Software Engineering Patterns for Machine Learning Applications (SEP4MLA) - Part 2

HIRONORI WASHIZAKI, Waseda University / National Institute of Informatics / System Information / eXmotion
FOUTSE KHOMH, Polytechnique Montréal
YANN-GAËL GUÉHÉNEUC, Concordia University
HIRONORI TAKEUCHI, Muashi University
SATOSHI OKUDA, Japan Advanced Institute of Science and Technology
NAOTAKE NATORI, Aisin Seiki
NAOHISA SHIOURA, Aisin Seiki

Practitioners and researchers study best practices to develop and maintain ML application systems and software to address quality and constraint problems. Such practices are often formalized as software patterns. We discovered software-engineering design patterns for machine-learning applications by doing a thorough search of the literature available on the subject. Among the ML patterns found, we describe three ML patterns in the standard pattern format so that practitioners can (re)use them in their contexts: "Different Workloads in Different Computing Environments", "Encapsulate ML Models Within Rule-base Safeguards", and "Data Flows Up, Model Flows Down"

Categories and Subject Descriptors: I.2.6 [**Artificial Intelligence**]: Learning—*Machine learning*; D.2.11 [**Software Engineering**]: Software Architectures—*Patterns*

Additional Key Words and Phrases: Machine learning patterns

ACM Reference Format:

Washizaki, H. Khomh, F. Guéhéneuc, Y.-G. Takeuchi, H. Okuda, S. Natori, N. and Shioura, N. 2020.Software Engineering Patterns for Machine Learning Applications (SEP4MLA) - Part 2. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 0 (March 2020), 10 pages.

1. INTRODUCTION

Practitioners and researchers study best practices to develop and maintain ML application systems and software to address quality and constraint problems. Such practices are often formalized as software patterns. We call these software-engineering patterns for machine-learning applications SEP4MLA to distinguish them from non-software-engineering patterns for ML, such as patterns for designing ML models. Among various patterns related

Author's address: H. Washizaki, 3-4-1 Okubo, Shinjuku-ku, Tokyo, Japan; email: washizaki@waseda.jp; F. Khomh, Polytechnique Montréal, QC, Canada; email: foutse.khomh@polymtl.ca; Y.-G. Guéhéneuc, Concordia University, Montréal, QC, Canada; email: yann-gael.gueheneuc@concordia.ca; H. Takeuchi, Muashi University, Tokyo, Japan; email: h.takeuchi@cc.musashi.ac.jp; S. Okuda, Japan Advanced Institute of Science and Technology, Japan; email: okuda@jaist.ac.jp
N. Natori, Aisin Seiki, Japan; email:naotake.natori@aisin.co.jp
N. Shioura, Aisin Seiki, Japan; email:naohisa.shioura@aisin.co.jp

Table I. Identified ML Patterns

Category	ID	Pattern name	Abstract
Topology	P_1	**Different Workloads in Different Computing Environments**	Physically isolate different workloads to different machines; then, optimize the machine configurations and the network usage [Wu et al. 2019].
	P_2	Distinguish Business Logic from ML Models	Separate the business logic and the inference engine, loosely coupling business logic and ML-specific dataflows [Yokoyama 2019; Washizaki et al. 2020].
	P_3	ML Gateway Routing Architecture	Install a gateway before a set of applications, services, or deployments and use routing requests to the appropriate instance [Yokoyama 2019].
	P_4	Microservice Architecture	Define consistent input and output data and provide well-defined services to use for ML frameworks [Everett 2018; Smith 2017; Washizaki et al. 2020].
	P_5	Lambda Architecture	The batch layer produces views at every chosen batch interval; the speed layer creates real-time/speed views; the serving layer orchestrates and uses both layers [Menon 2017; Packt 2017; Basak 2017].
	P_6	Kappa Architecture	Support both real-time data processing and continuous reprocessing with a single stream processing engine [Tyagi 2017].
Programming	P_7	Data Lake	Store data, both structured to unstructured, as “raw” as possible into a data storage [Gollapudi 2016; Menon 2017; Singh 2019; Washizaki et al. 2020].
	P_8	Separation of Concerns and Modularization of ML Components	Decouple computations at different levels of complexity from simplest to most complex [Rahman et al. 2019].
	P_9	**Encapsulate ML Models Within Rule-base Safeguards**	Encapsulate functionality provided by ML models and handle the inherent uncertainty of their outcomes using of deterministic and verifiable rules [Kläs and Vollmer 2018].
	P_{10}	Discard PoC Code	Discard the code created for the Proof of Concept (PoC) and rebuild maintainable code based on the findings from the PoC [Sculley et al. 2015].
Model Operation	P_{11}	Parameter–Server Abstraction	Distribute both data and workloads over worker nodes; the server nodes maintain globally-shared parameters, represented as vectors and matrices [Sculley et al. 2015].
	P_{12}	**Data Flows Up, Model Flows Down**	Enable mobile devices to collaboratively learn a shared prediction model in the cloud while keeping all the training data on the devices [Google 2017].
	P_{13}	Secure Aggregation	Encrypt data from each mobile device in collaborative learning and calculate totals and averages without individual examination [Google 2017].
	P_{14}	Deployable Canary Model	Run the explainable inference pipeline in parallel with the primary inference pipeline to monitor prediction differences [Blog 2018].
	P_{15}	ML Versioning	Record the ML model structure, training dataset, training system, and analytical code to ensure reproducible training and inference processes [Wu et al. 2019; Amershi et al. 2019; Sculley et al. 2015; Washizaki et al. 2020].

to machine-learning applications, such as ML requirements engineering patterns or ML security engineering patterns, we discovered 15 software-engineering design patterns for machine-learning applications (hereafter, ML patterns) by doing a thorough search of available literature on the subject.

We grouped these ML patterns into three categories, as shown in Table 1: ML system topology patterns that define the entire system architecture, ML system programming patterns that define the design/implementation of particular components, and ML system model-operation patterns that focus on the operations of ML models.

Not all of the identified ML patterns are well-documented in standard pattern formats, which includes clear problem statements and corresponding solution descriptions. Thus, we describe these ML patterns in the standard pattern format so that practitioners can (re)use them in their contexts[1].

[1] We described four other ML patterns: “Data Lake” (P_7), “Distinguish Business Logic from ML Models” (P_2), “Microservice Architecture” (P_4) and “ML Versioning” (P_5) in [Washizaki et al. 2020].

To describe each ML pattern uniformly, we adopted the well-known Pattern-Oriented Software Architecture form (POSA) [Buschmann et al. 1996], with a discussion section to address practical considerations. It is a well-structured format and practitioners with little knowledge of patterns can easily understand their contents.

In the following, we describe three ML patterns selected from the three different categories: "Different Workloads in Different Computing Environments" (P_1), "Encapsulate ML Models Within Rule-base Safeguards" (P_9), and "Data Flows Up, Model Flows Down" (P_{12}).

2. DIFFERENT WORKLOADS IN DIFFERENT COMPUTING ENVIRONMENTS (P_1)

2.1 Source

[Hazelwood et al. 2018]

2.2 Intent

To satisfy different resource requirements and constraints of different workloads by deploying them in different computing environments.

2.3 Context

The typical ML pipeline process [Amershi et al. 2019] consists of various stages including data collection, data processing (such as data cleaning and data labelling), feature engineering, model training, model deployment, and model execution (i.e., inference and prediction) and monitoring. When deploying ML on a large scale, there are various considerations to take into account, because different workloads correspond to different stages and impose different requirements on the necessary computing resources.

For many machine learning models, the availability of extensive, high-quality data is crucial for prediction. The ability to rapidly process and feed these data to the training machines is important for ensuring fast and efficient offline training. For some ML applications, the amount of data to ingest for each training task can be huge. Also, complex preprocessing may be necessary to clean and normalize data and allow efficient transfer and easy learning. These impose high demands on all resources: storage, network, CPU, and GPU.

Quality requirements and constraints on data and software systems also impose resource requirements, varying with the workload. For example, the sensitivity and security requirements of data is often an issue. Data for training a ML model must be protected within the training environment if it contains personally-identifiable information. Data processed from user input in the inference environment might be exposed to the outside. Requirements on quality attributes, such as flexibility and scalability, can also be different by workloads.

2.4 Problem

There are four types of ML workloads with different characteristics: data collection, data processing, model training, and model execution. These workloads have different resource requirements and constraints. Thus, it is hard to optimize machine configurations if these workloads are deployed on the same machines. Such co-existence also limit some quality attributes, such as security, flexibility, and scalability.

2.5 Solution

To decouple the workloads for data collection, data processing, model training, and model execution, and isolate them on different machines to satisfy different resource requirements and constraints, and optimize for each workload. Figure 1 shows the structure of those workloads deployed on different computing environments, which requires to structure communication among those workloads as follows:

—Collector: The data collection machine(s) collects the data from data sources and store it into storage.

—Reader: The data processing machine reads the data from the storage, process and condense them, and then send to the trainer.

—Trainer: The training machine solely focuses on executing the training options rapidly and efficiently. The trained model is deployed into the predictor.

—Predictor: The prediction machine focuses on executing the deployed model to conduct inference and prediction according to the given input.

Machine configurations can be highly optimized for each different workload. High flexibility and scalability can be achieved by distribution over the network. The resource requirements for training ML models, especially deep-learning models, are often much larger than the requirements for executing that model, the trainer (and the reader) is often located in Cloud computing environments. In contrast, the predictor can be deployed closer to its users in a more resource-limited environments, such as IoT edge and mobile devices.

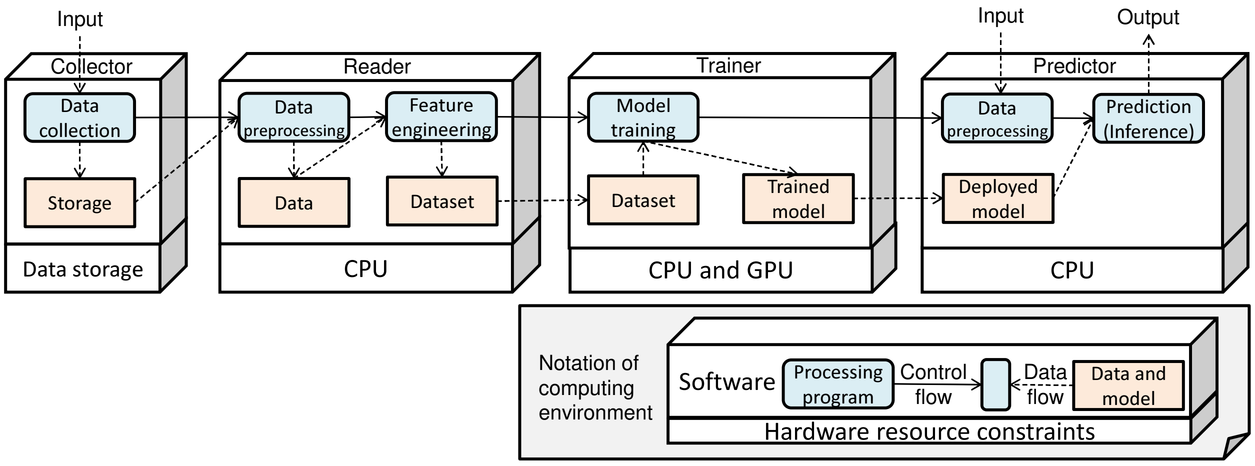

Fig. 1. Structure of the “Different Workloads in Different Computing Environments” pattern

2.6 Example

At Facebook, most of the ML training is run through the FBLearner platform [Hazelwood et al. 2018], which is a set of three tools: FBLearner Feature Store as reader, FBLearner Flow as trainer, and FBLearner Predictor as predictor. Each tool focuses on different parts of the ML pipeline, as shown in Figure 2. FBLearner has an internal job scheduler to allocate resources and schedule jobs on a shared pool of GPUs and CPUs to meet the different resource requirements of the reader, trainer, and predictor.

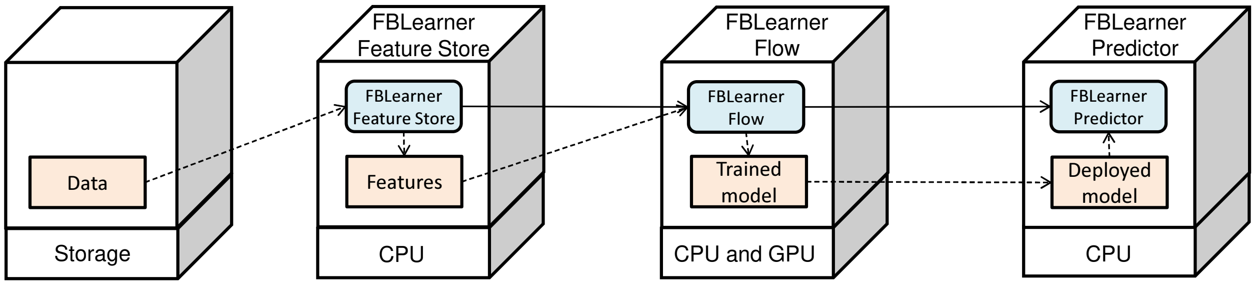

Fig. 2. Facebook’s Machine Learning Flow and Infrastructure

2.7 Discussion

The data traffic generated by training can become significant. If not handled carefully, it can easily saturate network devices, even disrupt other services. Thus, network optimization is also important, through compression, scheduling, and data/computation placement.

2.8 Related Patterns

—Data Lake [Washizaki et al. 2020] (P_7): The storage associated with the data collection workload is often implemented as a "Data Lake", which stores both structured to unstructured data.

—Data Flows Up, Model Flows Down (P_{12}): The training workloads can be distributed over clouds and mobile devices. In the pattern "Data Flows Up, Model Flows Down", if the data collected from some mobile devices is sensitive, the model is re-trained locally by using the data and the difference between the original and the re-trained model is uploaded to the cloud.

3. ENCAPSULATE ML MODELS WITHIN RULE-BASE SAFEGUARDS (P_9)

3.1 Source

[Kläs and Vollmer 2018]

3.2 Intent

To mitigate and absorb the low robustness of ML models by encapsulating ML models within rule-base safeguards.

3.3 Context

ML models are often part of highly-autonomous software systems, in which a high risk exists that humans, businesses or the environment may be harmed in case of failure.

The Sheridan-Verplanck taxonomy defines ten automation levels, which the AI applications aim or achieve [Feldt et al. 2018]. The higher the automation level, the more autonomous the AI application is in making decisions and the higher the risk. At level seven or higher, computers take decisions without getting approval from humans: "Computer makes and implements decision but must inform human after the fact" [Feldt et al. 2018].

3.4 Problem

Because of the complexity and empirical nature of ML models, no guarantee can be provided for their correctness. Thus, it is dangerous to rely on ML models in highly-autonomous systems, especially for safety-critical systems or highly-individualized or adaptive systems. Besides, ML models are unstable and vulnerable to adversarial attacks and to noise in data and they may "drift" overtime.

3.5 Solution

To encapsulate functionality provided by ML models and deal with the inherent uncertainty of their outcomes, inside the business logic, using deterministic and verifiable rules as safeguards. Figure 3 shows a structure in which the business logic, the safeguard, and the model are responsible for the followings:

—The business logic API is a Façade that wraps the corresponding ML model.

—The safeguard encapsulating the ML model is responsible for adequate risk management, taking into account the likelihood that the outcome of the model might be wrong, as well as the consequences of every decision. It could decide to consider other information sources or adapt its behavior to handle uncertain outcomes.

—The encapsulated ML model must deliver its service together with information about outcome-related uncertainty that can justifiably be trusted. It allows the system to conduct informed decisions via the business logic API.

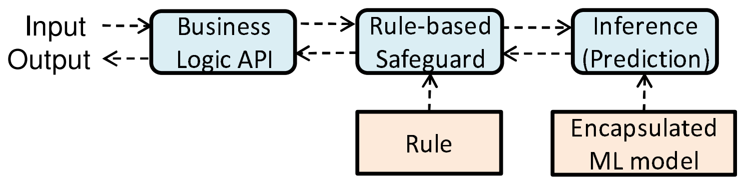

Fig. 3. Structure of the “Encapsulate ML models within rule-base safeguards” pattern

3.6 Example

In a scenario of autonomous intersection crossing using ML-based traffic sign recognition, the system could use GPS localization as an additional information source or slow down the vehicle, thereby having more time to analyze more images of the traffic situation [Kläs and Vollmer 2018].

Another scenario is a health recommender system that is individualized and adaptable to enable early detection of individual health problems [Mellin 2017]. If the change in health is slow enough, then the system may adapt to a situation that is risky and consider it to be normal. A rule-based safeguard that checks boundaries based on general knowledge would prevent such risky recommendation based on individual adaptation.

3.7 Discussion

Considering sources of uncertainty such as model fit, data quality, and scope compliance would lead to more systematic development and testing of models and systems.

Among “known-known,” “known-unknown,” “unknown-known,” and “unknown-unknown,” the rule-based safeguard is particularly useful for “known-unknown”, of which people are aware but which they do not fully understand. In “known-unknown”, there is a general problem for any kind of generalization due to the limited available data and the risk of overfitting [Milkau and Bott 2018]. A combination of heuristics and machine learning can be a helpful for active risk management [Milkau and Bott 2018], when deterministic and verifiable rules can be formulated. A rule-based safeguard does not apply to “unknown-unknown.”

3.8 Related Patterns

—ML Gateway Routing Architecture (P_3) [Yokoyama 2019]: A gateway can be installed in front of the business logic so that clients can use multiple ML-based services with safeguards while avoiding the setting and management of individual endpoints.

4. DATA FLOWS UP, MODEL FLOWS DOWN (P_{12})

4.1 Source

[Google 2017]

4.2 Intent

Decoupling the prediction and re-training from the storing in the cloud of the training data and trained ML models.

4.3 Context

Some ML applications are running on local devices, such as smart phones, cameras, and IoT devices. Such applications predict various things, such as human behavior and environmental changes, and perform their decision-making and activities in real time. The ML models are trained by data collected through the devices.

4.4 Problem

The ML application on the local devices should return the its prediction results in real time. The ML model execution should be performed on the edge and the model training in the cloud. The ML applications running on the local devices may return customized prediction results. The ML model must be re-trained with data collected through the device. Though the ML model on each device can be re-trained locally, the data collected could be stored into the cloud for more efficient re-training. Yet, the user's privacy and data confidentiality must be preserved.

4.5 Solution

To solve the problem described above, we introduce “Data Flows Up, Model Flows Down” pattern. Figure 4 illustrates the structure of this pattern. The ML model is trained in the cloud and deployed to each local device (model flows down). Using the ML application, the data for re-training is collected in each device and this data or the model re-trained locally are sent to the cloud (data flows up). Network connectivity is required between local devices and the cloud when the ML model is deployed to local devices and the data/model is sent to the cloud.

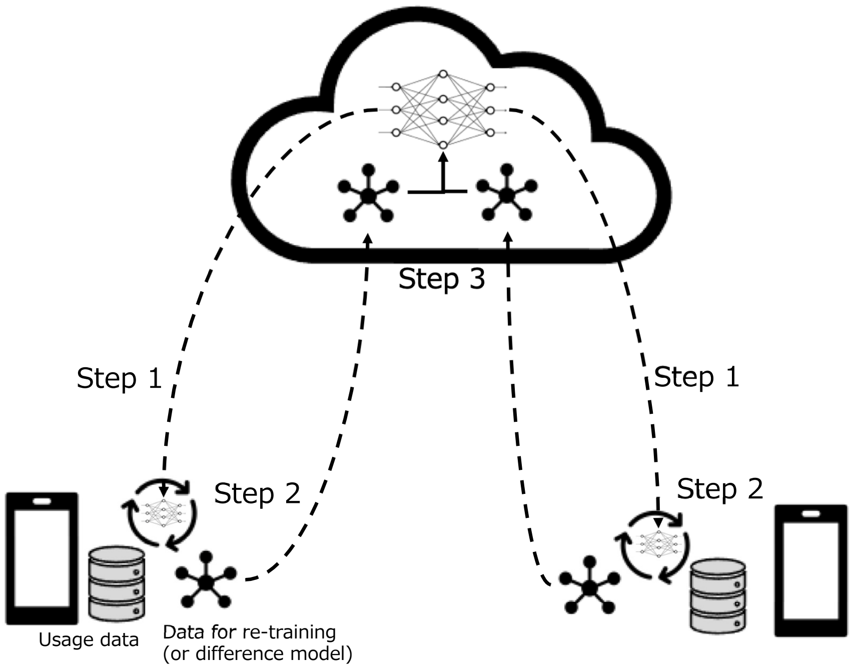

Fig. 4. Structure of the “Data Flows Up, Model Flows Down” pattern

Detailed steps of this pattern are as follows:

—The initial training is performed in the cloud; the trained ML model is deployed on the mobile devices (Step 1).

—In each mobile device, data is collected through the device (Step 2):

  —If the collected data is not private, it flows up to the cloud.

  - —If the collected data is private, the ML model is re-trained locally; the difference between the original ML model and the re-trained model is sent to the cloud.

—In the cloud, using the uploaded data, the ML model is retrained (Step 3). Difference models provided by mobile devices are averaged and updated into the ML model. Updating the ML model from difference models is a special case of this pattern, Federated Learning. The updated ML model is re-deployed to each mobile device.

4.6 Example

"Data Flows Up, Model Flows Down" is implemented in the ML system using mobile/edge devices, such as mobile phones, cameras, and IoT devices.

A major retail company in the USA introduced the ML system to identify traffic problems in their store parking lots [IBM 2020]. They trained an ML model from the image data collected by the parking-lot security cameras and deployed it at each store (model flows down). In the stores, the model is applied on real-time images from security cameras and notifications sent to the staff when traffic jams occur. When a store encountered an adversarial example, e.g., a false positive or false negative, the application sends the corresponding set of images to the cloud for model re-training (data flows up).

Federated Learning is a special case of this pattern. It is implemented in the Google Keyboard on Android called Gboard [Google 2017]. When users type texts, their phones store information about the current context and whether users selected the suggestions provided by Gboard. Gboard uses a ML model locally and re-trains the model using this stored data. Thus, Gboard reflects the users' specific behavior and suggestions improve. The difference between the original model and the re-trained models is uploaded to Google Cloud. At fixed intervals, the base ML model in the cloud is updated using these uploaded difference models and re-deployed.

The Federated Learning approach is also implemented in medical imaging systems [Wen and Rieke 2020]. To identify diseases from medical images using a ML system, huge amount of medical images are needed. Such images may not be available in a single institution but may not be allowed to be shared to respect privacy and confidentiality. Then, an initial ML model trained on a small dataset is deployed to each institution and the model is re-trained locally using private patient data. Difference models from the institutions are collected and used to update the ML model.

4.7 Discussion

When applying this pattern, we must assess the characteristics of the data collected locally. If the data contains sensitive information, we must consider whether we can send such data to the cloud directly. If we cannot store the data in the cloud, we must apply the Federated Learning approach. Before applying this Federated Learning, we must assess whether the following conditions are satisfied. First, the local devices, such as mobile phones, have enough computing resources to store the trained ML model and re-train the model. Second, the ML model can be updated by averaging difference models. Third, there are no negative effects when updating the local ML models with the re-trained, general model.

4.8 Related Patterns

—Different Workloads in Different Computing Environments (P_1): "Data Flows Up, Model Flows Down" can be seen as a special case of "Different Workloads in Different Computing Environments" from the viewpoint of computing environments.

—Secure Aggregation (P_{13}): "Secure Aggregation" is often necessary for Federated Learning on privacy-sensitive user-held data.

5. CONCLUSION

We described three patterns: "Different Workloads in Different Computing Environresments", "Encapsulate ML models within rule-base safeguards", and "Data Flows Up, Model Flows Down". These patterns are from a set of

ML patterns identified through a thorough search of the literature on patterns for machine-learning applications. We hope that these patterns can guide practitioners (and researchers) to consider how ML fits within their target contexts and design ML-based systems and software that address quality and constraint problems.

In the future, we plan to write all ML patterns in the standard pattern format to help developers adopt the good practices described by these patterns. We also plan to identify more concrete cases of these patterns in real applications. We will also create a map of the relationships among these ML patterns and other related patterns.

Acknowledgement

We are grateful to our shepherd, Kyle Brown, for his careful and valuable reviews that significantly improved this paper. We would like to thank also the participants of the writers' workshop at PLoP 2020. This work was supported by JSPS Bilateral Program JPJSBP120209936, JST-Mirai Program Grant Number JPMJMI20B8, enPiT-Pro Smart SE, FRQ, and NSERC Discovery Grants.

Received July 2020; revised August 2020; accepted

REFERENCES

Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald C. Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. 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*. ACM/IEEE, –, 291–300. `DOI:http://dx.doi.org/10.1109/ICSE-SEIP.2019.00042`

Anindita Basak. 2017. *Stream Analytics with Microsoft Azure: Real-time data processing for quick insights using Azure Stream Analytics*. Packt Publishing, –.

ParallelM Blog. 2018. A Design Pattern for Explainability and Reproducibility in Production ML. `https://www.parallelm.com/a-design-pattern-for-explainability-and-reproducibility-in-production-ml/`. (2018).

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 1996. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, –.

Julian Everett. 2018. Daisy Architecture. `https://datalanguage.com/features/daisy-architecture`. (July 2018).

Robert Feldt, Francisco Gomes de Oliveira Neto, and Richard Torkar. 2018. Ways of Applying Artificial Intelligence in Software Engineering. In *6th IEEE/ACM International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, RAISE@ICSE 2018, Gothenburg, Sweden, May 27, 2018*. ACM, –, 35–41.

Sunila Gollapudi. 2016. *Practical Machine Learning*. Packt Publishing, Birmingham, UK. `https://books.google.ca/books?id=3ywhjwEACAAJ`

Google. 2017. Federated Learning: Collaborative Machine Learning without Centralized Training Data. `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`. (2017).

Kim M. Hazelwood, Sarah Bird, David M. Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018*. IEEE Computer Society, –, 620–629. `DOI:http://dx.doi.org/10.1109/HPCA.2018.00059`

IBM. 2020. IBM Video Analytics with IBM Visual Insights. `https://www.ibm.com/downloads/cas/GJBJQM4Y`. (June 2020).

Michael Kläs and Anna Maria Vollmer. 2018. 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*. Springer, –, 431–438. `DOI:http://dx.doi.org/10.1007/978-3-319-99229-7_36`

Jonas Mellin. 2017. Can a machine learning model completely replace a rules based system? `https://www.quora.com/Can-a-machine-learning-model-completely-replace-a-rules-based-system`. (April 2017).

Pradeep Menon. 2017. Demystifying Data Lake Architecture. `https://www.datasciencecentral.com/profiles/blogs/demystifying-data-lake-architecture`. (August 2017).

Udo Milkau and Jürgen Bott. 2018. Active Management of Operational Risk in the Regimes of the "Unknown": What Can Machine Learning or Heuristics Deliver? *Risks* 6, 2 (2018), 1–16.

Packt. 2017. Lambda Architecture Pattern. `https://hub.packtpub.com/lambdaarchitecture-pattern/`. (2017).

Md Saidur Rahman, Emilio Rivera, Foutse Khomh, Yann-Gaël Guéhéneuc, and Bernd Lehnert. 2019. Machine Learning Software Engineering in Practice: An Industrial Case Study. *CoRR* abs/1906.07154 (2019), 1. `http://arxiv.org/abs/1906.07154`

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *Annual Conference on Neural Information Processing Systems*. Neural Information Processing Systems Foundation, Montréal, QC, Canada, 2503–2511. http://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems

Ajit Singh. 2019. Architecture of Data Lake. https://datascience.foundation/sciencewhitepaper/architecture-of-data-lake. (April 2019).

Daniel Smith. 2017. Exploring Development Patterns in Data Science. https://www.theorylane.com/2017/10/20/some-development-patterns-in-data-science/. (October 2017).

Vineet Tyagi. 2017. 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).

Hironori Washizaki, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2020. Software Engineering Patterns for Machine Learning Applications (SEP4MLA). In *9th Asian Conference on Pattern Languages of Programs (AsianPLoP 2020)*. Hillside, Inc., –, 1–10.

Y. Wen and N. Rieke. 2020. Federated Learning for Medical Imaging: Collaborative AI without Sharing Patient Data. https://developer.nvidia.com/gtc/2020/video/s21536-vid. (June 2020).

Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim M. Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, Tommer Leyvand, Hao Lu, Yang Lu, Lin Qiao, Brandon Reagen, Joe Spisak, Fei Sun, Andrew Tulloch, Peter Vajda, Xiaodong Wang, Yanghan Wang, Bram Wasti, Yiming Wu, Ran Xian, Sungjoo Yoo, and Peizhao Zhang. 2019. Machine Learning at Facebook: Understanding Inference at the Edge. In *25th International Symposium on High Performance Computer Architecture.* IEEE CS Press, Washington, DC, USA, 331–344. DOI:http://dx.doi.org/10.1109/HPCA.2019.00048

Haruki Yokoyama. 2019. Machine Learning System Architectural Pattern for Improving Operational Stability. In *International Conference on Software Architecture Companion*. IEEE CS Press, Hamburg, Germany, 267–274. DOI:http://dx.doi.org/10.1109/ICSA-C.2019.00055