Software Engineering Patterns for Machine Learning Applications (SEP4MLA) - Part 4 - ML Gateway Routing Architecture

HIRONORI WASHIZAKI, Waseda University / National Institute of Informatics / System Information / eXmotion
FOUTSE KHOMH, Polytechnique Montréal
YANN-GAËL GUÉHÉNEUC, Concordia University

Machine learning (ML) researchers study the best practices to develop and support ML-based applications to ensure quality and determine the constraints applied to their application pipelines. Such practices are often formalized as software 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, "ML Gateway Routing Architecture", in the standard pattern format so that practitioners can (re)use it in their contexts and benefits. The pattern addresses the problem of tight coupling among ML-implemented and non-ML business logic as well as the front-end client by installing a gateway that routes requests.

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. and Guéhéneuc, Y.-G. 2022. Software Engineering Patterns for Machine Learning Applications (SEP4MLA) - Part 4. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 0 (October 2022), 8 pages.

1. INTRODUCTION

Machine learning (ML) researchers study best practices to develop and maintain ML-based applications to ensure quality and determine the constraints on their application pipelines. 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]. 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 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. Based on the overview, we grouped these SEP4MLA into three categories, shown in Table 1: ML applications

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;

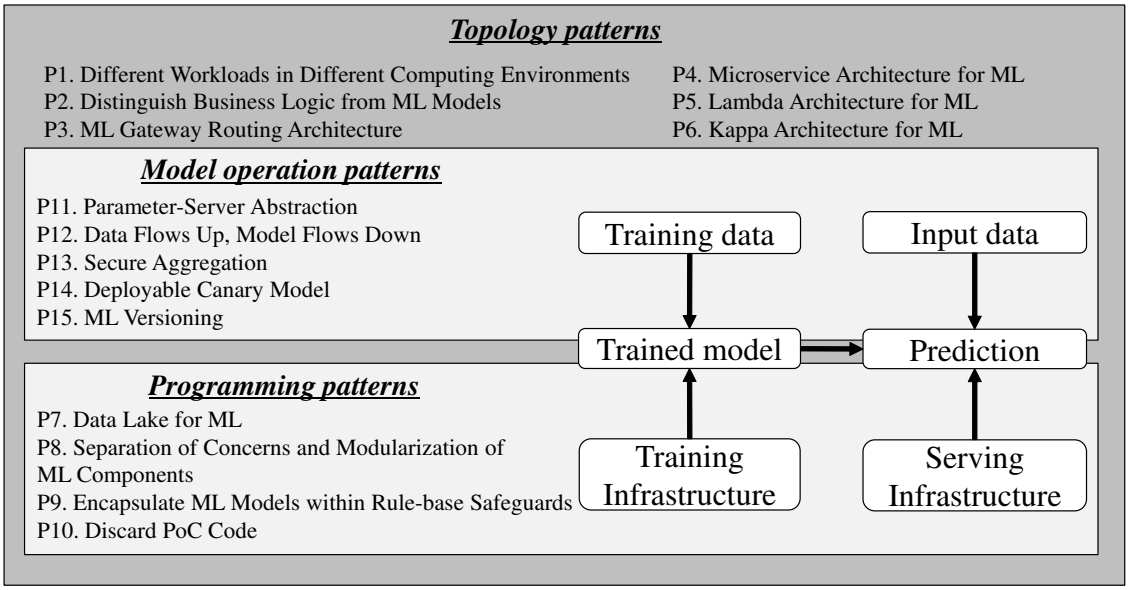

Fig. 1. Machine learning system overview and categories of software engineering design patterns for ML

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.

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 already described most of the patterns in Table 1: "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]; and, "Lambda Architecture for ML" (P_5) and "Kappa Architecture for ML" (P_6) in Part 3 [Runpakprakun et al. 2021].

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.

In the following, we describe the ML pattern "ML Gateway Routing Architecture" (P_3).

2. ML GATEWAY ROUTING ARCHITECTURE (P_3)

2.1 Source

[Yokoyama 2019]

2.2 Intent

Avoid tight coupling among ML-implemented and non-ML business logic as well as the front-end client by installing a gateway that routes requests to the appropriate services.

Table I. Identified ML Patterns

Category	ID	Pattern Name	Summary
Topology	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 layer, 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	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	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}	Deployable Canary Model	Run the explainable inference pipeline in parallel with the primary inference pipeline 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.

2.3 Example

Figure 2 presents an example of the architecture of a Slack-based Chatbot system with a calendar service (as a non-ML business logic service) and a Chatbot service (as an ML-implemented service supported by an inference engine). Since the Chatbot UI as a client has to know details of these ML/non-ML services to utilize them, these components are tightly coupled, resulting in low flexibility, scalability, and manageability.

2.4 Context

The target ML system contains various business applications, services, and deployments. Some of which are implemented by ML with an inference engine and data processing. These multiple applications and services can be deployed on different internal servers in the same internal network or on different virtual endpoints of the same server.

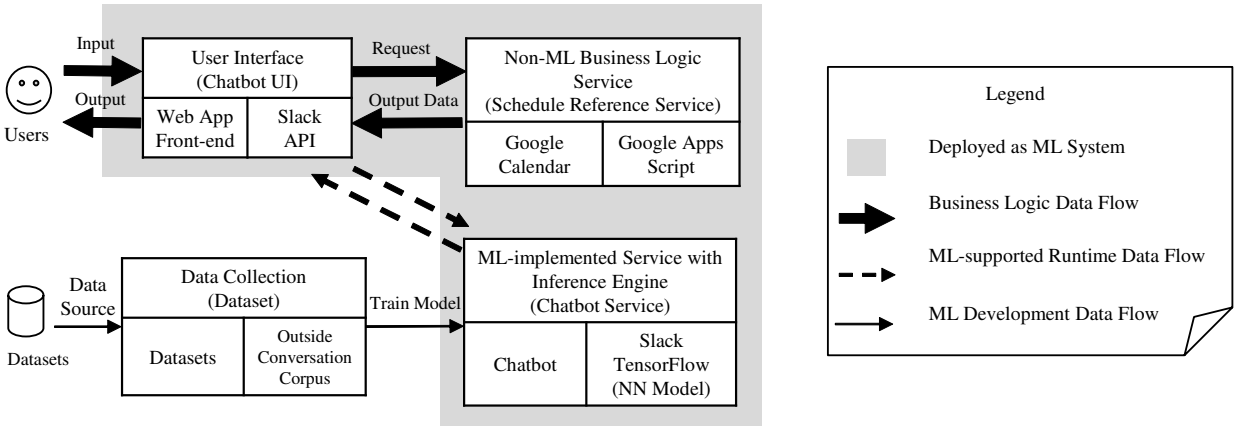

Fig. 2. Example of Chatbot system architecture without pattern application

2.5 Problem

Tight coupling among the front-end client (i.e., the UI), non-ML business logic, business logic supported by ML, and the inference engine leads to low flexibility and scalability. Moreover, when a client directly uses these multiple applications and services independently, it can be difficult to set up and manage individual endpoints for each service.

The following forces are associated with this problem:

—Flexibility and scalability: The components should be loosely coupling to ensure high flexibility and scalability.

—Manageability: Complex setting up and managing individual endpoints would lead to low manageability and extensibility of services, the client, and the entire system.

—Performance and reliability: Little performance and reliability degradation by adopting a single proxy would be acceptable while considering the above quality attributes.

2.6 Solution

To wrap each business logic with/without the support of the ML inference engine into a different service presenting a unique business API, install a gateway before a set of services, and use application layer routing requests to the appropriate instance. It avoids tight coupling among ML-implemented and non-ML services as well as the front-end client. The client can use multiple non-ML and ML-implemented business services without difficult setup and management of individual endpoints. Figure 3 shows the structure of the solution architecture.

A general inference engine might be provided as an independent service via the gateway; however, it can easily expose the inference engine details leads to the client, resulting in unnecessary tight coupling between the client and ML models.

This architecture can be easily extended to handle a variety of data ranging from structured to unstructured by additionally adopting “Data Lake for ML” as shown in Fig. 4.

2.7 Example resolved

Figure 5 presents a possible implementation of the pattern and the Data Lake for ML in the same motivating example (i.e., the Slack-based Chatbot system). Using the pattern, the necessary elements as well as their relationships are easily specified while having clear separation between the calendar service (as the related non-ML business logic service), the Chatbot service (as the ML-implemented service supported by the inference engine), and the underlying ML components.

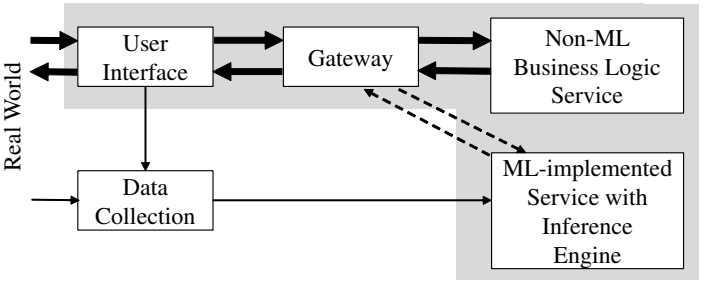

Fig. 3. Structure of the ML Gateway Routing Architecture

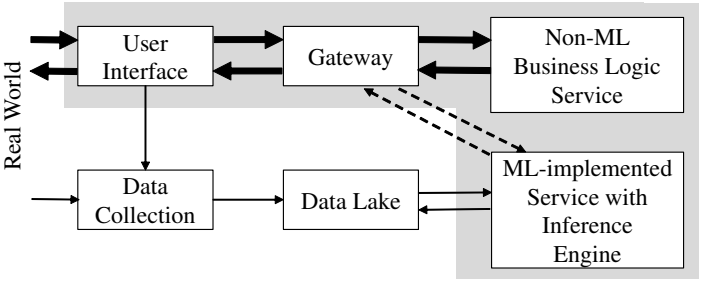

Fig. 4. Structure of the ML Gateway Routing Architecture with the Data Lake for ML

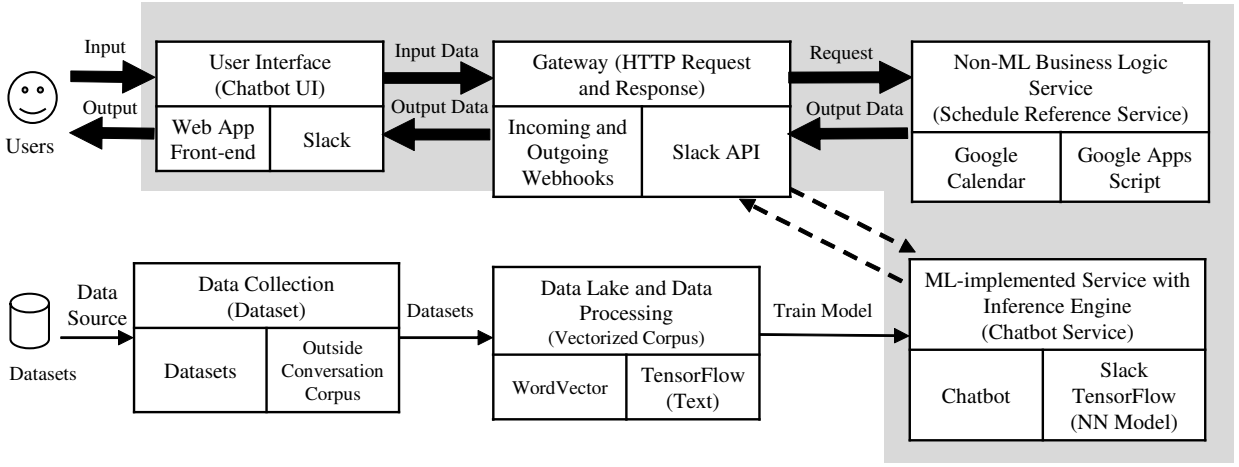

Fig. 5. Example of Chatbot system architecture by applying “ML Gateway Routing Architecture”

2.8 Known Uses

Cloud platforms provide gateway services, including Amazon API Gateway[1] and Azure Application Gateway[2] that support ML models and ML-implemented services.

[1] https://aws.amazon.com/api-gateway/
[2] https://docs.microsoft.com/azure/application-gateway/

Fig. 6. Overview of the Amazon API Gateway (adopted from `https://aws.amazon.com/api-gateway/`)

2.9 Consequences

The pattern presents the following advantages:

—Loose coupling and high flexibility and scalability: Clients can utilize multiple (ML-supported) applications and services without knowing their details, resulting in loose coupling among ML-implemented and non-ML services and the front-end clients.

—Ease of management: The client can use multiple non-ML and ML-implemented services without complex setup and management of individual endpoints, resulting in improved manageability of services, the client, and the entire system.

Possible liabilities include:

—Possibility of low performance: This architecture creates an extra overhead between the user interface and the business logic, and the latency performance of the system might decrease.

—Possibility of low reliability: The gateway can become a single point of failure (SPOF), compromising the system's availability.

2.10 See also

—Gateway Routing Architecture [Narumoto 2017]: "ML Gateway Routing Architecture" is an extended case of the "Gateway Routing Architecture" in the ML application domain.

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

—Encapsulate ML models within rule-base safeguards (P_9) [Kläs and Vollmer 2018; Washizaki et al. 2021]: 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 setup and management of individual endpoints.

—Distinguish Business Logic from ML Models (P_2) [Yokoyama 2019; Washizaki et al. 2020]: "Distinguish Business Logic from ML Models" can help achieving the objective of decoupling business logic and ML models with less flexibility in terms of logic services.

3. CONCLUSION

In this paper, we described the ML pattern "ML Gateway Routing Architecture", which we choose 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.

Acknowledgement

We are grateful to our shepherd, Eduardo B. Fernandez, for his careful and valuable reviews that significantly improved this paper.Also, We would like to thank all participants of the writers' workshop at PLoP 2022. This work was supported by JSPS Bilateral Program JPJSBP120209936, JSPS KAKENHI 21KK0179, and JST-Mirai Program Grant Number JPMJMI20B8.

Received July 2022; revised August 2022; accepted

REFERENCES

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

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

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

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/

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

M Narumoto. 2017. Gateway Routing pattern. https://docs.microsoft.com/en-us/azure/architecture/patterns/gateway-routing. (June 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.

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

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, 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.

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