Software Engineering Patterns for Machine Learning Application Systems

HIND MILHEM, The Hashemite University
NEIL B. HARRISON, Utah Valley University

Software Engineering Patterns provide reusable solutions to common design problems, offering a structured framework for developing robust, scalable, and maintainable software systems. When applied to the realm of machine learning, these patterns offer invaluable guidance in navigating the complexities inherent in building ML-powered applications. Understanding the writing patterns of machine learning application systems is essential for engineers, data scientists, and developers alike. It encompasses the strategies, methodologies, and best practices employed in articulating the architecture, design, and functionality of ML-driven solutions. In this paper, we document two patterns within machine learning application systems, examining their role in fostering collaboration, enhancing readability, and facilitating the seamless integration of complex components. One pattern is design pattern "Representative Validation Data" pattern and the other one is architectural pattern "Gateway Routing" pattern.

Categories and Subject Descriptors: **Computing methodologies • Software and its engineering**~ *Software creation and management~ Software verification and validation~ Formal software verification* • **Computing methodologies** ~ **Machine learning**

General Terms: Patterns

Additional Key Words and Phrases: Machine Learning Application System, Architectural Pattern, Design Pattern

ACM Reference Format:

Milhem, H. and B. Harrison, N. 2024. Software Engineering Patterns for Machine Learning Application Systems. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 24 (October 2024), 13 pages.

1. INTRODUCTION

In the rapidly evolving landscape of technology, Machine Learning (ML) has emerged as a transformative force, reshaping industries and revolutionizing the way we interact with data. Every successful ML application relies not only on sophisticated algorithms but also on carefully chosen writing patterns to ensure clarity, efficiency, and maintainability. By understanding and harnessing these patterns, engineers and data scientists can: enhance their approach to designing and deploying ML systems, effectively document and communicate various aspects of ML solutions, and Integrate strategies, methodologies, and best practices in articulating the architecture, design, and functionality of ML-driven solutions Chouliaras et al. (2023). This comprehensive understanding ultimately drives innovation and delivers impactful solutions in today's data-driven world.

Washizaki et al. (2019) performed a systematic literature review to collect, classify, and discuss the software-engineering (SE) design patterns for ML techniques. They collected good/bad SE design patterns for ML techniques to provide developers with a comprehensive classification of such patterns. They write down some of the detected patterns in the pattern format based on Harrison (2006), Meszaros and Doble (1996), Wellhausen et al. (2012) such as "Different Workloads in Different Computing Environments", "Encapsulate ML Models Within Rule-base Safeguards", "Data Flows Up, Model Flows Down", Distinguish Business Logic from

This work is supported by the Widget Corporation Grant #312-001.
Author's address: H. Milhem, Information Technology Department, Faculty of Prince Al-Hussein Bin Abdallah II For Information Technology, The Hashemite University, P.O. Box 330127, Zarqa 13133, Jordan; email: hinda_is@hu.edu.jo; N. B. Harrison, Department of Computer Science, Utah Valley University, Orem, Utah, 201 Presidents Circle Salt Lake City, UT 84112, USA; email: Neil.Harrison@uvu.edu.

ML Model (originally named as “Multi-Layer Architectural Pattern”, “Data Lake”, “Microservice Architecture” and “ML Versioning”.

In this paper, we document two other patterns within machine learning application systems based on Harrison (2006), Meszaros and Doble (1996), Wellhausen et al. (2012), examining their role in fostering collaboration, enhancing readability, and facilitating the seamless integration of complex components. These patterns are “*Representative Validation Data*” and “*Gateway Routing*”. Figure 1 shows the two patterns we documented in this paper and their relationships with their related patterns.

1.1 The Related Patterns between Representative Validation Data and Gateway Routing

The *Representative Validation Data* and *Gateway Routing* patterns are not directly connected, as they serve distinct roles and are classified differently *Representative Validation Data* as a design pattern and *Gateway Routing* as an architectural pattern. Consequently, they are somewhat distant from each other in the design space. However, certain patterns bridge the gap between them, connecting both patterns within a larger framework. As illustrated in Figure 1, the Separation of Concerns and Modularization of ML Components pattern links these two, with both patterns utilizing it in their respective solutions.

Additionally, Service Mesh and Microservices patterns complement the *Gateway Routing* pattern, supporting its implementation and integration. The *Representative Validation Data* pattern also intersects with several other patterns, as shown in Figure 1, each contributing to a cohesive approach to developing, deploying, and maintaining reliable machine learning systems. By understanding and applying these related patterns, organizations can build robust, scalable, and dependable machine learning solutions. Further details on these related patterns are provided in Sections 2.8 and 3.8.

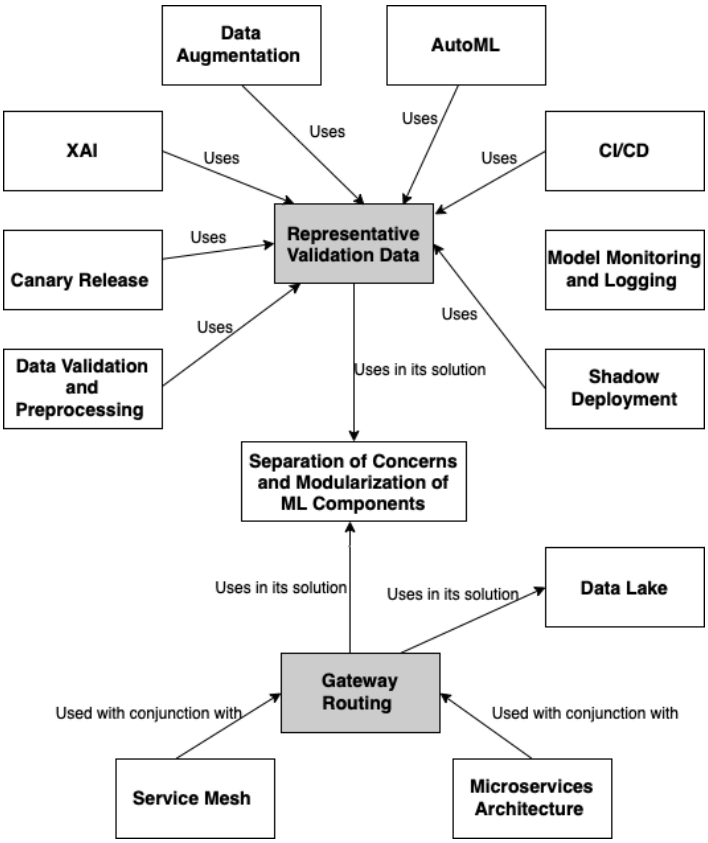

Fig. 1. The relationships of Representative Validation Data pattern, Gateway Routing pattern, and their related patterns

2. REPRESENTATIVE VALIDATION DATA PATTERN

In the following, we document the pattern based on Harrison (2006), Meszaros and Doble (1996), Wellhausen et al. (2012).

2.1 Intent

This pattern designed to ensure the accuracy, reliability, and trustworthiness of the model's outputs before they are integrated into a larger system or used for decision-making.

2.2 Context

- A trained model exists, ready for evaluation.
- Testing and validation datasets are available, containing samples with known ground truth labels or outcomes.

2.3 Problem

The *Representative Validation Data* pattern addresses the need for robust model evaluation to ensure machine learning models generalize well to unseen data which is data that the machine learning model has not encountered during its training phase, (i.e. new, previously unobserved data), and produce accurate outputs consistent with expected outcomes.

2.4 Forces

The forces of the Representative Validation Data pattern are:

- Data Preparation and Augmentation: This involves ensuring a diverse and representative dataset and using techniques for data augmentation to increase variability.
- Model Selection and Architecture: Which means choosing the right model architecture for the problem and comparing different models to find the best fit.
- Validation Techniques: This includes splitting data into training, validation, and test sets.
- Post-Deployment Monitoring: Which means monitoring model performance on live data and retraining and updating the model as new data becomes available.
- Bias and Variance Analysis: Which means understanding and addressing the bias-variance trade-off.
- Data Preparation and Preprocessing: This means ensuring that the dataset is clean and well-prepared. Additionally, splitting the data into training, validation, and test sets.
- Model Training: This includes training the model using the training dataset and using techniques such as cross-validation to ensure robustness.
- Initial Model Evaluation: Which means evaluating the model on the validation set and Use performance metrics such as accuracy, precision, and recall.
- Isolate Outputs: Isolating the model's predictions for individual data points and comparing these predictions against the expected outcomes.
- Final Model Evaluation: Conducting a final evaluation on the test set and reporting comprehensive metrics to summarize model performance.

This pattern can be split into several parts. Here are the main components:

- Training Data Set:

- This is the initial dataset used to train the model. It has no context.
- It contains input-output pairs (features and labels) that the model learns from.
- The goal is to minimize the error on this dataset during the training process.

    - Validation Data Set:

- This dataset is separate from the training data. However, the training data set considers as the context for the validation data set.
- It is used to tune hyperparameters and make decisions about the model architecture.
- Helps in assessing the model's performance during the training phase to prevent overfitting.

    - Test Data Set:

- This dataset is entirely separate from both the training and validation sets.
- It is used to evaluate the final model's performance after training.
- Provides an unbiased estimate of the model's generalization to new, unseen data.

    - Trained Model:

- This is the model that has been trained on the training data, which can be the context or the input to the next pattern, Gateway Routing pattern.
- It has learned the patterns and relationships from the training data.

    - Model Validation:

- The process of evaluating the model on the validation dataset.
- Involves calculating performance metrics like accuracy, precision, recall, etc.

    - Hyperparameter Tuning:

- The process of optimizing hyperparameters using the validation dataset.
- Involves selecting the best model parameters that improve performance on the validation set.

    - Model Testing:

- The final evaluation of the trained model on the test dataset.
- Provides a measure of how well the model is expected to perform on new, unseen data.

    - Performance Metrics:

- Quantitative measures used to evaluate the model’s performance.
- Common metrics include accuracy, precision, recall, etc.

    - Model Deployment:

- The process of putting the trained and validated model into production. This also can be the context of the next pattern, the Gateway Routing pattern.
- Involves monitoring the model's performance on real-world data and retraining as necessary.

The *Representative Validation Data* pattern provides a comprehensive approach to ensure that machine learning models are both accurate and generalizable. By focusing on the diversity and representativeness of the validation data, it offers non-obvious strategies like continuous validation, real-time monitoring, and detailed output analysis, which are critical in real-world applications. These techniques collectively help in identifying the right model architecture, balancing bias and variance, and ensuring the model remains robust over time.

2.5 Solution

The solution of this pattern is ensuring the validation and accuracy of a machine learning model and generalization to unseen data by testing its predictions in an isolated environment, comparing them against known outcomes, and ensuring it meets performance benchmarks before deployment to production.

Figure 2 shows the entire structure of the solution components to solve the problem shown in section 2.3. The starting point is the collection of input data, which can be historical data with known outcomes or newly gathered data intended for testing the model. The input data is processed by the machine learning model in a separate, isolated environment. This isolation ensures that the test does not interfere with the live production environment. Then, the model generates predictions based on the input data. These outputs remain isolated and are not yet part of the production system. The isolated outputs are then validated against predefined benchmarks or ground truth data. This step involves comparing the model's predictions with known, accurate results to evaluate performance.

Key performance metrics such as accuracy, precision, and recall are calculated to quantify the model's effectiveness Kla¨s and Vollmer (2018). These metrics help determine if the model meets the required standards. The model's validation status is determined based on performance Metrics. If the model meets the benchmarks (passes), it is considered ready for production. If it fails, further debugging and adjustments are necessary. If the model does not pass validation, this step involves analyzing the discrepancies and adjusting the model to improve performance. This could include refining the model, retraining with more data, or modifying the input features. Once the model passes validation, it is integrated into the production environment. This final step involves deploying the model to operate on live data and perform its intended tasks within the operational system.

By following this approach, the solution ensures that the machine learning model generalizes well to unseen data and produces accurate, consistent outputs, thereby addressing the outlined problems effectively.

2.6 Example

A financial institution has developed a machine learning model to detect fraudulent transactions. Given the high stakes associated with false positives (legitimate transactions being flagged as fraud) and false negatives (fraudulent transactions going undetected), it is crucial to ensure that the machine learning model generalizes well to unseen data and the model's outputs are reliable and accurate before deploying it into the production environment.

In this example, the pattern's structured approach to collecting data, processing it in isolation, generating predictions, and validating those predictions against benchmarks helps the financial institution enhance the robustness and maintain the reliability and effectiveness of the fraud detection system, thereby safeguarding against potential financial losses and maintaining customer trust.

Following the approach in Figure 2, in fraud detection system, historical transaction data with known fraud labels is collected. Additionally, new data can be gathered to test the model. Then, the fraud detection

model processes the input data in a separate, isolated environment. This ensures that testing does not interfere with the live production system. After that, the model generates predictions based on the input data, identifying transactions as either fraudulent or legitimate. Finally, the model's predictions are validated against ground truth data, which includes known fraudulent and legitimate transactions. This comparison helps evaluate the model's performance.

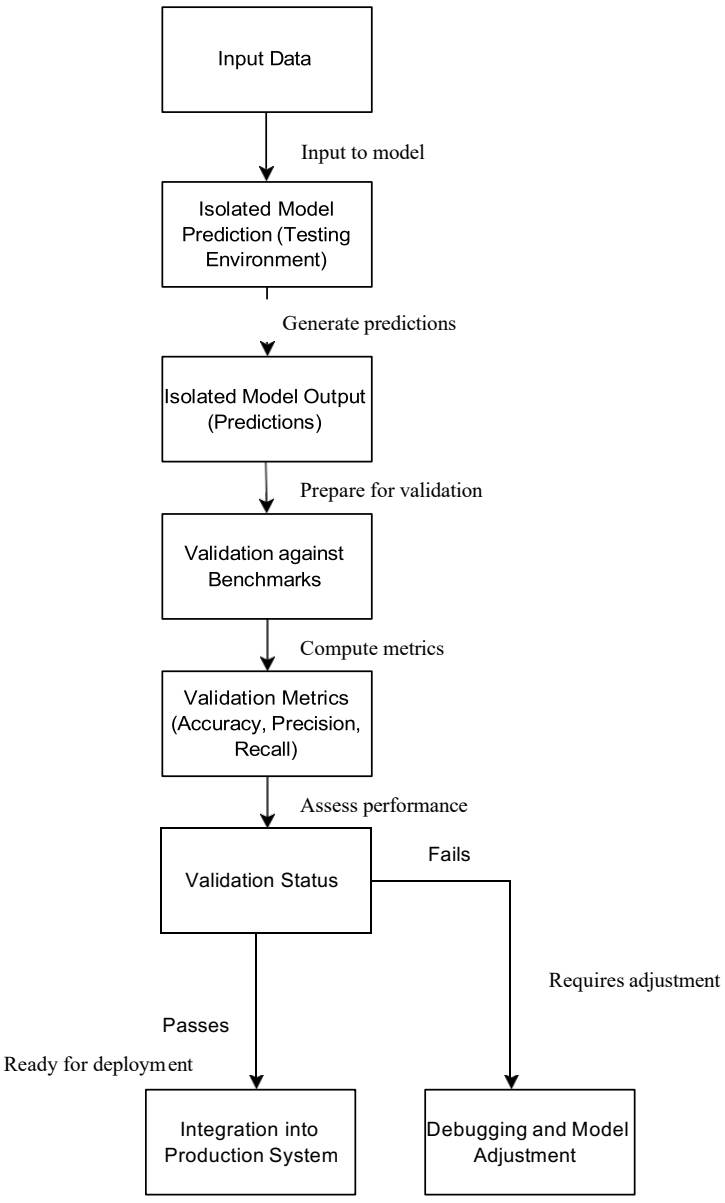

Fig. 2. Structure of the *Representative Validation Data* pattern

2.7 Discussion

The *Representative Validation Data* pattern is essential for building reliable and trustworthy machine learning application systems. Although it comes with certain challenges, its benefits—such as enhanced model reliability, quality assurance, and scalability—are significant. By adhering to best practices, organizations can successfully implement this pattern to improve their machine learning deployment and integration processes.

The challenges associated with this pattern include:

- **Bias in Data Selection**: If the validation data does not fully represent real-world scenarios, the model's performance may degrade in deployment.
- **Overfitting to Validation Data**: There's a risk of the model becoming overly fine-tuned to the validation set, reducing its ability to generalize.
- **Data Drift**: As data distributions change over time, this approach may miss new patterns, necessitating continuous updates and monitoring.

2.8 Related Patterns

Representative Validation Data pattern intersects with various other patterns, each contributing to a comprehensive framework for developing, deploying, and maintaining reliable machine learning systems. By understanding and leveraging these related patterns, organizations can create robust, scalable, and trustworthy machine learning solutions.

The related patterns are:

1- **Model Monitoring and Logging[1] :** while **Model Monitoring and Logging** focuses on ongoing performance and operational health, *Representative Validation Data* emphasizes pre-deployment validation to ensure model reliability. Together, they create a robust framework for managing machine learning models in production.

2- **Shadow Deployment[2] :** while the **Shadow Deployment** pattern ensures the model can handle real-world data and traffic without user impact, the *Representative Validation Data* pattern ensures the model's predictions are accurate and reliable. Together, they provide a comprehensive validation strategy for deploying machine learning models.

3- **Canary Release[3] :** It reduces the risk of widespread failure, allows for real-world testing and feedback, and provides a controlled environment to monitor and resolve issues. The **Canary Release** pattern focuses on software or system updates, while *Representative Validation Data* pattern specifically targets machine learning models and their predictions.

4- **Data Validation and Preprocessing** (Swarup et al. 2018): It is essential part of a robust machine learning workflow. Both patterns work together to ensure that a model is trained on high-quality data and that its predictions are reliable and generalize well to new data.

5- **Automated Machine Learning (AutoML)** Salehin et al. (2024): while **AutoML** accelerates the model development process by automating tasks, the *Representative Validation Data* pattern ensures that the resulting models are accurate, robust, and generalize well to new data, thereby enhancing overall model reliability and performance.

6- **Continuous Integration/Continuous Deployment (CI/CD) for Machine Learning[4] : CI/CD** for ML focuses on automation and deployment efficiency, while *Representative Validation Data* pattern focuses on ensuring the reliability and performance of deployed models in real-world applications. Both are crucial for robust ML deployment and maintenance.

---

[1] https://www.evidentlyai.com/ml-in-production/model-monitoring

[2] https://neptune.ai/blog/model-deployment-strategies

[3] https://www.getambassador.io/blog/comprehensive-guide-to-canary-releases

[4] https://medium.com/infer-qwak/ci-cd-for-machine-learning-in-2024-best-practices-to-build-test-and-deploy-c4ad869824d2

7- **Explainable AI (XAI) Pattern** Ali et al. (2023): Both **(XAI)** pattern and the pattern for *Representative Validation Data* can complement each other: *Representative Validation Data* pattern helps ensure accuracy, while XAI techniques help explain why certain outputs occur, providing insights into the model's behavior and aiding in diagnosing any issues found during validation.
8- **Data Augmentation** A. Mumuni and F. Mumuni (2022): **Data augmentation** involves creating new training samples by applying various transformations to the existing data (e.g., rotations, flips, translations for images). Augmenting data can help ensure that the validation data remains representative by increasing the diversity of training samples and reducing overfitting.

3. GATEWAY ROUTING PATTERN

In the following, we document the pattern based on Harrison (2006), Meszaros and Doble (1996), Wellhausen et al. (2012).

3.1 Intent

Gateway Routing pattern aims to manage the complexity of service communication in a distributed system by centralizing the routing logic through a gateway. This pattern helps to decouple clients from services, enabling easier management of service endpoints, versioning, and security.

3.2 Context

In machine learning application systems, especially those deployed in production environments, there are often multiple services and models that need to be accessed by end-users or other applications. These services and models might be implemented using different frameworks, hosted on different platforms, or require different access protocols. The trained and evaluated models, which come from the first pattern (i.e. Representative Validation Data pattern), can be the input or the context of this pattern. Efficiently routing requests to the appropriate service or model while maintaining scalability, flexibility, and manageability is a significant challenge.

3.3 Problem

The problem is how can we efficiently manage and route requests to various machine learning models and services in a scalable and flexible manner?

The Gateway Routing pattern is a powerful tool in microservices architecture, providing simplicity, centralized control, and security. However, it comes with trade-offs related to performance, complexity, and potential bottlenecks. The non-obvious benefits, such as improved decoupling and centralized management of security and monitoring, make it a valuable pattern when used appropriately, particularly in systems with complex, dynamic microservice environments. Balancing these trade-offs requires careful planning and understanding of the system's specific needs and traffic patterns.

3.4 Forces

The forces driving the adoption of the Gateway Routing pattern are:

1. Scalability

    - Ensuring that the system can handle increasing numbers of users and requests by balancing load across multiple services or instances.
    - Decoupling clients from backend services to enable independent scaling of services.

2. Flexibility in Service Access

    - Providing dynamic routing capabilities, such as routing requests based on user roles, device types, or service versions.
    - Supporting multiple protocols or APIs (e.g., REST, gRPC, WebSockets) through a unified gateway.

3. Maintainability and Modularity

    - Centralizing access logic, reducing redundancy across client applications.
    - Isolating backend service changes from directly impacting clients.

4. Security

    - Acting as a barrier to unauthorized access by enforcing authentication, authorization, and encryption (e.g., TLS termination).
    - Concealing the internal architecture and backend service endpoints from clients.

5. Monitoring and Observability

    - Aggregating logs and metrics at the gateway level to provide a unified view of traffic and system performance.
    - Enabling distributed tracing and diagnostics for complex systems.

6. Backward Compatibility and Versioning

    - Allowing support for different versions of services or APIs without requiring immediate updates to all clients.
    - Simplifying the transition to new versions of services.

7. Performance Optimization

    - Caching responses at the gateway level to reduce latency and backend load.
    - Compressing and optimizing payloads for better network utilization.

8. Resilience and Fault Tolerance

    - Implementing retries, timeouts, and circuit breakers at the gateway to handle service failures gracefully.
    - Providing fallback mechanisms for degraded experiences when backend services are unavailable.

9. Cross-Cutting Concerns

    - Standardizing cross-cutting concerns such as rate limiting, logging, and monitoring in a central component rather than duplicating them across services.

10. Ease of Integration

    - Simplifying integration with external systems or third-party APIs by acting as a single point of negotiation and transformation for protocols and data formats.

3.5 Solution

The solution of this pattern is managing all incoming requests and routing them to the appropriate backend services or models.

Figure 3 shows the entire structure of the solution components to solve the problem shown in Section 3.3. The discussions of the main steps shown in Figure 3 of the entire solution are shown below.

- User Interaction: The user initiates a request through the user interface.
- API Gateway: The request is received by the API gateway, which routes it to the appropriate services. This centralization simplifies the routing logic and ensures efficient management of the requests.
- Authentication: The request passes through the authentication service to verify user credentials. This step ensures that the user credentials are verified, providing a secure environment and preventing unauthorized access.
- Input Validation: The validated request is then forwarded to the input validation service. This service ensures that the incoming data adheres to the required format and standards, preventing erroneous or malicious data from propagating through the system.
- Data Processing: The data undergoes preprocessing and feature engineering to prepare it for model inference. This critical step transforms raw data into a suitable format for model inference, enhancing the accuracy and efficiency of the predictions.
- Model Inference: The prepared data is sent to the model inference service to generate predictions. This step is the core of the architecture, where the actual computation and analysis happen.
- Post-processing: The inference results are postprocessed to produce the final output.
- Data Storage: Relevant data and results are stored. This ensures that all interactions and predictions are recorded systematically.
- Monitoring & Logging: The entire process is monitored and logged for performance tracking and debugging. This enables real-time performance tracking, debugging, and ensuring that the system operates as expected.

This architecture ensures a modular, scalable, and maintainable system where each service can be independently developed, deployed, and scaled Yokoyama (2019). This flexibility allows for efficient handling of varying loads and the seamless integration of new models and services as the system evolves. By centralizing the request management through the API gateway and distributing the processing across specialized services, this solution addresses the core problem of efficiently managing and routing requests in a scalable manner.

3.6 Example

E-commerce Platform with Machine Learning Services. An e-commerce platform utilizes multiple machine learning models for various tasks like product recommendations, fraud detection, and customer sentiment analysis. Each model may have its own set of requirements and operational characteristics. The platform has millions of users, requiring a robust and scalable solution to handle the high volume of requests and efficiently route them to the appropriate model. The system needs to handle these efficiently without performance degradation. The platform also should be able to easily integrate new models and update existing ones without disrupting the overall service.

Following the solution of the *Gateway Routing* pattern shown in section 3.5, the gateway in the recommendation system receives a request for product recommendations. It routes the request to the recommendation model, which may be a collaborative filtering model or a deep learning-based model. The gateway ensures that the request is processed by the most appropriate and available model instance.

For a payment request, the gateway routes it to the fraud detection model, which uses machine learning to assess the risk of the transaction. If the primary model is overloaded, the gateway can redirect the request to a secondary instance. When sending personalized marketing emails, the gateway routes the requests to a customer segmentation model that categorizes users based on their behavior and preferences.

By implementing a gateway routing pattern, an e-commerce platform can effectively manage the complexity and demands of multiple machines learning services, ensuring a scalable, flexible, and efficient system.

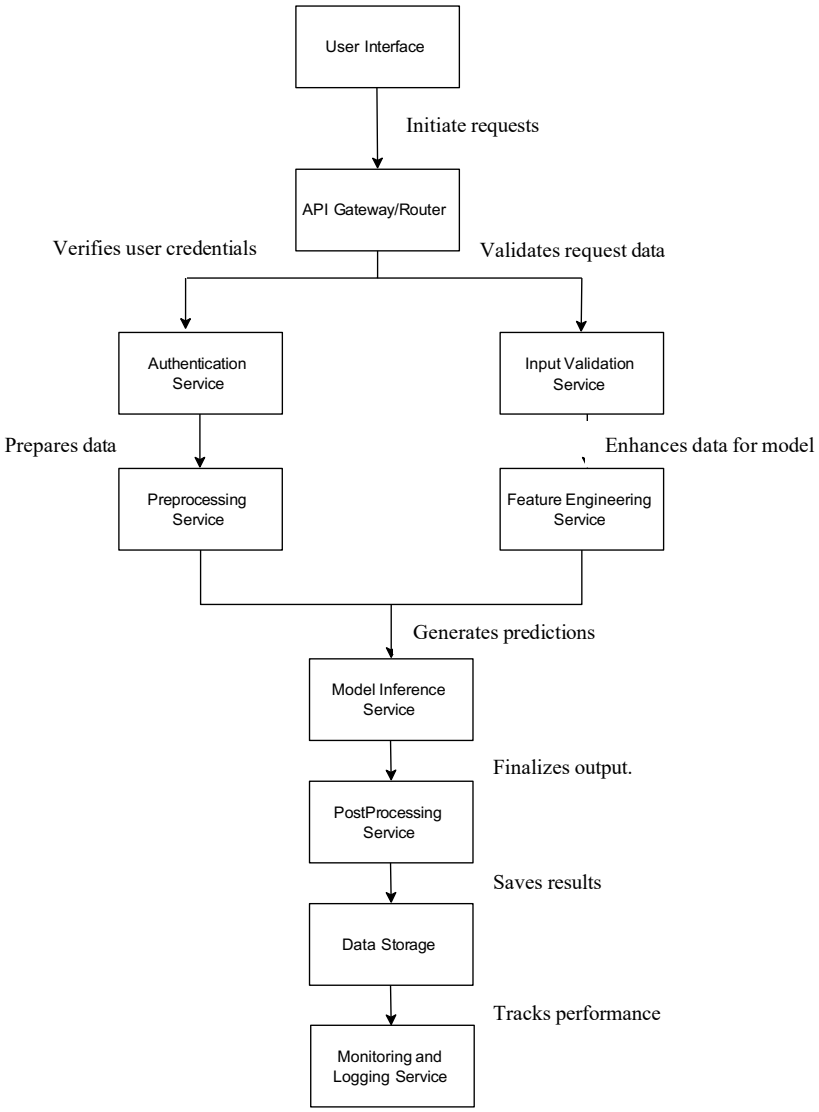

Fig. 3. Structure of the Architecture for Gateway Routing pattern

3.7 Discussion

This pattern enables efficient and adaptable routing of requests within complex machine learning application systems, supporting scalability, security, and maintainability. However, it introduces some challenges:

- **Increased Complexity**: Adding a routing layer can complicate the system, heightening the risk of configuration errors.
- **Latency**: Routing requests through a gateway may increase latency, which can be problematic in time-sensitive applications.
- **Single Point of Failure**: If not carefully managed and scaled, the gateway may become a bottleneck or a single point of failure.

3.8 Related Patterns

1. **Microservices Architecture** Victor and Pamela (2023): Often used in conjunction with the *Gateway Routing* pattern to manage multiple services. The *Gateway Routing* pattern complements the Microservices Architecture pattern by providing a centralized entry point and managing concerns like routing, security, and monitoring, thereby enhancing the overall scalability and manageability of a microservices-based application.
2. **Service Mesh** Maia and Correia (2022): Provides additional capabilities like service-to-service communication, security, and observability, complementing the gateway, while the *Gateway Routing* pattern manages external client interactions with the microservices ecosystem. Together, they form a robust infrastructure for scalable, secure, and observable microservices architectures.

4. CONCLUSION

In this paper, we have documented and explored two significant patterns for machine learning application systems: the "Representative Validation Data" and "Gateway Routing" patterns. These patterns address critical aspects of model reliability, scalability, and maintainability within ML-powered architectures. The Representative Validation Data pattern ensures model accuracy and robustness by facilitating structured validation and performance assessment, thereby improving the reliability of models before deployment. Meanwhile, the Gateway Routing pattern centralizes request management, enhancing modularity and scalability in complex service ecosystems.

Together, these patterns provide a comprehensive framework that enhances the effectiveness and efficiency of deploying ML systems in production environments. By adhering to these structured approaches, organizations can better manage the complexities of ML applications, fostering a robust infrastructure that supports innovation while ensuring secure, scalable, and maintainable solutions. As ML systems continue to evolve, these patterns offer a foundational methodology to address the growing demands for accuracy, scalability, and seamless integration across services.

ACKNOWLEDGMENTS

We would like to express our sincere gratitude to our shepherd, Steve Berczuk, for his invaluable guidance, insightful feedback, and continuous support throughout the development of this paper. His expertise and encouragement have significantly contributed to shaping the final version of this work.

We would also like to thank all the reviewers at the PLoP conference who provided detailed and constructive feedback at each stage. Their thoughtful comments and suggestions were instrumental in refining our ideas and improving the clarity and quality of our paper. We appreciate the time and effort they dedicated to helping us bring this work to fruition. Thank you all for your invaluable contributions.

REFERENCES

Hironori Washizaki, Hiromu Uchida, Foutse Khomh, Yann-Gaël Guéhéneuc. 2019. Studying software engineering patterns for designing machine learning systems, *In The 10th International Workshop on Empirical Software Engineering in Practice (IWESEP 2019),* Tokyo, Japan, 2019, pp. 1–6.

Michael Kla ̈s and Anna M. Vollmer. 2018. Uncertainty in machine learning applications: A practice-driven classification of uncertainty. In Computer Safety, Reliability, and Security - SAFECOMP 2018 Workshop. ASSURE, DECSoS, SASSUR, STRIVE, and WAISE, Va ̈stera ̊s, Sweden, September 18, 2018, Proceedings, 2018, pp. 431–438. [Online]. Available: https://doi.org/10.1007/ 978-3-319-99229-7 36.

Haruki Yokoyama. 2019. Machine learning system architectural pattern for improving operational stability. in *IEEE International Conference on Software Architecture Companion, ICSA Companion 2019, Hamburg, Germany, March 25-26, 2019*, 2019, pp. 267–274. [Online].

Neil B. Harrison. 2006. Advanced Pattern Writing – Patterns for Experienced Pattern Authors, In: Pattern Languages of Program Design 5, edited by Dragos Manolescu, Markus Voelter, James Noble. Addison-Wesley, 2006. http://hillside.net/europlop/HillsideEurope/Papers/EuroPLoP2003/2003_Harrison_ AdvancedPatternWriting.pdf.

Gerhard Meszaros, Jim Doble. 1996. A Pattern Language for Pattern Writing. PLoP-95. http://hillside.net/index.php/a-pattern-language-for-pattern-writing.

Tim Wellhausen, Andreas Fiesser.2012. How to write a pattern? a rough guide for first-time pattern authors", EuroPLoP '11: Proceedings of the 16th European Conference on Pattern Languages of Programs, 2012, 10.1145/2396716.2396721.

Chouliaras, Georgios, Kiełczewski, Kornel, Beka, Amit, Konopnicki, David and Bernardi, Lucas. 2023. Best Practices for Machine Learning Systems: An Industrial Framework for Analysis and Optimization. 10.13140/RG.2.2.31255. 14249.

Roy, Swarup, Sharma, Pooja, Nath, Keshab, Bhattacharyya, Dhruba K and Kalita, Jugal. 2018. Pre-Processing: A Data Preparation Step. 10.1016/B978-0-12-809633-8.20457-3.

Imrus Salehin, Md. Shamiul Islam, Pritom Saha, S.M. Noman, Azra Tuni, Md. Mehedi Hasan, Md. Abu Baten. 2024. AutoML: A systematic review on automated machine learning with neural architecture search, Journal of Information and Intelligence, Volume 2, Issue 1, 2024, Pages 52-81, ISSN 2949-7159, https://doi.org/10.1016/j.jiixd.2023.10.002.

Sajid Ali, Tamer Abuhmed, Shaker El-Sappagh, Khan Muhammad, Jose M. Alonso-Moral, Roberto Confalonieri, Riccardo Guidotti, Javier Del Ser, Natalia Díaz-Rodríguez, Francisco Herrera. 2023. Explainable Artificial Intelligence (XAI): What we know and what is left to attain Trustworthy Artificial Intelligence, Information Fusion, Volume 99, 2023, 101805, ISSN 1566-2535. https://doi.org/10.1016/j.inffus.2023.101805.

Alhassan Mumuni and Fuseini Mumuni. 2022. Data augmentation: A comprehensive survey of modern approaches, Array, Volume 16, 2022, 100258, ISSN 2590-0056, https://doi.org/10.1016/j.array.2022.100258.

Velepucha, Victor and Flores, Pamela. 2023. A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges. IEEE Access. PP. 1-1. 10.1109/ACCESS.2023.3305687.

João Tiago Duarte Maia and Filipe Figueiredo Correia. 2022. Service Mesh Patterns. *In 27th European Conference on Pattern Languages of Programs (EuroPLop '22)*, July 6–10, 2022, Irsee, Germany. ACM, New York, NY, USA 12. Pages. https://doi.org/10.1145/3551902.3551962.