ML systems and software. For the academic literature, we chose Engineering Village, which is a search platform providing access to 12 trusted engineering document databases, such as Ei Compendex and Inspec. The Engineering Village gives us the ability to search in all recognized scholarly engineering journals, conference, and workshop proceedings over different databases with a unique search query. Moreover the Engineering Village allows us to detect and remove most of duplicates in the search results automatically. On August 14, 2019, we designed and used the following query specifying "pattern" as well as keywords related to patterns including "recipe", "workflow", "practice" and "issue" to search for documents addressing ML design practice as much as possible.

```
((((system) OR (software)) AND (machine
    learning) AND ((implementation pattern) OR
     (pattern) OR (architecture pattern) OR (
    design pattern) OR (anti-pattern) OR (
    recipe) OR (workflow) OR (practice) OR (
    issue) OR (template))) WN ALL) + ((cpx OR
    ins OR kna) WN DB) AND (({ca} OR {ja} OR {
    ip} OR {ch}) WN DT)
```

For the gray literature, we used a Google search engine on August 16, 2019, with the following query that is basically the same as that for the academic literature:

```
(system OR software) "Machine learning" (
    pattern OR "implementation pattern" OR "
    architecture pattern" OR "design pattern"
    OR anti-pattern OR recipe OR workflow OR
    practice OR issue OR template)
```

and:

```
"machine implementation pattern" OR "
    architecture pattern" OR "design pattern"
    OR anti-pattern OR recipe OR workflow OR
    practice OR issue OR template
```

We retrieved 23 academic papers (S), and snowballing elucidated 9 additional scholarly documents (A). The Google Search Engine yielded 48 gray literature documents (G).

For each document, two of the authors vetted whether it should be included in our SLR or not. We started from the titles and abstracts. Then we read the entire document

to determine whether the document pertained to software-engineering practices for ML systems. We kept 10 of the 23 scholarly papers and 25 gray documents. Of the gray documents, 6 were actual papers or books not (yet) referenced in Engineering Village and were added to set A. Below, we call these papers, books, and documents "documents". From snowballing, we kept three additional scholarly documents. Consequently, there were 10 papers s1–s10 [8]–[17] in set S, 19 (=25-6) documents g01–g19 [18]–[36] in the set G, and 9 (=3+6) documents a01–a09 [37]–[45] in set A. All the data are available on-line[1].

B. RQ2. Does academic and gray literature address the design of ML systems and software?

Our SLR indicated that ML systems are very popular thanks to the promotion of artificial intelligence in recent years. Figure 1 shows the trend in the number of documents related to SE design for ML systems in the past decade.

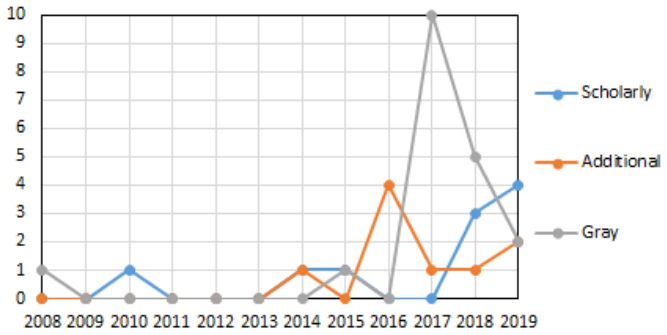

Fig. 1. Numbers of Documents per Year

ML systems are discussed in online communities from mathematics and library builders to the Maker Culture. SE development of ML systems is the subject of academic research. In gray literature, we found many documents discussing ML systems, often from the data scientists' viewpoints.

RQ2. Does academic and gray literature address the design of ML systems and software? **There are some academic documents and gray ones related to SE patterns to discuss good/bad practices of ML systems design. Although they are at different levels of abstraction, they mainly focus on data management.**

C. RQ3. Can ML architecture and design patterns be classified?

Through our SLR and reading the documents, we noted various characteristics that could help classify patterns. In the academic and gray literature, SE patterns for ML systems are often presented in the context of the ML pipeline or the SE development process.

Consequently, we identified a general pipeline and development process for each architectural and design pattern. We used the pipeline presented by Microsoft [41]. This pipeline is composed of nine stages: Model Requirements, Data Collection, Data Cleaning, Data Labelling, Feature Engineering, Model Training, Model Evaluation, Model Deployment, and Model Monitoring. For the software development process, we chose the software implementation processes specified in the ISO/IEC/IEEE 12207:2008 standard [46], which includes Requirements Analysis, Architectural Design, Detailed Design, Construction, Integration, and Qualification Testing. These pipelines and processes describe the stages and phases to develop and run any ML application system, regardless of its purpose and domain.

Herein we aim to answer the following question:

RQ3. Can ML architecture and design patterns be classified? **SE patterns for ML systems can be divided along two main dimensions: ML pipeline and SE development process.**

D. RQ4. What software-engineering architecture and design patterns for ML application systems and software exist?

Two of the authors then read half of the documents. Each author extracted patterns independently. Then the other author vetted each pattern. The extraction process identified 69 patterns. However, the vetting process reduced this to 33 patterns related to the architecture and design of ML systems. Table II shows the list of extracted ML (anti-)architecture and design patterns. Of these, 18 (55%) patterns were extracted from the scholarly papers and documents, while 15 (45%) were from the gray documents.

Then we classified each pattern according to the stages of the ML pipeline and phases of the SE development process. Table III shows our classification. The architecture patterns and design patterns pertain mostly to the Architectural Design, Detailed Design, and Construction phases. Some pertain to later phases. On the other hand, most patterns are associated with the later stages of the ML pipeline. We explain this observation by focusing on architecture patterns and design patterns that mostly impact later stages.

Thus, we aim to answer the following question:

RQ4. What software-engineering architecture and design patterns for ML application systems and software exist? **Several architecture patterns and design patterns currently exist. Some patterns are applicable to many stages of the pipeline or many phases of the development process. However, others are only applicable to one stage or phase only.**

V. Discussions

Here we describe two extracted patterns and discuss the threats to validity. For brevity, we omit information about the participants, collaborations, implementation, sample code, and known uses.

A. Example of Architectural Pattern

a) Pattern Name: s10a: Distinguish Business Logic from ML Model (originally named as "Multi-Layer Architectural Pattern" [17])

b) Intent: Isolate failures between business logic and ML learning to help developers debug ML application systems.

[1]http://www.washi.cs.waseda.ac.jp/iwesep19/

TABLE II
EXTRACTED PATTERNS ("ANTI?" DENOTES ANTI-PATTERNS)

Source	ID	Pattern Name	Anti?
[9]	s02a	Glue Code	Y
[9]	s02b	Wrap Black-Box Packages into Common APIs	
[9]	s02c	Pipeline Jungles	Y
[9]	s02d	Design Holistically about Data Collection and Feature Extraction	
[9]	s02e	Dead Experimental Codepaths	Y
[9]	s02f	Reexamine Experimental Branches Periodically	
[9]	s02g	Abstraction Debt	Y
[9]	s02h	Parameter-Server Abstraction	
[9]	s02i	Plain-Old-Data Type Smell	Y
[9]	s02j	Descriptive Data Type for Rich Information	
[9]	s02k	Multiple-Language Smell	Y
[9]	s02l	Undeclared Consumers	Y
[10]	s03a	Decouple Training Pipeline from Production Pipeline	
[10]	s03b	ML Versioning	
[12]	s05	Isolate and Validate Output of Model	
[17]	s10a	Distinguish Business Logic from ML Models	
[17]	s10b	Gateway Routing Architecture	
[40]	a04	Separation of Concerns and Modularization of ML Components	
[19]	g02a	Federated Learning	
[19]	g02b	Secure Aggregation	
[22]	g05	Handshake or Hand Buzzer	
[24]	g07a	Test Infrastructure Independently from ML	
[24]	g07b	Reuse Code between Training Pipeline and Serving Pipeline	
[25]	g08	Data-Algorithm-Serving-Evaluator	
[26]	g09	Closed-Loop Intelligence	
[27]	g10	Canary Model	
[28]	g11	Event-driven ML Microservices	
[29]	g12	Daisy Architecture	
[32]	g15a	Big Ass Script Architecture	Y
[29], [32]	g15b	Microservice Architecture	
[31], [33], [44]	g16	Data Lake	
[34]	g17	Kappa Architecture	
[31], [35], [45]	g18	Lambda Architecture	

c) Also Known As: Machine Learning System Architectural Pattern for Improving Operational Stability.

d) Motivation: ML systems are complex because their ML components must be (re)trained regularly and have an intrinsic non-deterministic behavior. Similar to other systems, the business requirements for these systems as well as ML algorithms change over time.

e) Structure: Define clear APIs between traditional and ML components. Place business and ML components with different responsibilities into three layers (Fig. 2). Divide data flows into three.

f) Applicability: Any ML systems with outputs that depend on ML techniques.

g) Consequences: Decoupling "traditional" business and ML components allows the ML components to be monitored and adjusted to meet users' requirements and changing inputs.

B. Example of Design Anti Pattern

a) Pattern Name: s02c: Pipeline Jungle [9]

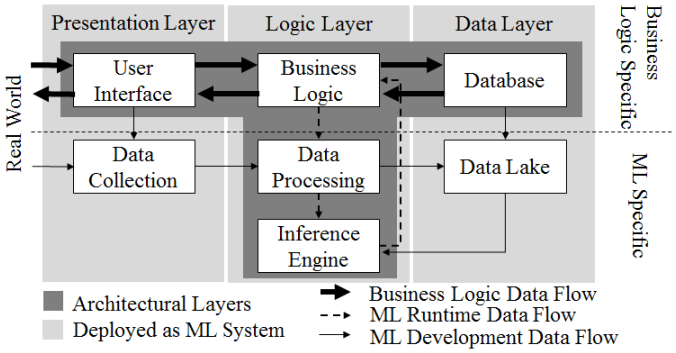

Fig. 2. Structure of Distinguish Business Logic from ML Model pattern [17]

b) Intent: Maintain one controllable, straightforward pipeline of ML components.

c) Motivation: ML systems combine several ML components with different input and output formats. These components interact with business components.

d) Problem: ML systems may include "glue code" to scrape, join, and sample input/output data into one pipeline. As this pipeline is fragile, it must be maintained and tested carefully. Glue code is a technical debt that can prevent further innovations. Additionally, testing requires expensive end-to-end integration tests.

e) Refactored Solution: Define unit and component tests. If possible, convert input/output files into first-class objects and glue code into clear APIs.

f) Applicability: Any ML application system using different techniques.

g) Related Patterns: s02a: Avoid Glue Code, s02b: Wrap Black-Box Packages into Common APIs, s02d: Design Holistically about Data Collection and Feature Extraction

C. Threats to Validity

Surveys have threats to their construct, internal, external, and conclusion validities. It is possible that our survey does not ask relevant or answerable questions due to its construct. Internally, the questions of our survey could be contradictory or misleading. However, Section III demonstrates the relevance and answerability of our study as well as alleviates this issue of being contradictory. Externally, our questions and their answers may not be generalizable to other participants or domains. However, our questions are general and do not assume any particular domain. Finally, it is possible that incorrect conclusions are drawn from the survey responses. However, our conclusions are consistent with the data.

SLRs can also [47] have threats to the internal validity and reliability of the results. One threat to the internal validity is the cause-effect conclusion that we drew from the SLR process and the results. To address this, we provide evidence from the data for each of the research questions. Another threat is the reliability of the quality and rigor in which the SLR was conducted. Section IV details our research steps and provides quantitative data for each step. Additionally, all of our data is available online. Another threat to reliability is

TABLE III
CLASSIFICATION OF THE IDENTIFIED PATTERNS

	Model Requirements	Data Collection	Data Cleaning	Data Labelling	Feature Engineering	Model Training	Model Evaluation	Model Deployment	Model Monitoring
Requirements Analysis						a04	a04	a04, g11	a04, g11
Architectural Design	g02a, g02b, g05, g09					s03a, a04, g02a, g02b	g07a, a04	g07a, g07b, s03a, s05, s10b, a04, s02a, s02b, s02c, s02d, s02e, s02f, g10, g11, g02a, g02b, g05, g08, g09, g15a, g15b, g16, g17, g18, s10a	s05, s10b, a04, g10, g09, g11, g16, g17, g18
Detailed Design							s02e, s02f, g10	g07b, s02a, s02b, s02i, s02j, g10, g11	s02i, s02j, g11
Construction							s02e, s02f	g07b, s02a, s02b, s02c, s02d, s02e, s02f, s02g, s02h, s02i, s02j, s02k, g11	s02i, s02j, g11
Integration						s03a	s02e, s02f	g07b, s03a, s05, s10b, s02e, s02f, s02l, g10	s05, s10b, s02l, g10
Qualification Testing							g07a, s02e, s02f	g07a, s03b, s02e, s02f	s03b

that an independent third-party has not vetted all the identified (anti)patterns. To address this, we intend to participate in Writers' Workshops at the Pattern Languages of Programs (PLoP) conference series[2] in order to receive the community's feedback on each pattern before publishing them.

VI. CONCLUSION

This study collects, classifies, and analyzes software-engineering architectural and design (anti-)patterns for ML systems to bridge the gap between traditional software systems and ML systems with respect to architecture and design. A survey of software developers and an SLR of ML systems and software in both academic and gray literature answer the four research questions. RQ1 confirms that SE developers are concerned with the complexity of ML systems and their lack of knowledge of the architecture and design (anti-)patterns. RQ2 demonstrates that the gray literature contains more (anti-)patterns than in the academic literature. RQ3 shows that SE patterns for ML systems are divided between two processes: the ML pipeline and SE development. RQ4 provides SE patterns for designing ML systems and some examples.

As a future work, we will complete our classification of the SE patterns for ML systems and produce a map of these patterns. Since not all patterns identified in this paper are originally written well like shown in the example subsection, we will write down all patterns in the pattern format. We will also investigate the impact of SE patterns on the quality attributes of ML systems. We will submit these patterns to the Writers' Workshop at the PLoP conference series. For further validation of identified ML patterns, we will contact developers only who work with ML systems.

ACKNOWLEDGEMILENT

The authors would like to thank Prof. Naoshi Uchihira, Mr. Norihiko Ishitani, Dr. Takuo Doi, Dr. Shunichiro Suenaga, Mr. Yasuhiro Watanabe and Prof. Kazunori Sakamoto for their helps. This work was supported by JST-Mirai Program Grant Number JP18077318, Japan.

[2]https://hillside.net/conferences/

REFERENCES

[1] P. Avgeriou and U. Zdun, "Architectural patterns revisited - A pattern language," in *EuroPLoP' 2005, Tenth European Conference on Pattern Languages of Programs, Irsee, Germany, July 6-10, 2005*, 2005, pp. 431–470.

[2] A. Ampatzoglou, S. Charalampidou, and I. Stamelos, "Research state of the art on gof design patterns: A mapping study," *Journal of Systems and Software*, vol. 86, no. 7, pp. 1945–1964, 2013. [Online]. Available: https://doi.org/10.1016/j.jss.2013.03.063

[3] B. B. Mayvan, A. Rasoolzadegan, and Z. G. Yazdi, "The state of the art on design patterns: A systematic mapping of the literature," *Journal of Systems and Software*, vol. 125, pp. 93–118, 2017. [Online]. Available: https://doi.org/10.1016/j.jss.2016.11.030

[4] J. Juziuk, D. Weyns, and T. Holvoet, "Design patterns for multi-agent systems: A systematic literature review," in *Agent-Oriented Software Engineering - Reflections on Architectures, Methodologies, Languages, and Frameworks*, 2014, pp. 79–99. [Online]. Available: https://doi.org/10.1007/978-3-642-54432-3_5

[5] H. Washizaki, N. Yoshioka, A. Hazeyama, T. Kato, H. Kaiya, S. Ogata, T. Okubo, and E. B. Fernández, "Landscape of iot patterns," in *Proceedings of the 1st International Workshop on Software Engineering Research & Practices for the Internet of Things, SERP4IoT@ICSE 2019, Montreal, QC, Canada, May 27, 2019.*, 2019, pp. 57–60. [Online]. Available: https://dl.acm.org/citation.cfm?id=3354013

[6] P. Ponde and S. Shirwaikar, "An exploratory study of the security design pattern landscape and their classification," *IJSSE*, vol. 7, no. 3, pp. 26–43, 2016. [Online]. Available: https://doi.org/10.4018/IJSSE.2016070102

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns – Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley, 1994.

[8] R. Shams, "Developing machine learning products better and faster at startups," *IEEE Engineering Management Review*, vol. 46, pp. 36–39, 09 2018.

[9] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2015, pp. 2503–2511. [Online]. Available: http://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems

[10] C. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. M. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, "Machine learning at facebook: Understanding inference at the edge," in *25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019, Washington, DC, USA, February 16-20, 2019*, 2019, pp. 331–344. [Online]. Available: https://doi.org/10.1109/HPCA.2019.00048

[11] C. Renggli, B. Karlas, B. Ding, F. Liu, K. Schawinski, W. Wu, and C. Zhang, "Continuous integration of machine learning models with ease.ml/ci: Towards a rigorous yet practical treatment," *CoRR*, vol. abs/1903.00278, 2019. [Online]. Available: http://arxiv.org/abs/1903.00278

[12] M. Kläs and A. M. Vollmer, "Uncertainty in machine learning applications: A practice-driven classification of uncertainty," in *Computer Safety, Reliability, and Security - SAFECOMP 2018 Workshops, ASSURE, DECSoS, SASSUR, STRIVE, and WAISE, Västerås, Sweden, September 18, 2018, Proceedings*, 2018, pp. 431–438. [Online]. Available: https://doi.org/10.1007/978-3-319-99229-7_36

[13] A. Ahmed, U. Abdullah, and M. J. Sawar, "Software architecture of a learning apprentice system in medical billing," Independent Researchers, Tech. Rep., 2010.

[14] S. Bethard, P. V. Ogren, and L. Becker, "Cleartk 2.0: Design patterns for machine learning in UIMA," in *Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC 2014, Reykjavik, Iceland, May 26-31, 2014.*, 2014, pp. 3289–3293. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2014/summaries/218.html

[15] S. Nalchigar, E. S. K. Yu, Y. Obeidi, S. Carbajales, J. Green, and A. Chan, "Solution patterns for machine learning," in *Advanced Information Systems Engineering - 31st International Conference, CAiSE 2019, Rome, Italy, June 3-7, 2019, Proceedings*, 2019, pp. 627–642. [Online]. Available: https://doi.org/10.1007/978-3-030-21290-2_39

[16] Q. Liu, P. Li, W. Zhao, W. Cai, S. Yu, and V. Leung, "A survey on security threats and defensive techniques of machine learning: A data driven view," *IEEE Access*, vol. 6, pp. 12 103–12 117, 02 2018.

[17] H. Yokoyama, "Machine learning system architectural pattern for improving operational stability," in *IEEE International Conference on Software Architecture Companion, ICSA Companion 2019, Hamburg, Germany, March 25-26, 2019*, 2019, pp. 267–274. [Online]. Available: https://doi.org/10.1109/ICSA-C.2019.00055

[18] "Scaling machine learning at uber with michelangelo," https://eng.uber.com/scaling-michelangelo/.

[19] "Federated learning: Collaborative machine learning without centralized training data," https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

[20] "Design patterns for deep learning," http://www.deeplearningpatterns.com/doku.phphttps://www.deeplearningpatterns.com/doku.php?id=overview.

[21] "Design patterns for machine learning in production," https://www.slideshare.net/0xdata/design-patterns-for-machine-learning-in-production.

[22] "Patterns (and anti-patterns) for developing machine learning systems," https://www.usenix.org/legacy/events/sysml08/tech/rios_talk.pdf.

[23] "The mvc for machine learning: Data-model-learner (dml)," https://hackernoon.com/the-mvc-for-machine-learning-data-model-learner-dml-8127d793f930.

[24] "Rules of machine learning: Best practices for ml engineering," https://developers.google.com/machine-learning/guides/rules-of-ml/#your_first_objective.

[25] "A design pattern for machine learning with scala, spray and spark," https://www.youtube.com/watch?v=hhXs4AOGRpI.

[26] "Closed-loop intelligence: A design pattern for machine learning," https://msdn.microsoft.com/en-us/magazine/mt833408.

[27] "A design pattern for explainability and reproducibility in production ml," https://www.parallelm.com/a-design-pattern-for-explainability-and-reproducibility-in-production-ml/.

[28] "Top trends: Machine learning, microservices, containers, kubernetes, cloud to edge. what are they and how do they fit together?" https://mapr.com/blog/top-technology-trends-machine-learning-event-driven-microservices-dataops-and-cloud-to-edge/.

[29] "Daisy architecture," https://datalanguage.com/features/daisy-architecture.

[30] "Event-driven architecture," https://www.daitan.com/wp-content/uploads/2017/11/Daitan_Whitepaper_Event-Driven_Architecture.pdf.

[31] "Demystifying data lake architecture," https://www.datasciencecentral.com/profiles/blogs/demystifying-data-lake-architecture.

[32] "Exploring development patterns in data science," https://www.theorylane.com/2017/10/20/some-development-patterns-in-data-science/.

[33] "Architecture of data lake," https://datascience.foundation/sciencewhitepaper/architecture-of-data-lake.

[34] "From insights to value - building a modern logical data lake to drive user adoption and business value," https://www.slideshare.net/Hadoop_Summit/from-insights-to-value-building-a-modern-logical-data-lake-to-drive-user-adoption-and-business-value.

[35] "Lambda architecture pattern," https://hub.packtpub.com/lambdaarchitecture-pattern/.

[36] "Busting event-driven myths," https://www.infoworld.com/article/3269207/busting-event-driven-myths.html.

[37] C. Hill, R. Bellarny, T. Erickson, and M. M. Burnett, "Trials and tribulations of developers of intelligent systems: A field study," in *2016 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2016, Cambridge, United Kingdom, September 4-8, 2016*, 2016, pp. 162–170. [Online]. Available: https://doi.org/10.1109/VLHCC.2016.7739680

[38] T. Seymoens, F. Ongenae, A. Jacobs, S. Verstichel, and A. Ackaert, "A methodology to involve domain experts and machine learning techniques in the design of human-centered algorithms," in *Human Work Interaction Design. Designing Engaging Automation - 5th IFIP WG 13.6 Working Conference, HWID 2018, Espoo, Finland, August 20-21, 2018, Revised Selected Papers*, 2018, pp. 200–214. [Online]. Available: https://doi.org/10.1007/978-3-030-05297-3_14

[39] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014.*, 2014, pp. 583–598. [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu

[40] M. S. Rahman, E. Rivera, F. Khomh, Y. Guéhéneuc, and B. Lehnert, "Machine learning software engineering in practice: An industrial case study," *CoRR*, vol. abs/1906.07154, 2019. [Online]. Available: http://arxiv.org/abs/1906.07154

[41] S. Amershi, A. Begel, C. Bird, R. DeLine, H. C. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: a case study," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019*, 2019, pp. 291–300. [Online]. Available: https://doi.org/10.1109/ICSE-SEIP.2019.00042

[42] M. H. Nguyen, D. Crawl, T. Masoumi, and I. Altintas, "Integrated machine learning in the kepler scientific workflow system," in *International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA*, 2016, pp. 2443–2448. [Online]. Available: https://doi.org/10.1016/j.procs.2016.05.545

[43] L. N. Smith and N. Topin, "Deep convolutional neural network design patterns," *CoRR*, vol. abs/1611.00847, 2016. [Online]. Available: http://arxiv.org/abs/1611.00847

[44] S. Gollapudi, *Practical Machine Learning*. Packt Publishing, 2016.

[45] A. Basak, *Stream Analytics with Microsoft Azure: Real-time data processing for quick insights using Azure Stream Analytics*. Packt Publishing, 2017.

[46] International Organization for Standardization, "ISO/IEC 12207:2008 Information technology – Software life cycle processes, institution = International Organization for Standardization," ISO/IEC, Tech. Rep., 2017.

[47] X. Zhou, Y. Jin, H. Zhang, S. Li, and X. Huang, "A map of threats to validity of systematic literature reviews in software engineering," in *Proceedings of the* 23^{rd} *Asia-Pacific Software Engineering Conference*, Dec 2016, pp. 153–160.