Software Engineering Design Patterns for Machine Learning Applications

Hironori Washizaki
Waseda University / National Institute of Informatics / SYSTEM INFORMATION / eXmotion

Foutse Khomh
Polytechnique Montréal

Yann-Gaël Guéhéneuc
Concordia University

Hironori Takeuchi
Musashi University

Naotake Natori
Aisin Corporation

Takuo Doi
Lifematics Inc.

Satoshi Okuda
Japan Advanced Institute of Science and Technology

***Abstract*—**
Practitioners and researchers study best practices to design machine learning (ML) application systems and software to address quality and constraint problems. Such practices are often formalized as design patterns. In this study, a multi-vocal literature review identified 15 software engineering design patterns for ML applications. A questionnaire survey inquired about ML developers' use of the ML design patterns to validate them in practice. 118 ML developers responded to our survey. Results show that developers were unfamiliar with most of the patterns, although there are several major patterns already used by 20+% of the respondents. For all patterns, most of the respondents would consider using them in future designs. As the respondents became more consistent in their approach to design problems by reuse, the pattern usage ratio increased. These findings suggest that there are opportunities to increase the patterns' adoption in practice by raising awareness of such patterns within the community.

THE POPULARITY OF machine learning (ML) techniques has increased in recent years. ML is used in many domains, including cyber security, the internet of things (IoT), and autonomous cars. ML techniques rely on mathematics and software engineering. The former generates algorithms, develops capabilities to learn from input data, and produces representative models. The latter is employed for implementation and performance.

Many works have investigated the mathematics and computer science on which the ML tech-

 Published by the IEEE Computer Society

niques are built, but few have examined implementation. This situation raises concerns such as the complexity of ML techniques and the quality of the available implementations, which software defects may negatively impact. These concerns should be alleviated if developers could demonstrate the software quality of their implementations. Consequently, researchers and practitioners study best practices to design ML application systems and software to address issues with software complexity and the quality of ML techniques. Such practices are often formalized as design patterns. These patterns encapsulate reusable solutions to commonly occurring problems within ML application design.

There are surveys and case studies on practices and practitioners' insights on ML systems development in general [1], [2], [3]. However, none of them focus on the use of concrete ML design patterns.

Herein we report the results of a multivocal literature review of design patterns for ML. Based on the results, we report on developers' perceptions to validate these patterns in practice. Preliminary literature review results and preliminary study on practitioners' perceptions were presented at [4], [5]. In this paper, we examined all patterns and conducted a large-scale in-depth study on developers' perceptions. We also describe one major ML design pattern to show how the ML design patterns are documented and used for resolving design problems.

ML Design Patterns in the Literature

We define "software engineering patterns for ML application systems and software design" (hereafter, "ML design patterns") as any patterns that include design structure directly or address design concerns of ML software systems indirectly. We performed a multivocal literature review of both academic and gray literature to collect them.

For the academic literature, we chose Engineering Village, which is a search platform that provides access to 12 engineering document databases, such as Ei Compendex and Inspec. Engineering Village can search all recognized scholarly engineering journals, conferences, and workshop proceedings with a unique search query. On August 14, 2019, we designed and used the following query specifying "pattern" as well as keywords related to patterns to search for documents addressing ML design practice. Based on the broad definition of ML design patterns in this paper, we included relevant keywords such as "implementation pattern" and "architecture pattern" since they may handle design concerns indirectly.

```
((((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 on August 16, 2019. The query was 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 32 scholarly documents and 48 gray literature documents. Two of the authors examined whether each document should be included in our review using the following criteria: Documents written in English addressing concrete software-engineering patterns or practices to design ML application systems and software should be included. Documents focusing on design of ML techniques and algorithms should be excluded. This process identified 19 scholarly documents and 19 gray documents. All the data are available at [6]. Among these documents, there was no paper published at PLoP series that are conferences on patterns and pattern languages, although PLoP series proceedings published at ACM have been included in the search using the Engineering Village.

Figure 1 shows the trend in the number of documents related to the design of ML application systems in the past decade. ML application

systems have recently become popular due to the promotion of artificial intelligence (AI). Since 2008, academic and gray documents have discussed good practices of ML application design.

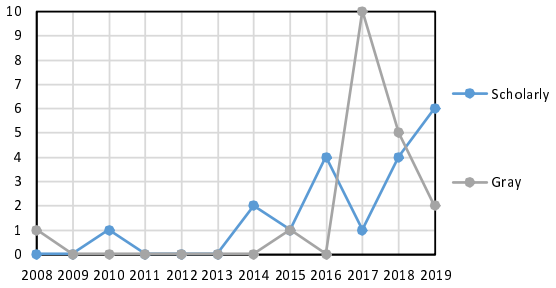

Figure 1. Number of documents per year

Overview and Classification of ML Design Patterns

Two of the authors each read half of the documents. Each author extracted patterns independently. Then one of the authors checked each pattern by reading the entire document to determine whether the pattern pertained to software engineering design practices for ML systems. The extraction process identified 69 patterns. However, the checking process reduced this to 33 patterns related to the architecture and design of ML systems. Finally, three industrial ML developers reviewed the 33 candidates from the viewpoint of practical usefulness. During the review process, any disagreement has been resolved by discussion. They identified only 15 ML design patterns (Table 1). The remaining 18 candidates were not identified as ML design patterns due to their unclear descriptions or shortage of information supporting their usefulness.

Not all of the identified ML design patterns are well-documented in a standard pattern format, which includes a clear problem statement and the corresponding solution descriptions. Thus, we described most of these ML design patterns in a standard pattern format so that practitioners can easily (re)use them in their contexts [7], [8], [9].

Through our literature review and reading of the documents, we noted various characteristics that could help classify ML design patterns. Figure 2 shows an abstract structural overview of ML applications consisting of models, data, and infrastructures. Based on the overview, we classified these ML design patterns into three categories according to their scopes (Table 2):

- P_1–P_6 are ML system topology patterns that define the entire system architecture.
- P_7–P_{10} are ML system programming patterns that define the design of a particular component.
- P_{11}–P_{15} are ML model operation patterns that focus on ML models.

Topology patterns can be regarded as architecture patterns that handle unique architectural rules specific to ML systems, while programming patterns can be seen as design patterns that are relatively less specific to ML. Nevertheless, the programming patterns still address some design characteristics of ML software in addition to the general design characteristics. Model operation patterns are all specific to ML models.

Furthermore, any design pattern should address one or more quality attributes that are associated with design problems. For ML design patterns, we assumed that the following product quality attributes defined in ISO/IEC 25010:2011 as well as model and prediction quality attributes can be addressed.

- System and software product quality attributes: Functional suitability, performance efficiency (denoted as ‘E’ in the table), compatibility (C), usability, reliability (R), security (S), maintainability (M), and, portability (P)
- ML model and prediction quality attributes: Model robustness (Mr), model explainability (Me), prediction accuracy (Pa), and, prediction fairness

One of the authors analyzed the quality attributes by reading problems and solutions descriptions of the 15 ML design patterns and identifying related specific descriptions or keywords (e.g., “If … has data dependency, it is difficult to localize the erroneous part” implies maintainability). Then, four of the authors reviewed and confirmed the result. Many ML design patterns address maintainability (Table 2). Most operation patterns address model and prediction quality attributes. Since no ML design pattern addresses usability and prediction fairness, these attributes are excluded from the table.

Table 1. Extracted ML design patterns

ID	Pattern Name	Problem (excerpt)	Solution (excerpt)
P_1	Different Workloads in Different Computing Environments [8], [10]	It is necessary to separate and quickly change the ML data workload and stabilize the training workload to maximize efficiency.	Physically isolate different workloads to separate machines. Then optimize the machine configurations and the network usage.
P_2	Distinguish Business Logic from ML Models [7], [11]	The overall business logic should be isolated as much as possible from the ML models so that they can be changed/overridden as necessary without impacting the rest of the business logic.	Separate the business logic and the inference engine, loosely coupling the business logic and ML-specific dataflows.
P_3	ML Gateway Routing Architecture [11]	When a client uses multiple services, it can be difficult to set up and manage individual endpoints for each service.	Install a gateway before a set of applications, services, or deployments. Use application layer routing requests to the appropriate instance.
P_4	Microservice Architecture for ML [7], [12]	ML applications may be confined to some "known" ML frameworks, missing opportunities for more appropriate frameworks.	Define consistent input and output data. Provide well-defined services to use for ML frameworks.
P_5	Lambda Architecture for ML [9], [13]	Real-time data processing requires scalability, fault tolerance, predictability, and other qualities. It must be extensible.	The batch layer keeps producing views at every set batch interval while the speed layer creates the relevant real-time/speed views. The serving layer orchestrates the query by querying both the batch and speed layer, and then merges them.
P_6	Kappa Architecture for ML [9], [14]	It is necessary to deal with huge amount of data with less code resource.	Support both real-time data processing and continuous reprocessing with a single stream processing engine.
P_7	Data Lake for ML [7], [13]	We cannot foresee the kind of analyses that will be performed on the data and which frameworks will be used to perform such analyses.	Store data, which range from structured to unstructured, as "raw" as possible into a data storage.
P_8	Separation of Concerns and Modularization of ML Components [2]	ML applications must accommodate regular and frequent changes to their ML components.	Decouple at different levels of complexity from the simplest to the most complex.
P_9	Encapsulate ML Models within Rule-base Safeguards [8], [15]	ML models are known to be unstable and vulnerable to adversarial attacks, noise in data, and data drift overtime.	Encapsulate functionality provided by ML models and appropriately deal with the inherent uncertainty of their outcomes in the containing system using deterministic and verifiable rules.
P_{10}	Discard PoC Code [16]	The code created for Proof of Concept (PoC) often includes code that sacrifices maintainability for efficient implementation of trial and error, and code that is ultimately no longer needed.	Discard the code created for the PoC and rebuild maintainable code based on the findings from the PoC.
P_{11}	Parameter-Server Abstraction [16]	For distributed learning, widely accepted abstractions are lacking.	Distribute both data and workloads over worker nodes, while the server nodes maintain globally shared parameters, which are represented as vectors and matrices.
P_{12}	Data Flows Up, Model Flows Down [17]	Standard ML approaches require centralizing the training data on one machine or in a datacenter.	Enable mobile devices to collaboratively learn a shared prediction model in the cloud while keeping all the training data on the device as federated learning.
P_{13}	Secure Aggregation [17]	The system needs to communicate and aggregate model updates in a secure, efficient, scalable, and fault-tolerant way.	Encrypt data from each mobile device in collaborative learning and calculate totals and averages without individual examination.
P_{14}	Deployable Canary Model [18]	A surrogate ML that approximates the behavior of the best ML model must be built to provide explainability.	Run the explainable inference pipeline in parallel with the primary inference pipeline to monitor prediction differences.
P_{15}	ML Versioning [1], [7], [10], [16]	ML models and their different versions may change the behavior of the overall ML applications.	Record the ML model structure, training dataset, training system and analytical code to ensure a reproducible training process and an inference process.

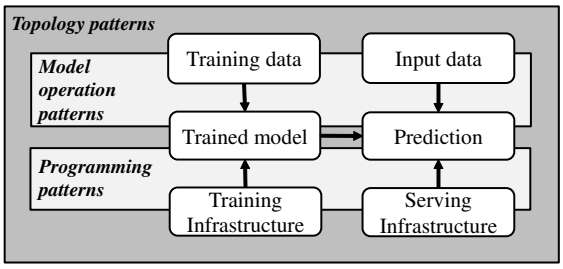

Figure 2. Machine learning system overview and categories of ML design patterns

Table 2. Classification of ML design patterns

ID	E	C	R	S	M	P	Mr	Me	Pa
Category: Topology									
P_1	E				M				
P_2					M				
P_3		C			M				
P_4		C			M	P			
P_5	E		R						
P_6	E		R						
Category: Programming									
P_7	E	C			M				
P_8					M				
P_9			R	S					
P_{10}					M				
Category: Model operation									
P_{11}	E		R						
P_{12}	E						Mr		Pa
P_{13}				S			Mr		Pa
P_{14}			R					Me	
P_{15}					M		Mr		Pa

Engineers' Perceptions

ML techniques are concrete solutions to practical problems. Hence, ML developers may have already built a body of knowledge on good design practices of ML development. To clarify how ML developers perceive existing ML design patterns, we surveyed 600+ software and ML developers who participated in an online seminar on ML design patterns in March 2021. During the seminar, we explained the concept of software patterns and introduced the 15 ML design patterns. Afterwards developers answered the following questions about reuse practices and patterns anonymously.

- SQ1. How do you solve and share design challenges of ML application systems?
- SQ2. (For each pattern) Have you ever referred to this ML design pattern?

Of the 600+ participants, 118 answered our questionnaire, which corresponds to a response rate of approximately 20%. Table 3 summarizes the survey result of SQ1. Of the 118 respondents, 37 (i.e., 31%) organized design patterns and past design results. Then they reused them to resolve ML design problems. These are the most mature practices in terms of design solution reuse. Thirty-one (26%) reused externally documented patterns but were not organizing patterns or past results by themselves. These are the second most mature practices in terms of reuse. 37 (31%) resolved problems in an adhoc way without reusing patterns. These are the worst practices.

Table 3 also shows numbers of ML design patterns used and the usage ratios, which are calculated by $\#Patterns\_used/(\#Respondents \times 15)$. For example, 37 respondents organized patterns, and they answered that in total they had used 64 patterns. They had the opportunity to use 37×15 patterns in total, so their pattern usage ratio is 11.5% ($= 64/(37 \times 15)$). As respondents become more organized in their approach to design problems by reuse, the pattern usage ratio increased, as shown in the table, from 6.3% to 10.8% and even to 11.5%. Based on the result, it is expected that development teams and organizations will reuse more ML design patterns to resolve design problems effectively and efficiently as they become more consistent in their reuse approach.

Figure 3 summarizes the result of SQ2. The most used patterns were P_{15} (used by 24% of the respondents), P_4 (21%), and P_{10} (15%). In terms of the use rate calculated by $\#Used/\#Knew$, P_2 was the most frequently used pattern with a use rate=15/28=0.54. On the other hand, no respondents actually used P_{12} or P_{13}, although some respondents knew of these patterns. There is a threat to validity that some participants might answer "yes" for patterns that they generally know and have used the essential part of the patterns' problems and solutions, while some might answer the same but have less understanding (or even worse, misunderstanding) of the patterns. Nevertheless, the survey result should help grasp the general tendency of acceptance of ML design patterns.

In terms of quality attributes, our previous survey targeting 300+ developers showed that maintainability is most considered among non-functional attributes during their ML system developments [5]. And the most used patterns P_{15}, P_4, and P_{10} commonly address maintainability, as

Table 3. Survey result of SQ1 (N=118)

Design solution and reuse practice	# Respondents	# Patterns used	Pattern usage ratio
Organizing patterns and past design results and reusing them	37	64	11.5%
Reusing externally documented patterns	31	50	10.8%
Resolving problems in an adhoc way without patterns	37	35	6.3%
Other (incl. those with little experience in ML system development)	13	3	1.5%

shown in Table 2. Their frequent use may suggest that they consider them effective for improving maintainability since maintainability was reported to be their primary concern.

The developers were unfamiliar with most ML design patterns, although there were several major patterns used by 20+% of the respondents. For all patterns, most respondents indicated that they would consider using them in future designs. These findings suggest that the identified ML design patterns are expected to help resolve particular problems, and there are opportunities to utilize existing ML design patterns and realize more consistent reuse by increasing awareness of such patterns within the ML community.

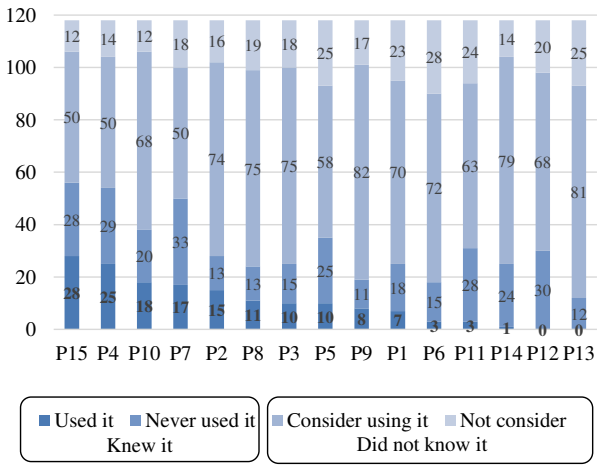

Figure 3. Survey result of SQ2 of seminar participants (N=118): Knew and used it, Knew but never used it, Did not know but will consider using it in the future, and Did not know and will not consider using it in the future

Example of a Major ML Design Pattern

Here, we describe one major ML design pattern and its usage. We selected "Distinguish Business Logic from ML Model" (P_2) since it was one of the most popular patterns among our survey participants. Moreover, it provides a basis for other patterns (such as P_7 "Data Lake for ML") by clearly decomposing the ML system into multiple layers and components. For brevity, participants, collaborations, implementation, and known uses are omitted below.

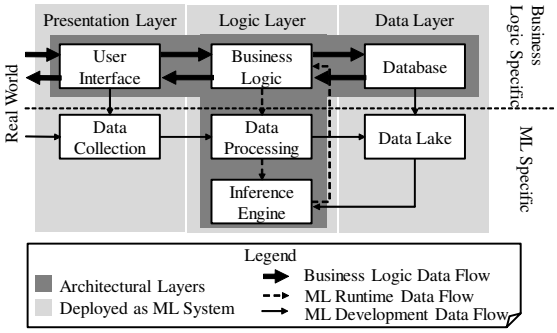

Figure 4. Structure of "Distinguish Business Logic from ML Model" pattern [11]

Pattern Name

Distinguish Business Logic from ML Model [7] (original name "Multi-Layer Architectural Pattern" [11])

Intent

Isolate failures between business logic and ML learning layer to help developers debug ML application systems easily.

Problem

ML application 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 and the ML algorithms change over time.

Solution

Define clear APIs between the traditional and ML components. Place the business and ML components with different responsibilities into three layers (Fig. 4). Divide data flows into three.

Applicability

It is applicable to any ML application system with outputs that depend on ML techniques.

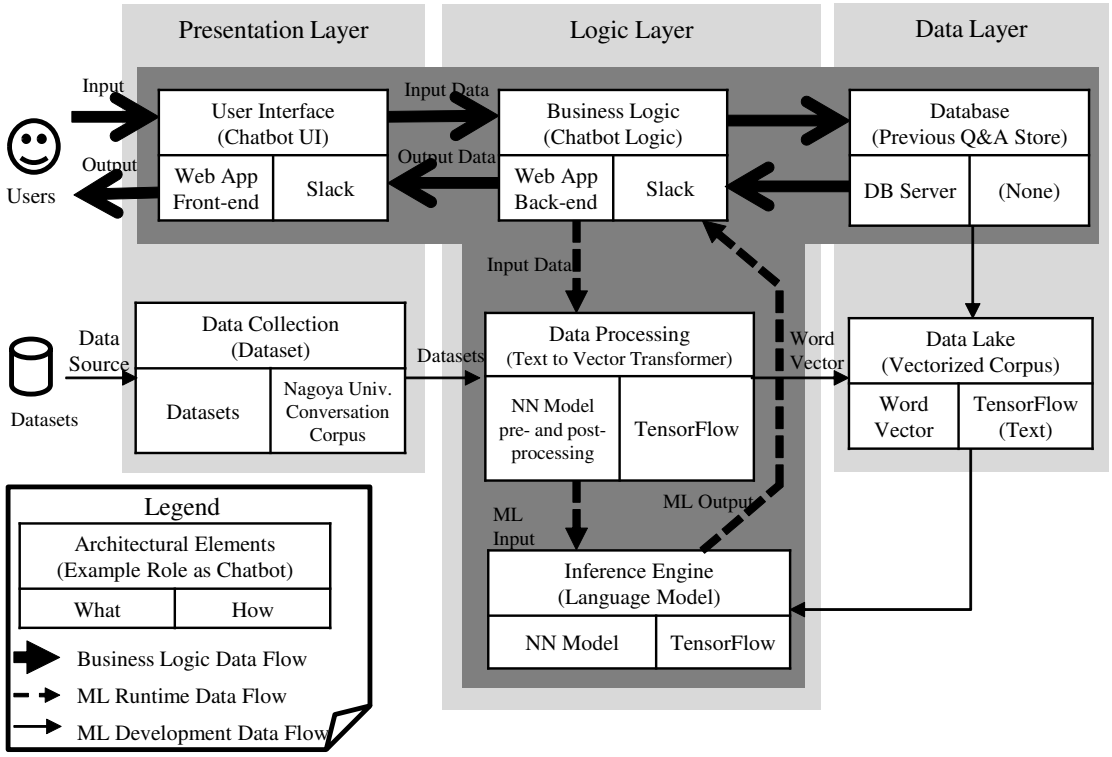

Figure 5. Example of Chatbot system architecture by applying "Distinguish Business Logic from ML Model"

Consequences

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

Usage Example

Figure 5 presents an implementation example of the pattern in a Slack-based Chatbot system. By referring to the pattern, the necessary elements as well as their relationships are easily specified while having clear separation between the Chatbot service (as the business logic) and the underlying ML components.

Conclusion

ML application systems are quite popular due to the recent promotion of AI. To bridge the gap between traditional software systems and ML application systems with respect to design, software-engineering design patterns for ML applications were analyzed via a multivocal literature review and a survey of developers. From the 32 scholarly documents and 48 gray documents identified in the literature review, 15 ML design patterns were identified. A survey of developers revealed that there are some major ML design patterns.

Although the literature review was conducted in 2019, we believe this is the first step to explore ML design patterns and build on them to propose refined patterns. We plan to revise the identified patterns by sharing them to obtain reviews from the public. Since we surveyed the practitioners' perceptions on the patterns 2-5 years after their original publications, our survey should reveal the meaningful perceptions. And practitioners may not be aware of recent patterns such as [19], [20], which have emerged after our literature review. As our future work, we will continue our survey by extending the scope of ML design patterns to include these recent patterns and ones published at some of the PLoP series proceedings, which were not included in the original search.

We also plan to create a map of the relationships among these ML design patterns and other related patterns. Furthermore, we will investigate applications of these ML design patterns in actual ML systems and software design.

Acknowledgement

The authors would like to thank Mr. Hiromu Uchida for his help. This work was supported by JST-Mirai JPMJMI20B8, JSPS JPJSBP 120209936, and KAKENHI 21KK0179.

REFERENCES

1. S. Amershi *et al.*, "Software engineering for machine learning: a case study," in *41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2019, pp. 291–300.
2. M. S. Rahman *et al.*, "Machine learning software engineering in practice: An industrial case study," *CoRR*, vol. abs/1906.07154, 2019.
3. A. Serban *et al.*, "Adoption and effects of software engineering best practices in machine learning," in *International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, 2020, pp. 3:1–3:12.
4. H. Washizaki *et al.*, "Studying software engineering patterns for designing machine learning systems," in *10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*. IEEE, 2019, pp. 49–54.
5. ——, "Practitioners' insights on machine-learning software engineering design patterns: a preliminary study," in *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 797–799.
6. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5168886
7. H. Washizaki *et al.*, "Software engineering patterns for machine learning applications (sep4mla)," in *9th Asian Conference on Pattern Languages of Programs (AsianPLoP)*. Hillside, Inc., 2020, pp. 1–10.
8. ——, "Software engineering patterns for machine learning applications (sep4mla) - part 2," in *27th Conference on Pattern Languages of Programs (PLoP)*. Hillside, Inc., 2020, pp. 1–10.
9. J. Runpakprakun *et al.*, "Software engineering patterns for machine learning applications (sep4mla) - part 3," in *(under review)*, 2021, pp. 1–9.
10. C. Wu *et al.*, "Machine learning at facebook: Understanding inference at the edge," in *25th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE Computer Society, 2019, pp. 331–344.
11. H. Yokoyama, "Machine learning system architectural pattern for improving operational stability," in *International Conference on Software Architecture Companion (ICSA-C)*. IEEE Computer Society, 2019, pp. 267–274.
12. D. Smith, "Exploring development patterns in data science," https://www.theorylane.com/2017/10/20/some-development-patterns-in-data-science/, 2017.
13. P. Menon, "Demystifying data lake architecture," https://www.datasciencecentral.com/profiles/blogs/demystifying-data-lake-architecture, 2017.
14. V. Tyagi, "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, 2017.
15. M. Kläs and A. M. Vollmer, "Uncertainty in machine learning applications: A practice-driven classification of uncertainty," in *Computer Safety, Reliability, and Security (SAFECOMP) Workshops*, 2018, pp. 431–438.
16. D. Sculley *et al.*, "Hidden technical debt in machine learning systems," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*. Neural Information Processing Systems Foundation, 2015, pp. 2503–2511.
17. B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," https://ai.googleblog.com/2017/04/federated-learning-collaborative.html, 2017.
18. S. Ghanta *et al.*, "Interpretability and reproducability in production machine learning applications," in *17th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018, pp. 658–664.
19. V. Lakshmanan *et al.*, *Machine Learning Design Patterns*. O'Reilly, 2020.
20. Y. Shibui, "Machine learning system design patterns," https://github.com/mercari/ml-system-design-pattern, 2020.

Hironori Washizaki is a Professor at Waseda University in Tokyo, and a Visiting Professor at National Institute of Informatics. He also works in industry as an Outside Director of SYSTEM INFORMATION and eXmotion. He serves as IEEE-CS Vice President for PEAB. Contact him at washizaki@waseda.jp.

Foutse Khomh is a Full Professor at Polytechnique Montréal and FRQ-IVADO Research Chair on Software Quality Assurance for Machine Learning Applications. He is also a member of Mila - Quebec AI Institute. Contact him at foutse.khomh@polymtl.ca.

Yann-Gaël Guéhéneuc has been a full Professor at Concordia University since 2017. Contact him at yann-gael.gueheneuc@concordia.ca.

Hironori Takeuchi is a Professor at Musashi University. Contact him at h.takeuchi@cc.musashi.ac.jp.

Naotake Natori is with Aisin Corporation. Contact him at naotake.natori@aisin.co.jp.

Takuo Doi is with Lifematics Inc. Contact him at doi@lifematics.co.jp.

Satoshi Okuda is with the Japan Advanced Institute of Science and Technology. Contact him at okuda@jaist.ac.jp.