Software Engineering Patterns for Machine Learning Applications (SEP4MLA) – Part 5 – Explainable Proxy Model

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 Corporation

Machine learning (ML) researchers and practitioners study the best practices to develop and support ML-based applications to ensure the quality and resolve constraints applied to their applications. Following best practices in software engineering, they often formalize these practices as software patterns. To clarify an overview of the current landscape of these practices and patterns, we discovered software-engineering design patterns for machine-learning applications by thoroughly searching the available literature on the subject. Among the ML patterns found, we describe in this paper one ML topology pattern, "Explainable Proxy Model", in the standard pattern format so that practitioners can (re)use it in their contexts and benefits from its advantages. The pattern addresses the problem of low explainability and reproducibility of highly accurate machine learning models by building a surrogate proxy ML model, called a canary model, which approximates the behavior of the best ML models (i.e., primary models) and monitoring deviations between canary and primary models. By describing the "Explainable Proxy Model" pattern, we make explicit its advantages and limitations as well as the contexts in which it applies.

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. and Natori, N. 2023. Software Engineering Patterns for Machine Learning Applications (SEP4MLA) - Part 5. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. V (October 2023), 9 pages.

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

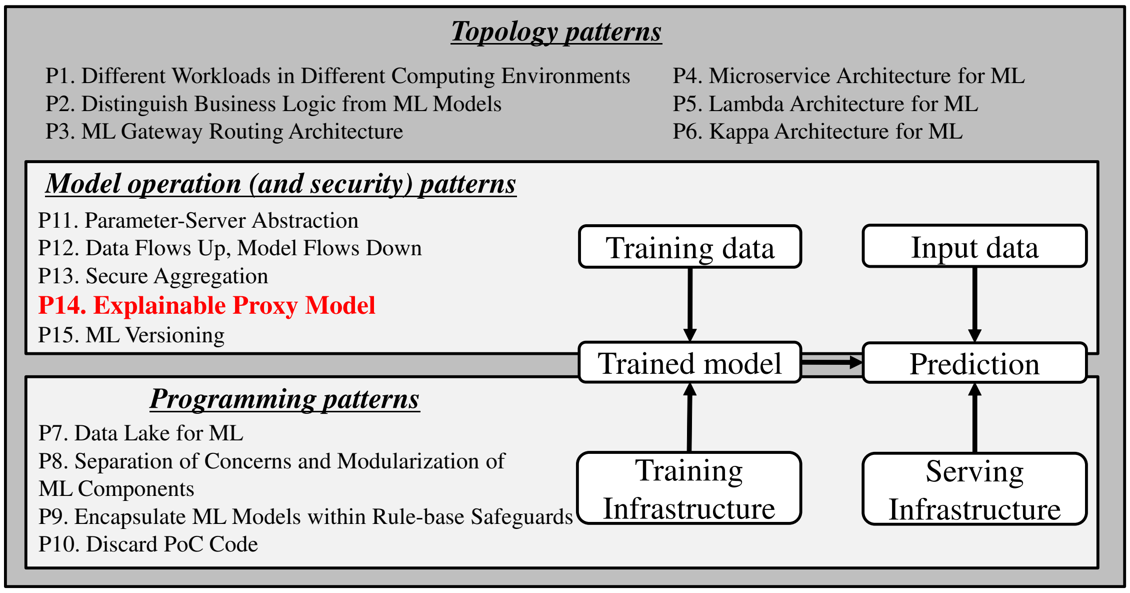
Topology patterns
P1. Different Workloads in Different Computing Environments
P2. Distinguish Business Logic from ML Models
P3. ML Gateway Routing Architecture
P4. Microservice Architecture for ML
P5. Lambda Architecture for ML
P6. Kappa Architecture for ML
Model operation (and security) patterns
P11. Parameter-Server Abstraction
P12. Data Flows Up, Model Flows Down
P13. Secure Aggregation
P14. Explainable Proxy Model
P15. ML Versioning
Programming patterns
P7. Data Lake for ML
P8. Separation of Concerns and Modularization of ML Components
P9. Encapsulate ML Models within Rule-base Safeguards
P10. Discard PoC Code
Training data
Input data
Trained model
Prediction
Training Infrastructure
Serving Infrastructure

Fig. 1. Machine learning system overview and categories of software engineering design patterns for ML

1. INTRODUCTION

Machine learning (ML) researchers and practitioners study best practices to develop and maintain ML-based applications to ensure quality and resolve the constraints on their applications. Such practices are often formalized as software patterns. We call these software-engineering design patterns for machine-learning applications, SEP4MLA, to distinguish them from patterns for ML, which are unrelated to software engineering, such as patterns for designing ML models [Lakshmanan et al. 2020] and from software engineering patterns, which relate only to the engineering of (any) software application. Among various patterns related to machine-learning applications, such as ML requirements engineering patterns or ML security engineering patterns, we discovered 15 SEP4MLA by doing a thorough search of the literature available on the subject. Details of our methodology are available in our previous work [Washizaki et al. 2020; Washizaki et al. 2022].

Figure 1 shows an abstract structural overview of ML applications consisting of models, data, and infrastructures. We group the SEP4MLA into three categories: ML applications topology patterns that define the applications architectures, ML applications programming patterns that define the design/implementation of particular components of the applications, and ML applications model-operation patterns that focus on the operations of ML models. Summaries of all patterns are available in the appendix.

Not all of the identified SEP4MLA are well-documented in standard pattern format, which includes clear context, problem statement, and corresponding solution description. Thus, we describe these SEP4MLA in a standard pattern format so that practitioners can (re)use them in their contexts.

In previous works, we described most of the patterns in Table 3: “Distinguish Business Logic from ML Models” (P_2), “Microservice Architecture for ML” (P_4), “ML Versioning” (P_5), and “Data Lake for ML” (P_7) in Part 1 [Washizaki et al. 2020]; “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}) in Part 2 [Washizaki et al. 2021]; “Lambda Architecture for ML” (P_5) and “Kappa Architecture for ML” (P_6) in Part 3 [Runpakprakun et al. 2021]; and, “ML Gateway Routing Architecture” (P_3) in Part 4 [Washizaki et al. 2022]. These are classified into three categories according to their scopes: ML system topology patterns that define the entire system architecture, ML system programming patterns that define the design of a particular component, and ML model operation patterns that focus on ML models.

The following describes the ML pattern “Explainable Proxy Model” (P_{14}) as an ML model operation pattern. The pattern contributes to increasing the explainability of target ML models, while other patterns address one or more quality attributes (except for explainability), including maintainability, associated with design problems [Juziuk et al. 2014]. To describe each ML pattern uniformly, we partially adopted the well-known Pattern-Oriented Software Architecture format (POSA) [Buschmann et al. 1996]. It is a well-structured format, and practitioners with little knowledge of patterns can easily understand its content.

2. EXPLAINABLE PROXY MODEL (P_{14})

2.1 Name

Explainable Proxy Model

2.2 Also Known As

Deployable Canary Model: This alias emphasizes that a surrogate proxy ML model can indicate problems in a primary, more complex model in production, as the status of carried small canaries provided warnings in mine tunnels with dangerous gases. Note that this pattern is different from another well-known practice, “Canary Deployment” [Fang et al. 2023], where a new version of a software application or service is incrementally released to a subset of its user base, resulting in minimization of the impact of production issues and easing rollbacks.

2.3 Source

[Ghanta et al. 2018] briefly shows a basic structure, and potential benefits of the pattern “Explainable Proxy Model.”

2.4 Intent

The pattern helps ML engineers ensure the reproducibility of their highly accurate ML models as well as good explainability.

2.5 Context

An organization wants to use a highly accurate ML-based software system in a production environment. Highly accurate ML models often have problem with low explainability and reproducibility. When evaluating a ML model to determine whether it’s ready for production, metrics like accuracy only provide data on how correct a model’s predictions are relative to ground truth values in the test set. Still, they carry no insight into why a model arrived at those predictions. In many ML scenarios, users may hesitate to accept a model’s prediction at face value [Lakshmanan et al. 2020]. Explainability is critical for practical ML usage as more and more industries use ML to make essential decisions, and everyone, from consumers to regulators, tries to determine how each algorithm makes its determination.

2.6 Problem

By the very definition and functioning of ML models, some complex ML models achieve high accuracy but have low explainability and reproducibility. When such complex highly accurate ML models that cannot be directly interpreted are in deployment, a second explainable model, called a “canary model”, is used as a proxy to approximate the highly accurate model [Ghanta et al. 2018]. We call the highly accurate ML model “primary” model to distinguish it from the canary model (i.e., proxy model). The canary model is a relatively simple model with a lower accuracy (i.e., low prediction performance) than the primary model. Due to its simplicity, the canary model is interpretable and is utilized to provide human-understandable explanations. The class of such interpretable models is typically limited to linear models, fuzzy model, or decision trees where one can present the reasoning behind predictions in a human-understandable way [Ghanta et al. 2018].

However, even straightforward explainability approaches (like canary models) can be challenging to put into production due to the complexities and variabilities in production environments. The problem is how to coordinate the primary and canary models properly in production. ML applications have unique dependencies between training and inference that grow increasingly complex, particularly with ensemble models and online training. The primary and canary models could have been trained or retrained on different pipelines. Even worse, such training could have been at various different intervals in different ways; for example, if one used an online training algorithm and another might use an offline algorithm or a transfer learning algorithm [Ghanta et al. 2018].

Such complexities and variabilities make ML software systems hard to maintain while keeping good prediction accuracy with relevant explainability and reproducibility, especially when experiencing data drift. Two inference subsystems [1], one with the primary model and the second with a canary model, are needed while avoiding high complexities and maintenance difficulty. Note that since a highly accurate prediction is mandatory in the given context, simple, explainable canary models cannot be used for making final predictions.

2.7 Solution

Run the explainable canary (i.e., proxy) inference subsystem in parallel with the primary inference subsystem using the same data/input and monitor prediction differences, resulting in providing explainability and reproducibility while maintaining high accuracy in production. Figure 2 shows the structure of the solution architecture. The architecture structurally consists of the following components and features [Ghanta et al. 2018].

- —There must be the primary inference subsystem and the canary inference subsystem, together with retraining pipelines for each model as well as a pipeline to compare the results of primary and canary, and mechanisms to trigger when canary outputs and primary outputs mismatch so that potential issues in canary-based explainability can be detected. The complex primary model achieves better accuracy necessary for the final actual prediction, while the simple canary model as a proxy of the primary model is interpretable but has low prediction accuracy.
- —A reproducibility mechanism is required to replay prediction inputs and outputs so that alternative means can be used to track what happened in cases where the canary cannot be used to explain the primary.
- —The entire system should provide replay dashboards to let engineers examine in depth any execution, including executions during which explainability alerts occurred.
- —It is necessary to provide configuration and operational functionalities to let ML and operation engineers specify and control the followings:
  - —how the subsystems and pipelines should run, such as batch processing and REST-based implementation,

[1] Since ML inference components may have different architectures, such as batch pipelines and online services, we call them inference "subsystems" rather than inference "pipelines" to consider such possible differences in architectures.

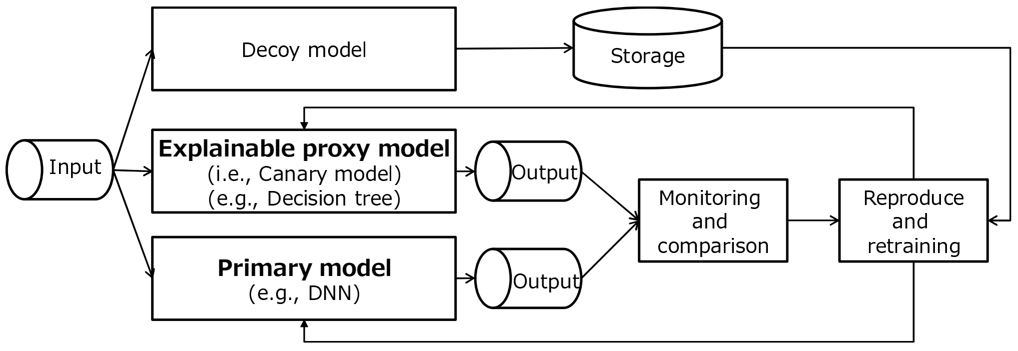

Fig. 2. Structure of the "Explainable Proxy Model"

—how frequently to retrain both primary and canary models,
—how to decide if new models are pushed to production inference for either primary or canary,
—how to compare the outputs and generate an alert if the primary and the canary mismatch.

The behavioral process of training, monitoring, replaying, and retraining proceeds as follows [Ghanta et al. 2018].

(1) During runtime, the same data/input is sent to both primary and canary inference subsystems as well as the storage (via an optional decoy model) to archive all inputs and parameters/conditions. The models used in these subsystems may also need to be periodically retrained.
(2) The system collects statistics on each subsystem once the ML application runs in the deployment environment and maintains a full lineage and temporal sequence of all datasets, computations, events, and results occurring in all pipelines.
(3) The system generates explainability alerts if the primary and canary display significant deviation in prediction patterns. Deviation can be identified by continuously measuring and tracking the root mean square error (RMSE) between the predictions in real time with a proper configurable threshold. Then, reproduce the entire sequence for examination to ensure that the actual computation and any non-deterministic runtime factors are fully understood.
(4) By having the features mentioned above, if an explainability alert occurs, and the two inference subsystems (i.e., primary and canary) are no longer providing similar predictions, the temporal sequence can be used to determine exactly how the primary subsystem generated any particular prediction, and if necessary replay that prediction. There are various possible reasons for alerts, such as data drift in the input data distribution lowering the prediction performance of the canary model, as well as low robustness in the prediction of the primary model. After examining the result and the reason, the primary and canary models can be retrained or replaced by other potentially better models (particularly if the canary models have low prediction accuracy), or other explainability techniques can be applied.

2.8 Example

The pattern has been applied to various ML-based software systems in production. For example, Figure 3 (adopted from [Takeuchi et al. 2020]) presents a typical implementation of the pattern in an ML-based image inspection system used in a factory. In this example, images as input data are captured by cameras installed over the transportation line. Multiple processing units play the roles of the canary subsystem as well as the primary one to be compared with each other to identify significant deviations by the main unit. When a deviation alert occurs, offline replay and retraining will be conducted to revise models.

2.9 Known Uses

The pattern has been used indirectly and directly in many cases, including the following.

—In typical Explainable Artificial Intelligence (XAI) developments, an additional explainable surrogate module is added to the learned original complex model to achieve a more transparent and trustworthy model by comparing the complex, accurate model with the explainable one and monitoring their difference [Yang et al. 2022]. Such explainable surrogate adoption can be seen as a primitive abstract structure of the concept of this pattern, while the pattern provides a more concrete solution with technical details about how to achieve such comparison and monitoring in production.
—The implementation of the pattern described in the example section is taken from an actual ML-based image inspection system operated at a Japanese factory. In this case, a processing unit for self-diagnosis plays the role of the canary (i.e., explainable proxy) in order to identify significant deviations compared with the primary model.

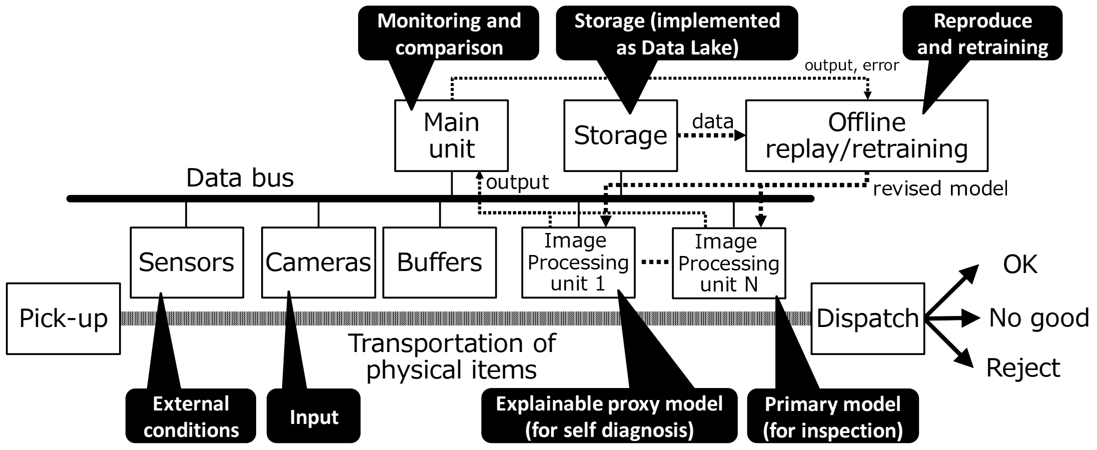

Fig. 3. Example of factory ML-based image inspection system architecture by applying “Explainable Proxy Model”

—ParallelM, which is now a part of DataRobot, has provided an MLOps infrastructure named MCenter [Fleming 2020]. Based on the pattern, the infrastructure can handle primary and canary models to monitor deviations and reproducing, as well as training features with a visual dashboard. [Ghanta et al. 2018] reported the infrastructure has been adopted in many uses.

2.10 Consequences

The proposed pattern can be used for deploying canary explainability in production, which can be used with a wide range of ML models and captures the full degree of explainability, reproducibility, and alerting needed to deliver explainability in production.

As a potential negative consequence, the implementation of the pattern requires additional storage. Furthermore, additional complexity to handle all subsystems and pipelines has to be managed well, although it should be relatively low compared with the situation of having explainable canary models independently in an ad-hoc way.

The matching of two outputs from the primary and canary models may not guarantee the reasoning of the complex deep learning models. Furthermore, it can be challenging to choose simpler explainable models if the primary models are more complex. In these cases, post hoc explainability methods to approximate the relationships between the primary model’s features and its output may increase explainability, although post hoc explainability methods are not always trustworthy; for example, explanations by using SHAP, which is one of the well-accepted methods to extract model-agnostic local feature attributions, are known not robust and can be manipulated when it comes to explaining the difference in outcomes between groups [Laberge et al. 2023].

2.11 See also

The pattern is mainly related to the following patterns.

—Data Lake for ML (P_7) [Gollapudi 2016; Menon 2017; Singh 2019; Washizaki et al. 2020]: The storage associated with the input (and output) data collection is often implemented as a “Data Lake for ML”, which stores both structured and unstructured data.

—ML Versioning (P_{15}) [Wu et al. 2019; Amershi et al. 2019; Sculley et al. 2015; Washizaki et al. 2020]: To properly address replay and retraining processes, it is necessary to manage canary and primary model revisions. “ML Versioning” provides a way to record the ML model structure, training dataset, training system, and analytical code to ensure a reproducible training process and an inference process.

—Explainable Predictions [Lakshmanan et al. 2020]: To support choosing proper explainable models for canary models, “Explainable Predictions” introduces simpler models that are often interpretable by design. The pattern also introduces post hoc explainability methods to approximate the relationships between a complex model’s features and its output if the primary models are more complex.

3. CONCLUSION

In this paper, we described the ML pattern “Explainable Proxy Model”, which we chose in a set of SEP4MLA identified through a thorough search of the literature on patterns for machine-learning applications. We hope that this pattern can guide practitioners (and researchers) to consider how ML applications fit within their target contexts and design ML-based applications with the required quality.

In the future, we plan to write all SEP4MLA in a standard pattern format to help developers adopt good practices in the development of ML applications. We also plan to identify more concrete occurrences of these patterns in real applications. We will also create a map of the relationships among these SEPMLA and other patterns, often across categories.

Appendix

The following table shows summaries of all 15 discovered software-engineering design patterns for machine-learning applications (SEP4MLA).

Acknowledgement

We are grateful to our shepherd, Alfredo Goldman, for their careful and valuable reviews that significantly improved this paper. Also, we would like to thank all participants of the writers’ workshop at PLoP 2023. This work was supported by JSPS Bilateral Program JPJSBP120209936, JSPS KAKENHI 21KK0179, and JST-Mirai Program Grant Number JPMJMI20B8.

Received July 2023; revised August and September 2023; accepted September 2023

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

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

Zhengxin Fang, Yi Yuan, Jingyu Zhang, Yue Liu, Yuechen Mu, Qinghua Lu, Xiwei Xu, Jeff Wang, Chen Wang, Shuai Zhang, and Shiping Chen. 2023. MLOps Spanning Whole Machine Learning Life Cycle: A Survey. *CoRR* abs/2304.07296 (2023). https://arxiv.org/abs/2304.07296

Stephen Fleming. 2020. *Accelerated DevOps with AI, ML & RPA – Non-Programmer’s Guide to AIOPS & MLOPS*. Independently published.

Sindhu Ghanta, Sriram Subramanian, Swaminathan Sundararaman, Lior Khermosh, Vinay Sridhar, Dulcardo Arteaga, Qianmei Luo, Dhananjoy Das, and Nisha Talagala. 2018. Interpretability and Reproducability in Production Machine Learning Applications. In *17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018, Orlando, FL, USA, December 17-20, 2018*, M. Arif Wani, Mehmed M. Kantardzic, Moamar Sayed Mouchaweh, João Gama, and Edwin Lughofer (Eds.). IEEE, 658–664.

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

Joanna Juziuk, Danny Weyns, and Tom Holvoet. 2014. Design Patterns for Multi-agent Systems: A Systematic Literature Review. In *Agent-Oriented Software Engineering - Reflections on Architectures, Methodologies, Languages, and Frameworks*. 79–99. DOI:http://dx.doi.org/10.1007/978-3-642-54432-3_5

Gabriel Laberge, Ulrich Aïvodji, Satoshi Hara, Mario Marchand, and Foutse Khomh. 2023. In *11th International Conference on Learning Representations, ICLR 2023, May, Kigali Rwanda*. OpenReview.net, 1–30.

Valliappa Lakshmanan, Sara Robinson, and Michael Munn. 2020. *Machine Learning Design Patterns*. O’Reilly Media, Inc., New York, NY, USA. https://learning.oreilly.com/library/view/machine-learning-design/9781098115777/

Table I. Identified software-engineering design patterns for machine-learning applications (SEP4MLA)

Category	ID	Pattern Name	Summary
Topology Patterns	P_1	Different Workloads in Different Computing Environments	Physically isolate different workloads to separate machines. Then optimize the machine configurations and the network usage.
	P_2	Distinguish Business Logic from ML Models	Separate the business logic and the inference engine, loosely coupling the business logic and ML-specific dataflows.
	P_3	ML Gateway Routing Architecture	Install a gateway before a set of applications, services, or deployments, some of which are implemented by ML with an inference engine. Use application layer routing requests to the appropriate instance.
	P_4	Microservice Architecture for ML	Define consistent input and output data. Provide well-defined services to use for ML frameworks.
	P_5	Lambda Architecture for ML	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 layers and then merges them.
	P_6	Kappa Architecture for ML	Support both real-time data processing and continuous reprocessing with a single stream processing engine.
Programming Patterns	P_7	Data Lake for ML	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	Decouple at different levels of complexity from the simplest to the most complex.
	P_9	Encapsulate ML Models Within Rule-base Safeguards	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	Discard the code created for the PoC and rebuild maintainable code based on the findings from the PoC.
Model Operation (and Security) Patterns	P_{11}	Parameter–Server Abstraction	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 (Federated Learning)	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	Encrypt data from each mobile device in collaborative learning and calculate totals and averages without individual examination.
	P_{14}	**Explainable Proxy Model**	**Run the explainable inference subsystem in parallel with the primary inference subsystem to monitor prediction differences.**
	P_{15}	ML Versioning	Record the ML model structure, training dataset, training system, and analytical code to ensure a reproducible training process and an inference process.

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

Jomphon Runpakprakun, Sien Reeve Ordonez Peralta, Hironori Washizaki, Foutse Khomh, Yann-Gael Gueheneuc, Nobukazu Yoshioka, and Yoshiaki Fukazawa. 2021. Software Engineering Patterns for Machine Learning Applications (SEP4MLA) – Part 3 – Data Processing Architectures. In *28th Conference on Pattern Languages of Programs in 2021 (PLoP'21)*. Hillside, Inc., 1–11.

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).

Hironori Takeuchi, Hironori Washizaki, Naoshi Uchihira, Satoshi Okuda, Naohisa Shioura, Takuo Doi, Naotake Natori, and Kiyoshi Honda. 2020. Machine Learning Application System Patterns. `https://www.slideshare.net/HironoriTAKEUCHI1/ss-236852460` (in Japanese). (July 2020).

Hironori Washizaki, Foutse Khomh, Yann-Gaël Guéhéneuc, Hironori Takeuchi, Naotake Natori, Takuo Doi, and Satoshi Okuda. 2022. Software-Engineering Design Patterns for Machine Learning Applications. *Computer* 55, 3 (2022), 30–39.

Hironori Washizaki, Foutse Khomh, and Yann-Gael Gueheneuc. 2022. Software Engineering Patterns for Machine Learning Applications (SEP4MLA) – Part 4 – ML Gateway Routing Architecture. In *29th Conference on Pattern Languages of Programs in 2022 (PLoP'22)*. Hillside, Inc., 1–8.

Hironori Washizaki, Foutse Khomh, Yann-Gael Gueheneuc, Hironori Takeuchi, Satoshi Okuda, Naotake Natori, and Naohisa Shioura. 2021. Software Engineering Patterns for Machine Learning Applications (SEP4MLA) – Part 2. In *27th Conference on Pattern Languages of Programs in 2020 (PLoP'20)*. Hillside, Inc., 1–10.

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.

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

Guang Yang, Qinghao Ye, and Jun Xia. 2022. Unbox the black-box for the medical explainable AI via multi-modal and multi-centre data fusion: A mini-review, two showcases and beyond. *Inf. Fusion* 77 (2022), 29–52. DOI:http://dx.doi.org/10.1016/j.inffus.2021.07.016