

Software Engineering Patterns for Machine Learning Applications (SEP4MLA) - Part 3 - Data Processing Architectures

JOMPHON RUNPAKPRAKUN, Waseda University

SIEN REEVE ORDONEZ PERALTA, Waseda University

HIRONORI WASHIZAKI, Waseda University / National Institute of Informatics / System Information / eXmotion

FOUTSE KHOMH, Polytechnique Montréal

YANN-GAËL GUÉHÉNEUC, Concordia University

NOBUKAZU YOSHIOKA, Waseda University

YOSHIAKI FUKAZAWA, Waseda University

Machine learning researchers regularly try to study the best practice to develop and maintain the ML-based application to maintain the quality level and determine their application pipeline's constrained. 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 two ML patterns in the standard pattern format so that practitioners can (re)use them in their contexts: "Lambda Architecture for ML" and "Kappa Architecture for ML."

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:

Runpakprakun, J. and Washizaki, H. 2021. Software Engineering Patterns for Machine Learning Applications (SEP4MLA) - Part 3. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 0 (March 2020), 11 pages.

1. INTRODUCTION

Machine learning researchers regularly try to study the best practice to develop and maintain the ML-based application to maintain the quality level and determine their application pipeline's constrained. Such practices are often formalized as software patterns. We call these software-engineering 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

Author's address: J. Runpakprakun, H. Washizaki, S. Peralta and N. Yoshioka, 3-4-1 Okubo, Shinjuku-ku, Tokyo, Japan; email: k-jomphon@moegi.waseda.jp and 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;

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission. A preliminary version of this paper was presented in a writers' workshop at the 28th Conference on Pattern Languages of Programs (PLoP). PLoP'21, OCTOBER, Virtual. Copyright 2021 is held by the author(s). HILLSIDE 978-1-941652-03-9

Table I. Identified ML Patterns

Category	ID	Pattern Name	Summary
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 for ML	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 for ML	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 for ML	Support both real-time data processing and continuous reprocessing with a single stream processing engine [Tyagi 2017].
Programming	P_7	Data Lake for ML	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 [Klås 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 (Federated Learning)	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].

discovered 15 software-engineering design patterns for machine-learning applications (hereafter, ML patterns) by doing a thorough search of available literature on the subject. Details of our methodology are available in [Washizaki et al. 2020].

We grouped these ML patterns into three categories, shown in Table 1: ML applications topology patterns that define an entire application architecture, 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 ML patterns are well-documented in standard pattern format, which includes clear problem statement and corresponding solution description. Thus, we describe these ML patterns in a standard pattern format so that practitioners can (re)use them in their contexts¹.

To describe each ML pattern uniformly, we adopted the well-known Pattern-Oriented Software Architecture format (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 its content.

In the following, we describe the two interrelated ML patterns “Lambda Architecture for ML” (P_5) and “Kappa Architecture for ML” (P_6).

2. LAMBDA ARCHITECTURE FOR ML (P_5)

2.1 Source

[López-Martínez et al. 2020]

2.2 Intent

To enable the deployment of ML-based applications in real-time while maintaining the prediction accuracy.

2.3 Context

In the present day, the demand for real-time serving ML-based applications is increasingly high, since these kinds of applications can be utilizing to support the decision making process to the human, whether it is a product recommendation system in online shopping site for the everyday user or the professional application like the medical support system in a hospital. These kinds of applications are usually required to serve a real-time request, which differs from the traditional way we usually design the ML-application pipeline, which usually deals with the batch process and needs a certain period of time to process.

2.4 Problem

Traditionally in a ML-based system, the ETL Architecture (which ETL stand for Extract, Transform, Load) [Zweben 2016], is used in the data processing pipeline. When we try to serve the real-time application, we could use “Online Transaction Processing” (OLTP) to aggregate and process data in real-time, then “Online Analytical Processing” (OLAP), to further gain insight from the data obtained from the OLTP. Even though in terms of accuracy, the ETL Architecture would produce an accurate ML model because it will be utilizing the Batch process, which is one of the most reliable ways to training the model. However, to stream and utilize the model in a real-time application, this architecture also yields delays in the process (of OLTP) and can take some time to complete the OLAP process.

2.5 Solution

Therefore, to overcome the problem of the real-time utilization of our ML-based application, we could come with a new pipeline design for the ML application by adopting the Lambda Architecture paradigm for processing the data. Since the regular Lambda data processing pipeline maintains both Speed-Layer and Batch Layer (Refer to Fig.1), we could deploy two separate ML-model for analyzing data in each layer. The ML-model in the real-time layer focuses on producing a certain reliable result in real-time and costuming a few pieces of input data, while another ML-model in the batch layer could focus on processing a larger throughput of data to extract more insight from it, consequently yield a more accurate inference result. With the combination of Lambda architecture and the ML-process, we call this solution the Lambda architecture for ML.

¹We previously described seven other ML patterns: “Data Lake for ML” (P_7), “Distinguish Business Logic from ML Models” (P_2), “Microservice Architecture for ML” (P_4), “ML Versioning” (P_5) in [Washizaki et al. 2020], and “Different Workloads in Different Computing Environments” (P_1), “Encapsulate ML Models Within Rule-base Safeguards” (P_9), and “Data Flows Up, Model Flows Down” ($P_{1,2}$) in [Washizaki et al. 2021].

Looking into the detail of each layer, in the speed layer, after the input data had been injected into the layer, the real-time engine will perform the ETL process to the stream of data (in this case, all data are stream data. Hence differ from the tradition ETL which data has to be pulled from the database). After finishing ETL, the ML stream analysis has been applied to perform a training and/or inferencing process, then return the result to the real-time serving layer.

On the other side, when data is injected into the Batch layer, it will be processed and be categorized to the data warehouse, or data lake [Washizaki et al. 2020] (P_7). When the threshold is reached (whether a certain time has passed or a certain amount of input data is reached), the batch process starts the ML analysis on that batch of data. When the analysis is finished, the result is pushed to the dashboard serving, then the cycle restart again.

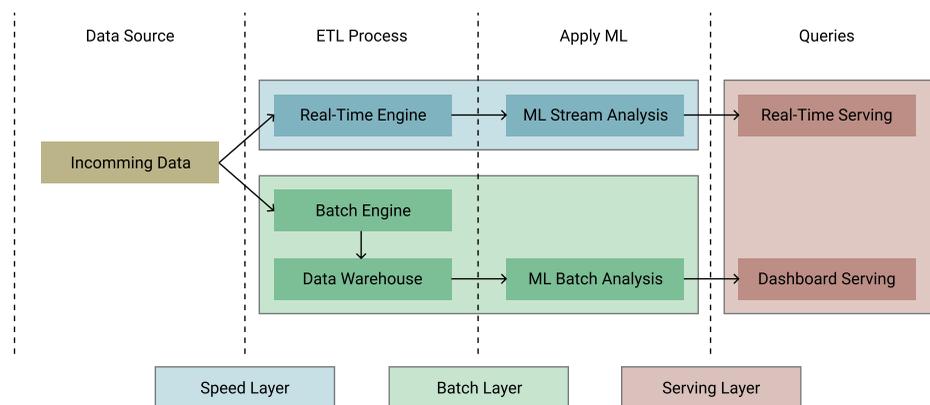


Fig. 1. Structure of the “Lambda Architecture for ML” pattern

Lastly, here are some of the possible specification that make up each layer, as shown below:

—For Real-Time/Speed Layer [Lakshmanan et al. 2020]:

- The Speed Layer will likely be implemented as Microservices, so it is simple to implement auto-scaling.
- The inference process can be used on the client-side and deployed on different environments. there is an increasing trend of deploying multiple competing models in production, and using feedback from customers and reinforcement learning (multi-armed bandits) to make optimal decisions about which model to select for a given input, such deployment strategy may require a different architecture.
- Related Service: Apache Storm, Apache Samza, Apache Spark, SQLstream, and Azure Stream Analytics [Karbhari 2020].

—For Batch Layer [Lakshmanan et al. 2020]:

- The Batch Layer could interact with a Data Lake on the server thus, it could process large amount of data.
- The Batch Layer can be deployed across several distributed data processing infrastructures, therefore having more resources to validate the data, enhancing prediction accuracy.
- Related Service: MapReduce, Apache Spark, Apache Beam, and BigQuery [Karbhari 2020].

2.6 Known Usage

The Kerry organization designs and propose the “Digital Health Platforms” [López-Martínez et al. 2020] with the aim to improve the integration of the integrated health model for the care center and hospital, which have to deal with various kinds of information, for example, the Health Information System (HIS), the Lab Information

System (LIS), some Enterprise Resource Planning (ERP), and several other types, which traditionally maintain separately, resulting in an inefficient health management process to support the physicians when they make a clinical decision. Consequently, the Kerry organization proposes bringing together various types of data, from clinical data to business data. Then applying machine learning to gain more helpful insight for clinical decisions, which potentially help improve the decision-making process for the medical operator. The overview result of this architecture design has shown in Figure number 2.

By adopting the design of Lambda Architecture and deploying on hybrid cloud infrastructure, the system can deal with both structure and unstructured, which in this architecture design, the structure data such as diagnostic reports, will be processed by ML in batch layer. The unstructured, such as real-time patient status, and early warning system, will be processed in real-time in the speed layer. Since it is on a hybrid cloud, the batch process can also be scaled horizontally, and the streaming process can also be dynamically computed, utilizing Stateless Serving Function.

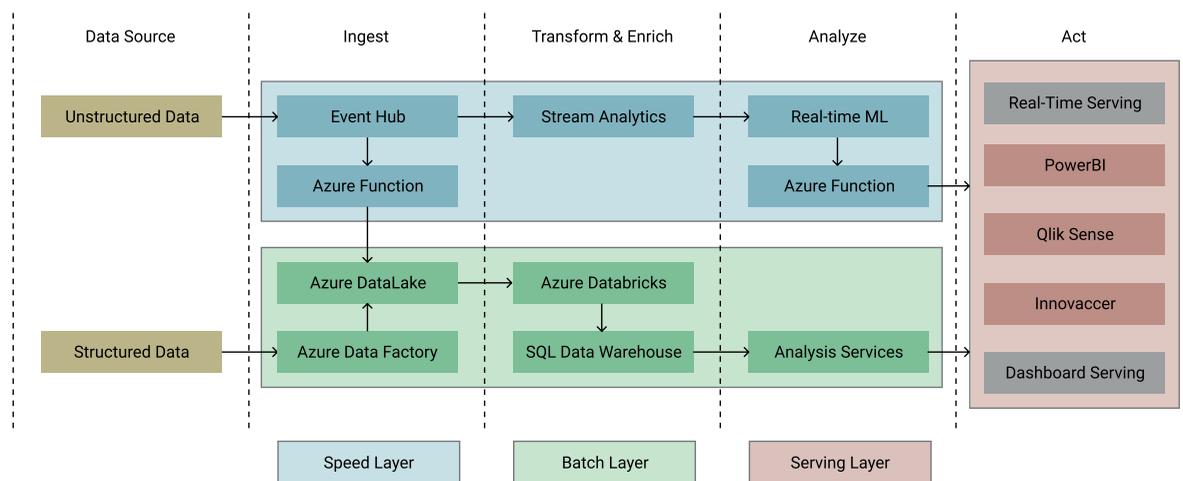


Fig. 2. Azure Big Data and Machine Learning Lambda Architecture.

2.7 Discussion

One of the key benefits of using the Lambda Architecture in this design is data validation [nullQueries 2021] because this design maintains a Batch Layer. Nevertheless, the real-time prediction is handled by the speed layer; when a cycle of the batch process has complete (in the batch process apply ML analysis to the dataset collecting from a period of time), the result from the batch layer will be replacing the one produced by the speed layer, since the result from batch trend to be more accurate. Therefore, the program can perform real-time analysis and give some intermediate results. Nevertheless, the more accurate result will also be processed and collect for later use by the batch process.

As can be seen in this architecture, these two layers work hand in hand to complement the drawback of each technique. The speed of the real-time Layer help to serving a live request, while the Batch layer help to improving the accuracy of the predictions result for later use. However, without a good design of the Lambda architecture, both Batch Layer and Speed Layer could potentially cause various issues, from compatibility issues, which cause from the fundamental perspective of Lambda Architecture itself, that is, it requires to maintain two separate

technology stack for each layer. Hence the architectural designer and programmer have to know various sets of tools that will be put together in each layer, and there is no guarantee that each component, which might perform very well on its own, will be able to appropriately interact with other parts in the pipeline without significant issue.

Secondly, as mentioned earlier about the data validation process, when the batch process finishing the analysis cycle, the result obtains earlier from the speed layer will be replaced by the one obtained from the batch layer. Therefore, in the type of application that needs a very accurate prediction result, such as scientific analysis, but also requires to serve the live request, then the program provides the real-time inferencing result based on the result from the speed layer. Eventually, the more accurate analysis results were obtained from the batch layer, which might differ from the previous result produced by the speed layer. Therefore, the prediction result obtained earlier could be considered inaccurate, which might have a negative consequence follow up.

2.8 Related Patterns

- The (normal) Lambda Architecture: is one of a famous a data processing technique that design to handling with a large quantity of data, with it unique design which is the maintaining of both stream process (speed layer) and the batch process, each layer can compensate the drawback of one another, for instance, the batch process, which is famous for its throughput, but it also came with latency, so it causes a latency when serving a live request, on the other hand, for the streaming process, as it names suggest, its focus on utilizing the parallel process to scaling and serving many requests with minimum delay, however, each client can only be served with a limited amount of processing throughput. With all of these reasons, to combine these two data processing paradigms, the Lambda Architecture is becoming a more versatile and flexible design to adopt.
- Batch Serving Function [Lakshmanan et al. 2020]: performing asynchronous forecasts with a heavy load is challenging. The Batch Serving process allows the serving infrastructure to asynchronously handle occasional or periodic requests for a large amount of the input data in a distributed environment.
- The Stateless Serving Function [Lakshmanan et al. 2020]: with a Stream Processing (Speed Layer), the environment used for hosting the finished model may differ from the one used during the training process. The Stateless Serving Function allows the serving infrastructure to scale and handle prediction requests dynamically by exporting the model core and deploying it as a stateless REST API.
- The Microservice Architecture for ML [Washizaki et al. 2020] (P_4): in implementing the real-time layer in the Lambda Architecture for ML, to simplify the process scaling and the orchestration, implementing the pipeline in the Microservices style help simplify pipeline implementation and make it easier to operate.
- The Kappa Architecture for ML (P_6 , see Section 3): for some workloads, the data can be fully streamed and does not require reinterpretation when the application obtains more data. In such a case, the Kappa Architecture for ML simplifies the Lambda Architecture for ML by removing the Batch Layer. Therefore, we can also consider the Kappa Architecture for ML as a special form of the Lambda Architecture for ML.
- The Data Lake for ML [Washizaki et al. 2020] (P_7): when implementing the Batch Layer, the data storage plays a significant role in processing, storing, and feeding data to the ML training process. Consequently, a Data Lake, which is designed to deal with both structure and unstructured data, could be more versatile to use with the ML model than the traditional data warehouse paradigm.

3. KAPPA ARCHITECTURE FOR ML (P_6)

3.1 Source

[Tyagi 2017; Hayashida and Sasaki 2018]

3.2 Intent

To enable the deployment of ML-based applications in real-time while maintaining the prediction accuracy, similar to Lambda Architecture, but further reduce the redundancy and complexity of the pipeline.

3.3 Context

Similar to Lambda Architecture, to satisfy an increasing demand in the real-time ML-based application, several tools in the pipeline have been developed, and as we tried to put more of these components into the Lambda pipeline, the harder it can be achieved. As we already know, the Lambda Architecture maintains two separate layers to take advantage of each layer (real-time layer for speed and the batch layer for accuracy); however, maintaining two sets of technologies could bring several compatibility and consistency issues during design maintenance and operation.

3.4 Problem

In terms of consistency, for instance, after the ML model in batch finishes its analyzing cycle and merges the prediction result to the database, some of the inferencing results will be different from the one obtained from the ML-model deployed in a real-time layer. Hence the change of the prediction result has occurred, causing the inconsistency result providing in the serving layer.

On the other hand, in the compatibility perspective, to design and maintain the Lambda Architecture pipeline, we might have to use two separate sets of tools for implementing each layer. It requires the developer to find or developing the bridge to connect one component to another, which will cause more complexity in the final design. Consequently, when trying to adding more features or fixing some failures without affecting the integrity of the whole application pipeline has become relatively more difficult to achieve.

3.5 Solution

From these reasons, considering using the Kappa Architecture pipeline, which reduces complexity by maintaining only streaming process (speed layer). With the removal of the batch layer, the system has fewer parts to maintain. But since the real-time layer is entirely streamable process, by its nature, it cannot process a big batch of data and perform batch analysis, consequently, when we need to re-process some historical data to gain more insight from data which generally performed in a batch process, we could also design the Kappa architecture to collect the raw data and the initial prediction result to some kind of database, whether the data warehouse or data lake, in its arriving order (to manipulate the way batch process preparing the data before feeding to the batch analysis, in this case to the ML model).

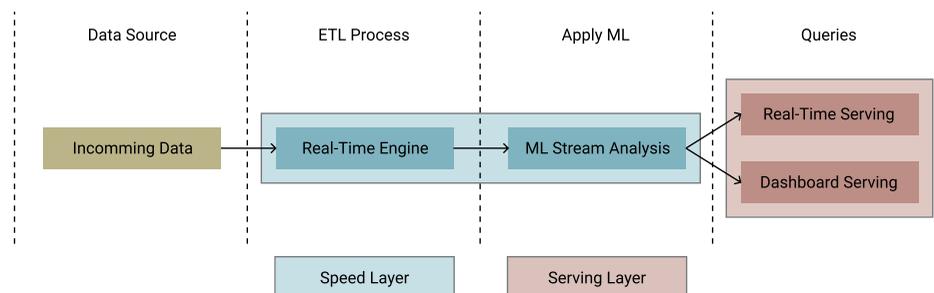


Fig. 3. Structure of the "Kappa Architecture for ML" pattern

Then, we can stream these collected data back to the streaming layer to perform re-interpretation using ML to these data and obtain more insight from it. With this paradigm we could produce the system that improving overtime, for instance, the recommendation system which will change the recommending result after obtaining more input data from client side. Therefore, we call this solution the Kappa Architecture for ML.

3.6 Known Usage

A company name Recruit Lifestyle has run many kinds of services [Hayashida & Sasaki, 2018] (website and application) in various sectors, from gourmet, HR, housing, travel, which require a lot of data analysis. From their experience, they found that several problems occur when they try to design and work with data analysis to serving several of their services, such as, too complex data pipeline from overusing ETL batch process, and the problem about unclear schema, availability and accessibility of the data for their data science team. Therefore, their goals are 1) To reduce the pipeline complexity and 2) To have an explicit, unified schema and context information.

For these reasons, the team came up with the approach called “Datahub Architecture”, which utilizing only a stream process to perform data processing - hence reducing the complexity of their pipeline design and since it is a stream process, it can be flexible scale to match the need - before feeding to the Google service like “Google Dataflow” then “Google Pub/Sub” [Google 2021] to perform further data analysis with ML, with the unified salvation format across their several products. Therefore, the team of data science can now working with the data more efficiently. The overview result of this architecture design has shown in Figure number 4.

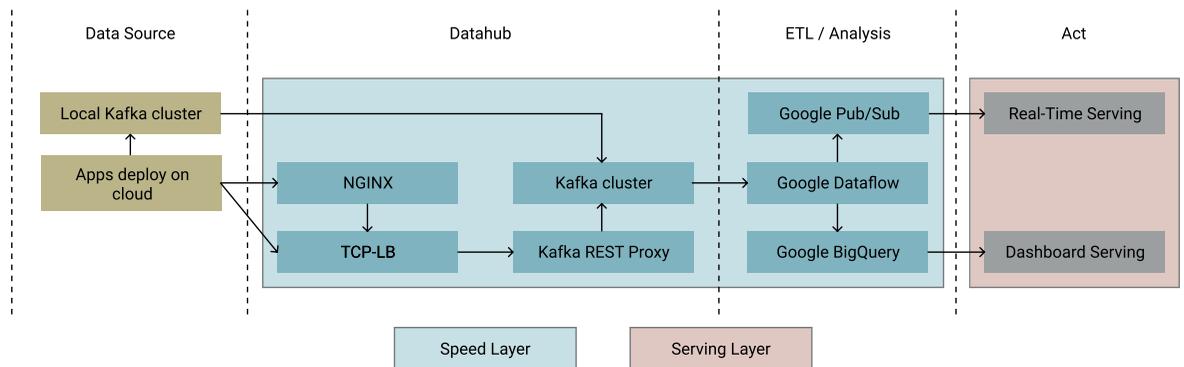


Fig. 4. Design of Datahub Platform implemented on GCP (Kappa Architecture).

3.7 Discussion

There are several benefits from using the Kappa Architecture [Seyvet and Viela 2016] in this ML pipeline. Since this pipeline could both serve in real-time, maintain the low latency requirement, and perform a similar process generally performed in the batch process by adding some workaround steps. Also, it does not require us to work on so many tools like Lambda Architecture, which may not be able to be implemented by a single technology stack; therefore, the more straightforward implementation and maintenance it becomes. Also, since the whole pipeline can be stream, it is very convenient for this architecture to be horizontally scaled because it can be implemented in the Microservice style.

Nevertheless, the Kappa Architecture also came with its own drawbacks [nullQueries 2021] because it requires the whole process to be streamable; hence it might not be suitable for some workloads in which the data might not be easily put on to the stream.

Furthermore, as an inherent problem of stream process, when the system record the data into the database according to the arriving order (for an ML re-processing purpose), the “dirty data” or duplicate data might still mingle with the good data, which usually in the normal batch process, we can utilize batch engine to filtering out

these type of data, to preserving only valuable data. So it is hard to achieve in Kappa Architecture which only relies on stream process.

3.8 Related Patterns

- The (normal) Kappa Architecture: is one of a famous data processing technique designed to handle data with a latency-sensitive application, differs from the Lambda Architecture, this architecture discard the batch process entirely, and remain only the stream process. With this fact, this architecture still working fine in real-time based data processing; however, to compensate for the miss of batch layer, while performing real-time data processing, the Kappa architecture also collecting ingested data in the chronological to the database. Consequently, when needing to reprocess the data by using a historical dataset similar to what the normal batch process does, this architecture will stream the recorded data back to the speed layer again to perform a re-interpretation without the need to maintaining batch process like in Lambda Architecture. On the other hand, since this architecture maintains only one layer, there are fewer components to deal with when designing and maintaining.
- The Stateless Serving Function [Lakshmanan et al. 2020]: similar to Lambda Architecture, Kappa Architecture also maintain a Stream Processing (Speed Layer) in the pipeline, which we might deploy the model in other environments. The Stateless Serving Function will play a role in serving, scaling, and handling prediction requests dynamically.
- Microservice Architecture [Washizaki et al. 2020] (P_4): Since Kappa architecture is fully implemented on the speed layer, all the process can be efficiently pipelined through the cloud service. To further optimize computing resource use, implementing the Kappa-based system in the microservice style would be more straightforward to implement while maintaining its efficiency.
- The Lambda Architecture for ML (P_5): for some ML workloads, which need to operate with some legacy ML batch system or require the reinterpretation of the prediction results after obtaining more data. The Lambda Architecture for ML is a more appropriate choice because it offers both a Speed and a Batch Layer to improve prediction latency and accuracy.

4. CONCLUSION

In this paper, we described two patterns: “Lambda Architecture for ML” and “Kappa Architecture for ML”, which 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 application with the required quality.

In the future, we plan to write all ML patterns in a 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 patterns.

Acknowledgement

We are grateful to our shepherd Eduardo Guerra for their careful and valuable reviews that significantly improved this paper. We would like to thank also the participants of the writers’ workshop at PLoP 2021. 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 2021; revised August 2021; 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).
- 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).
- Google. 2021. What is Pub/Sub? <https://cloud.google.com/pubsub/docs/overview>. (2021).
- Kenji Hayashida and Toru Sasaki. 2018. Best practices for developing an enterprise data hub to collect and analyze 1 TB of data a day from a multiple services with Apache Kafka and Google Cloud Platform [Big data and data science in the cloud, Data engineering and architecture] Strata Data Conference, New York, NY, USA. <https://medium.com/acing-ai/machine-learning-system-design-real-time-processing-6a952793925> (September 13, 2018).
- Vimarsh Karbhari. 2020. Machine Learning System Design: Real-time processing <https://medium.com/acing-ai/machine-learning-system-design-real-time-processing-6a952793925>. *Medium* (14 May 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
- Valliappa Lakshmanan, Sara Robinson, and Michael Munn. 2020. *Machine Learning Design Patterns*. O'Reilly Media, Inc. <https://learning.oreilly.com/library/view/machine-learning-design/9781098115777/>
- Fernando López-Martínez, Edward Rolando Núñez-Valdez, Vicente García-Díaz, and Zoran Bursac. 2020. A Case Study for a Big Data and Machine Learning Platform to Improve Medical Decision Support in Population Health Management. *Algorithms* 13, 4 (2020), 4–6. DOI:<http://dx.doi.org/10.3390/a13040102>
- Pradeep Menon. 2017. Demystifying Data Lake Architecture. <https://www.datasciencecentral.com/profiles/blogs/demystifying-data-lake-architecture>. (August 2017).
- nullQueries. 2021. How to Design Realtime Data Consumption using Lambda and Kappa Architectures [Video]. (2021). <https://www.youtube.com/watch?v=wEue0J40S1g>
- 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>
- Nicolas Seyvet and Ignacio Mulas Viela. 2016. Applying the Kappa architecture in the telco industry <https://www.oreilly.com/content/applying-the-kappa-architecture-in-the-telco-industry/>. *O'Reilly Media, Inc.* (19 May 2016).
- 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, Yann-Gaël Guéhéneuc, 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 Software Engineering Patterns for Machine Learning Applications (SEP4MLA) - Part 3 - Data Processing Architectures — Page 10

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>

Monte Zweben. 2016. 5 Reasons Machine Learning Applications Need a Better Lambda Architecture <https://www.kdnuggets.com/2016/05/5-reasons-machine-learning-applications-lambda-architecture.html>. *KDnuggets* (May 2016).