Software Architecture and Design for AI-based systems

SOEN 691: Engineering AI-based Software Systems

Emad Shihab, Diego Elias Costa
Concordia University

What is software architecture?

Software architecture

- Defines how the system is organized

- Breaks the entire system into a set of communicating modules

What is software architecture?

Architecture in the small:

- At the “single system” level
- Defines how a large system is broken down into subsystems

Architecture in the large:

- At the “collection of systems” level
- Defines how a system of systems is broken down into individual systems

What are Architectural Styles?

- An **Architectural Style** defines a **family of systems** in terms of a **pattern of structural organization**.

Why Architectural Styles

- Makes for an easy way to communicate among stakeholders
- Documentation of early design decisions
- Allow for the reuse and transfer of qualities to similar systems

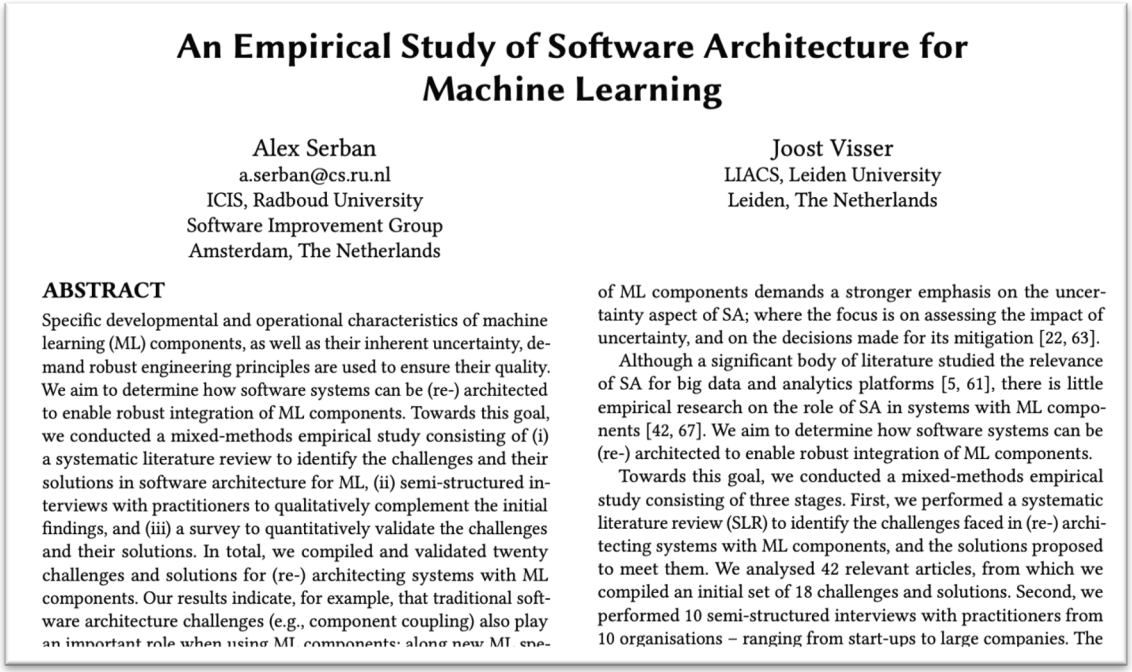
What is a Design Pattern

- A **Design Pattern** systematically *names*, *explains*, and *evaluates* an important and recurring design
- “descriptions of communicating objects that are customized to **solve a general problem** in a particular **context**”

Classifying Design Patterns

- **Structural**: concern the process of **assembling** objects and classes (e.g., façade, adapter, etc.)
- **Behavioral**: concern the **interaction** between classes or objects (e.g., iterator, observer, etc.)
- **Creational**: concern the process of object **creation** (e.g., abstract factory)

Empirical Study of Software Architecture for Machine Learning



An Empirical Study of Software Architecture for Machine Learning

Alex Serban
a.serban@cs.ru.nl
ICIS, Radboud University
Software Improvement Group
Amsterdam, The Netherlands

Joost Visser
LIACS, Leiden University
Leiden, The Netherlands

ABSTRACT

Specific developmental and operational characteristics of machine learning (ML) components, as well as their inherent uncertainty, demand robust engineering principles are used to ensure their quality. We aim to determine how software systems can be (re-) architected to enable robust integration of ML components. Towards this goal, we conducted a mixed-methods empirical study consisting of (i) a systematic literature review to identify the challenges and their solutions in software architecture for ML, (ii) semi-structured interviews with practitioners to qualitatively complement the initial findings, and (iii) a survey to quantitatively validate the challenges and their solutions. In total, we compiled and validated twenty challenges and solutions for (re-) architecting systems with ML components. Our results indicate, for example, that traditional software architecture challenges (e.g., component coupling) also play an important role when using ML components; along new ML spe-

of ML components demands a stronger emphasis on the uncertainty aspect of SA; where the focus is on assessing the impact of uncertainty, and on the decisions made for its mitigation [22, 63].

Although a significant body of literature studied the relevance of SA for big data and analytics platforms [5, 61], there is little empirical research on the role of SA in systems with ML components [42, 67]. We aim to determine how software systems can be (re-) architected to enable robust integration of ML components.

Towards this goal, we conducted a mixed-methods empirical study consisting of three stages. First, we performed a systematic literature review (SLR) to identify the challenges faced in (re-) architecting systems with ML components, and the solutions proposed to meet them. We analysed 42 relevant articles, from which we compiled an initial set of 18 challenges and solutions. Second, we performed 10 semi-structured interviews with practitioners from 10 organisations – ranging from start-ups to large companies. The

Motivation

- ML systems have different needs, i.e.
  - Serving
  - Monitoring
  - Retraining
  - Redeployment

- Uncertainty of ML systems is key for SA

Introduction

- Study is based on a SLR, semi-structured interviews and a survey
- Two RQs:
  - What are the **challenges** when (re) architecting software systems with ML?
  - What **solutions**, tactics or patterns successfully address these challenges?
- Identified 20 challenges and solutions

Study Design

- **Vocabulary**
  - Automatic queries
  - Gray literature
  - Snowball strategy
  - Inclusion/exclusion strategies
  - Thematic analysis
  - Multiple Choice answers from SLR
  - Purposeful sampling
  - Survey Pilot
  - Any others…?

Results - Req

Nr.	Category	Challenges	Solutions	References
1	Reqs.	At design time the information available is insufficient to understand the customers or the projects.	Run simulations to gather data. Use past experience. Measure and document uncertainty sources.	[10, 17, 31, 41, 42]
2	Reqs.	ML components lack functional requirements.	Use metrics as functional requirements. Include understandability and explainability of the outputs.	[10, 17, 20, 31, 42]
3	Reqs.	ML projects have regulatory restrictions and may be subject to audits.	Analyse regulatory constraints up-front. Adopt an AI code of conduct. Design audit trails.	[24, 34, 49, 64]

Results - Data

4	Data	Data preparation may result in a jungle of scrapes, joins, and sampling steps, often with intermediate outputs.	Design separate modules/services for data collection and data preparation. Integrate external tools.	[20, 40, 60]
5	Data	Data quality is hard to test, and may have unexpected consequences.	Design separate modules/services for data quality assessment. Integrate external tools.	[20, 40, 45, 54, 76]

Results - Design

6	Design	Separate concerns between training, testing, and serving, but reuse code between them.	Standardise model interfaces. Use one middleware. Reuse virtualisation, infrastructure and test scripts.	[2, 73, 77]
7	Design	Distinguish failures between ML components and other business logic.	Separate business logic from ML components. Standardise interfaces and use one middleware between them.	[55, 74]
8	Design	ML components are highly coupled, and errors can have cascading effects.	Design independent modules/services for ML and data. Standardise interfaces and use one middleware. Relax coupling heuristics between ML and data.	[29, 51, 67]
9	Design	ML components bring inherent uncertainty to a system.	Use n-versioning. Design and monitor uncertainty metrics. Employ interpretable models/human intervention.	[3, 29, 51, 63, 65]
10	Design	ML components can fail silently. These failures can be hard to detect, isolate and solve.	Use metric monitoring and alerts to detect failures. Use n-versioning. Employ interpretable models.	[11, 65, 72]
11	Design	ML components are intrinsically opaque, and deductive reasoning from the architecture artefacts, code or metadata is not effective.	Instrument the system to the fullest extent. Use n-versioning. Employ interpretable models. Design log modules to aggregate/visualise metrics.	[29, 51, 59, 77]
12	Design	Avoid unstructured components which link frameworks or APIs (e.g., glue code).	Wrap components in APIs/modules/services. Use standard interfaces and one middleware. Use virtualisation.	[60]
13	Design	Automation and understanding of ML tasks is difficult (AutoML).	Version configuration files. Design the log and versioning systems to support AutoML data retrieval.	[40, 57, 64, 67, 73]

Results - Testing

14	Testing	ML testing goes beyond programming bugs to issues that arise from model, data errors, or uncertainty.	Design model and data tests. Use CI/CD. Use integration and unit tests. Use data ownership for test modules.	[2, 4, 50, 56, 76]
15	Testing	Validation of ML components for production is difficult.	Use metrics and CI/CD for validation. Use alerts, visualisations, human intervention. Design release processes.	[58]

Results - Operations

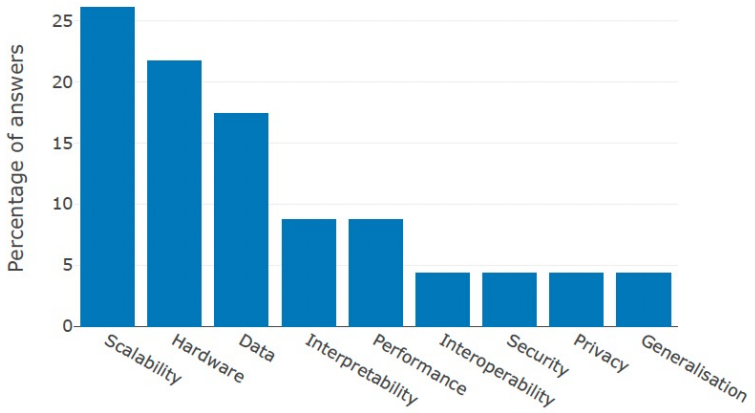
16	Ops.	ML components require continuous maintenance, retraining and evolution.	Design for automatic continuous retraining. Use CI/CD. Use automatic rollback. Use infrastructure-as-code. Adopt standard release processes.	[8, 41, 51, 58, 67, 69, 76]
17	Ops.	Manage the dependencies and consumers of ML applications.	Encapsulate ML components in identifiable modules/services. Use authentication and access control. Log consumers of ML components.	[7, 23, 29, 60, 74]
18	Ops.	Balance latency, throughput, and fault-tolerance, needed for training and serving.	Design for batch processing (training) and stream processing (serving), i.e., lambda architecture. Physically isolate the workloads. Use virtualisation.	[16, 40, 47, 68, 73]
19	Ops.	Trace back decisions to models, data and reproduce past results.	Design for traceability and reproducibility; log pointers to versioned artefacts, version configurations, models and data.	P10
20	Org.	ML applications use heterogeneous technology stacks which require diverse backgrounds and skills.	Form multi-disciplinary teams. Adopt an AI code of conduct. Define processes for decision-making. Raise awareness about ML risks within the team.	P1

Results

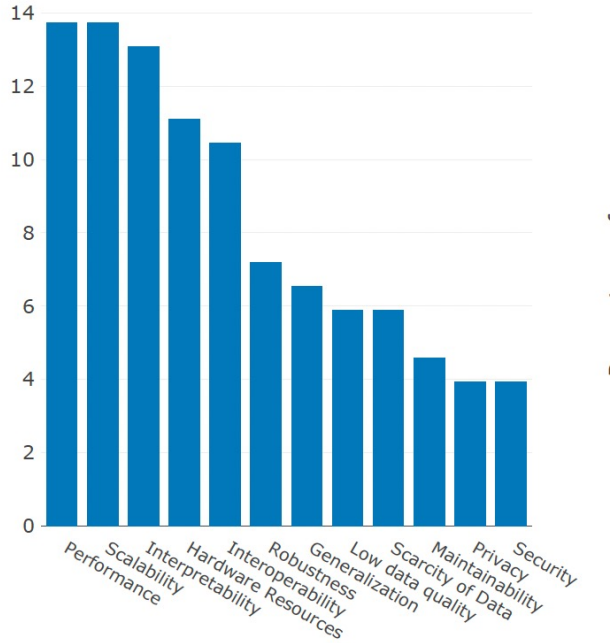
- **Requirements** - inability to understand a project and estimate the effort upfront. Also, regulatory restrictions, emerged as challenging.
- **Data** – data preparation and quality is important.
- **Design** –managing coupling, managing inherent uncertainty & development automation (traditional). Also, integration of ML with business logic.
- **Testing** – model testing, validation for production (accuracy, bias, robustness).
- **Operational** – Maintenance of ML is based on retraining and redeploying, eroding the boundaries between maintenance and evolution.

Results – Decision Drivers

Figure 7: Distribution of architectural decision drivers from interviews.



Interviews



(b) SA decision drivers.

Survey

Results – Impact

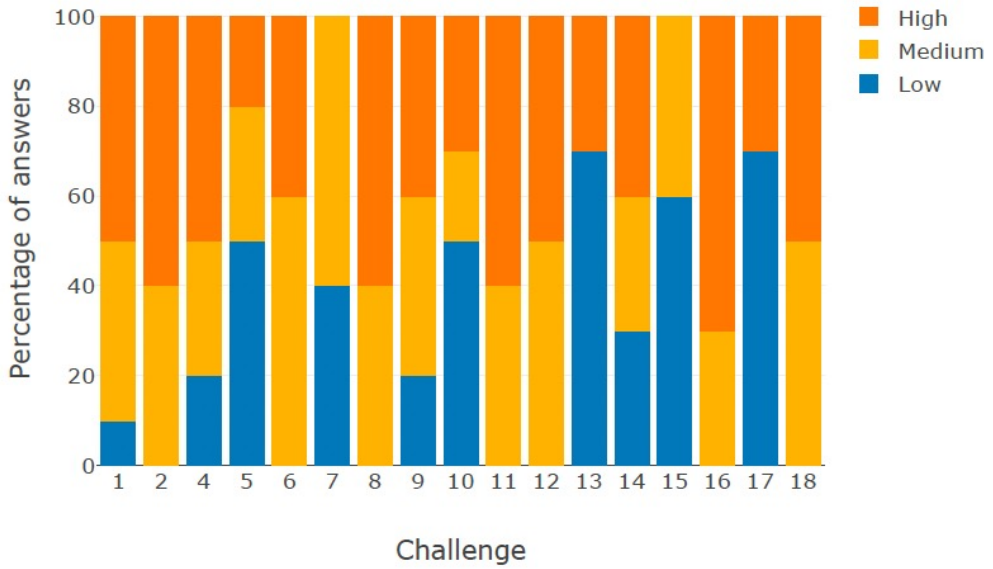
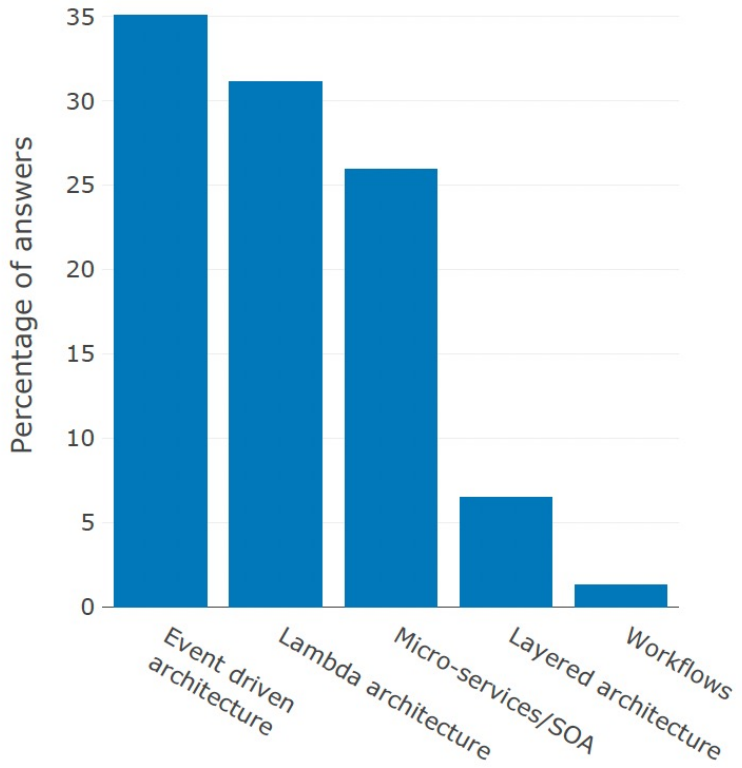


Figure 1: The impact of the challenges from Table 4 on SA, as assessed by interview participants.

16 (highest): continuous maintenance, retraining and evolution

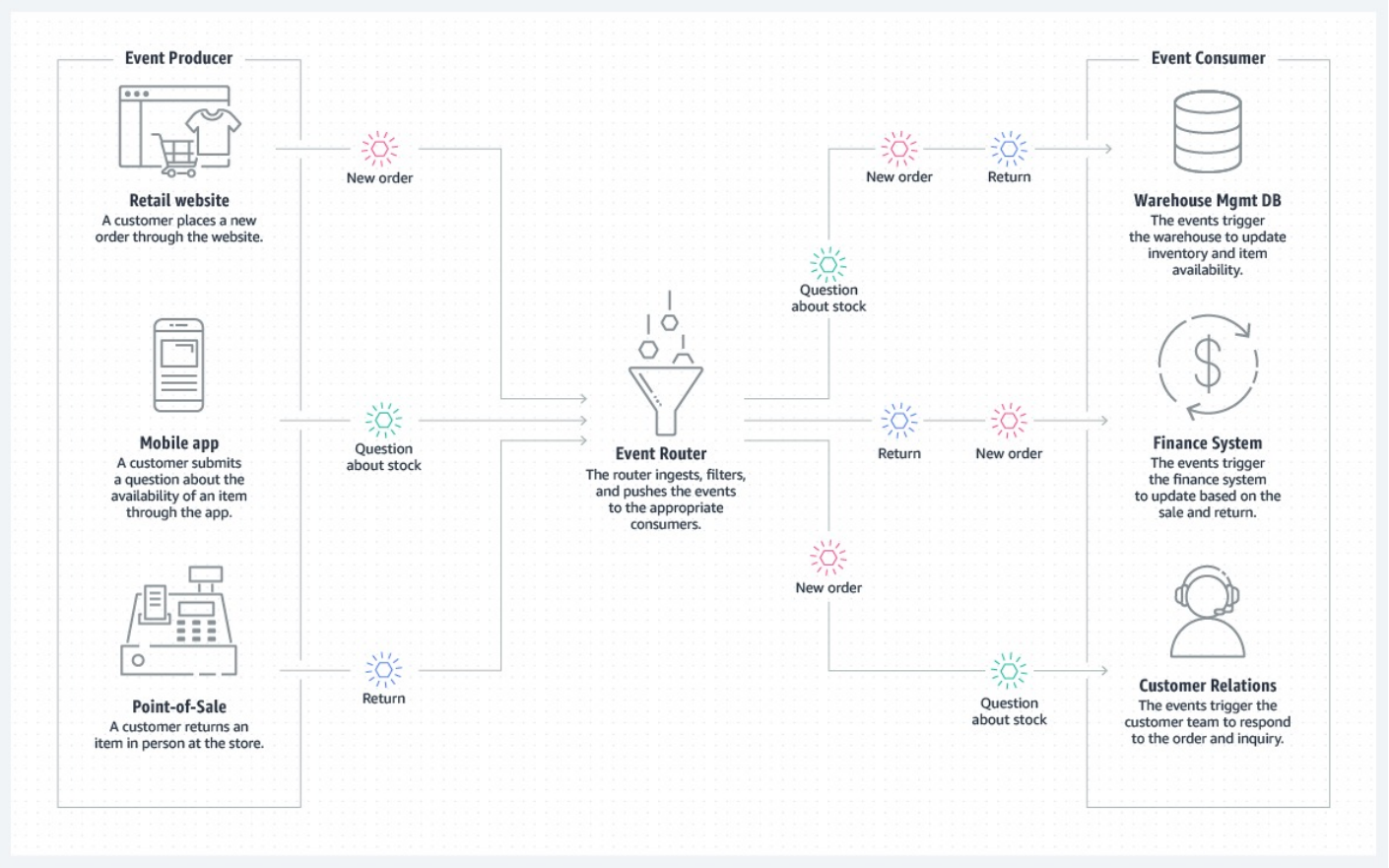
13 (lowest): Automation and understanding of of ML tasks is difficult

Results – Architectural Styles



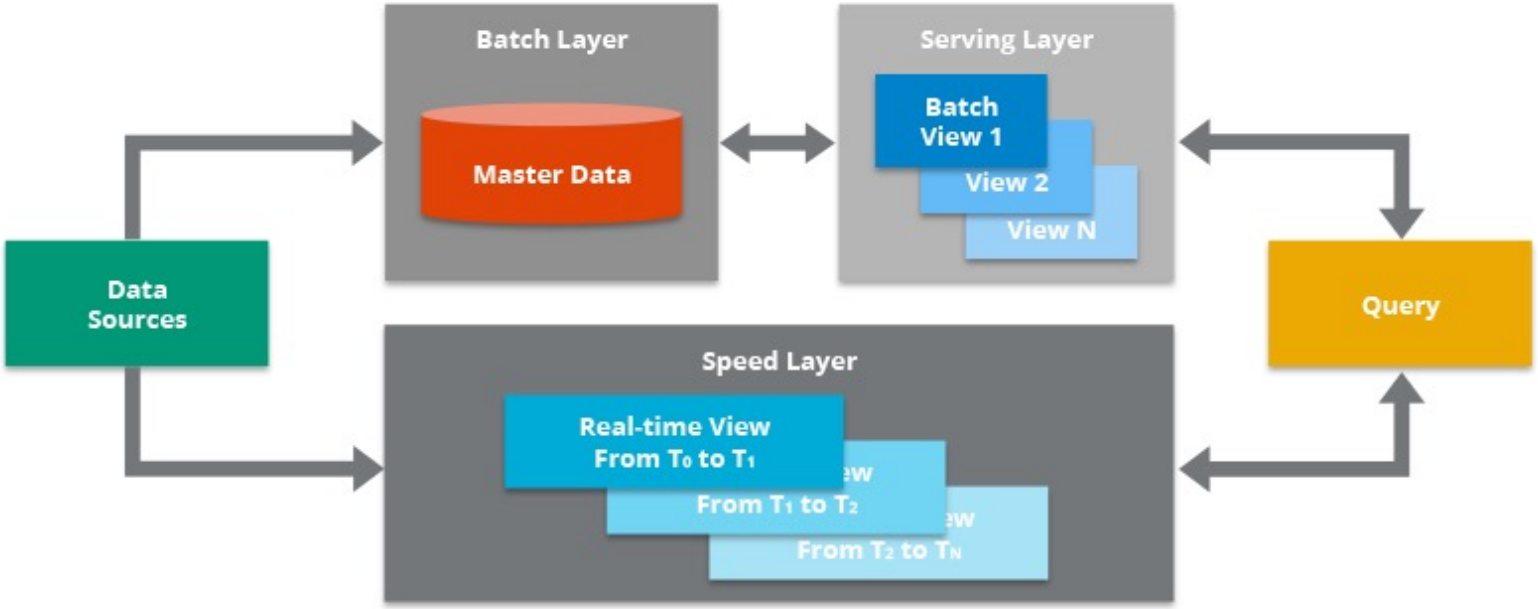
(c) Architectural styles.

Results – Event-driven



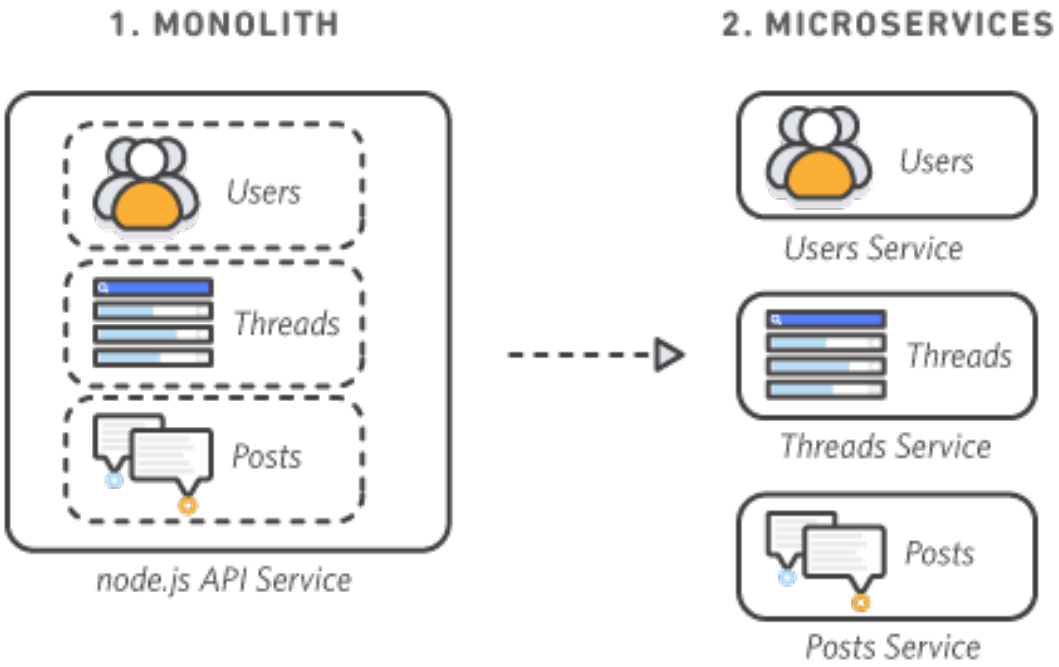
https://aws.amazon.com/event-driven-architecture/

Results – Lambda



https://hazelcast.com/glossary/lambda-architecture/

Results – Micro-services



https://aws.amazon.com/microservices/

Open Discussion

Machine Learning Architecture and Design Patterns

Machine Learning Architecture and Design Patterns

Hironori Washizaki
Waseda University / National Institute of Informatics / SYSTEM INFORMATION / eXmotion

Hiromu Uchida
Waseda University

Foutse Khomh
Polytechnique Montréal

Yann-Gaël Guéhéneuc
Concordia University

***Abstract*—Researchers and practitioners studying best practices strive to design Machine Learning (ML) application systems and software that address software complexity and quality issues. Such design practices are often formalized as architecture and design patterns by encapsulating reusable solutions to common problems within given contexts. In this paper, software-engineering architecture and design (anti-)patterns for ML application systems are analyzed to bridge the gap between traditional software systems and ML application systems with respect to architecture and design. Specifically, a systematic literature review confirms that ML application systems are popular due to the promotion of artificial intelligence. We identified 32 scholarly documents and 48 gray documents out of which 38 documents discuss 33 patterns: 12 architecture patterns, 13 design patterns, and 8 anti-patterns. Additionally, a survey of developers reveals that there are 7 major architecture patterns and 5 major design patterns. Then the relationships among patterns are identified in a pattern map.**

Introduction

- Goal: To find reusable solutions to common problems, particularly for ML systems

- Performed a SLR of academic and grey literature
  - Used 32 scholarly documents
  - 48 grey literature documents

Study Design

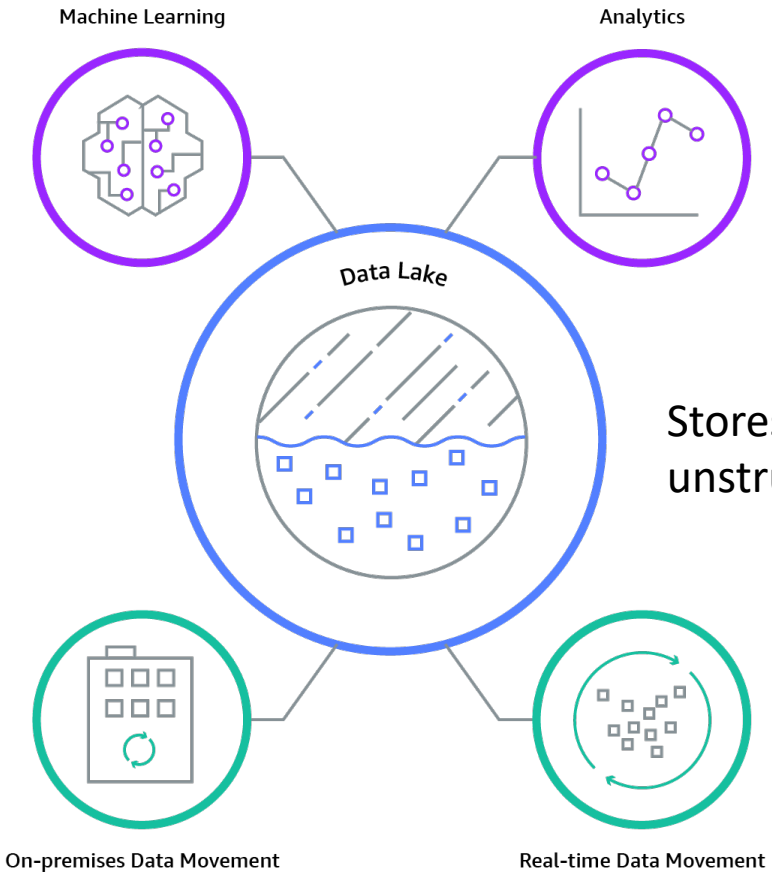
- Two authors read half of the documents and extracted patterns
- Rest of the authors vetted the patterns
- Identified 69 patterns and 33 remained after vetting
  - **ML architecture patterns**: address recurrent architectural problems such as ensuring maintainability of ML components
  - **ML design patterns**: address recurrent detailed design problems such as enabling proper communications among specific modules

Results – Architectural Patterns

Table 1. Extracted Architecture Patterns (NP: Number of participants who used the pattern.)

Pattern Name	Problem (excerpt)	Solution (excerpt)	NP and Source
Data Lake	We cannot foresee the kind of analyses that will be performed on the data and which frameworks will be used to perform these analyses.	The data ranging from structured data to unstructured data should be stored as "raw" as possible and the centralized data repository should allow parallel analyses of different kinds and with different frameworks.	5, http://bit.ly/33DTKTe
Distinguish Business Logic from ML Models	The overall business logic should be isolated as much as possible from the ML models so that they can be changed/overridden when necessary without impacting the rest of the business logic.	Separate the business logic and the inference engine, loosely coupling the business logic and ML-specific dataflows.	4, [3]
Microservice Architecture	ML applications may be confined to some "known" ML frameworks and miss opportunities for more appropriate frameworks.	Data scientists working with or providing ML frameworks can make these frameworks available through microservices.	4, http://bit.ly/2DyHGrV
Data-Algorithm-Serving-Evaluator	Prediction systems should connect different pieces in the data processing pipeline into one coherent system and prototyping predictive model.	Separate the following like MVC for ML: data (data source and data preparator), algorithm(s), serving, and evaluator.	2, http://bit.ly/2r6edmu
Event-driven ML Microservices	Due to frequent prototyping of ML models and constant changes, development teams must be agile to build, deploy, and maintain complex data pipelines.	Construct pipelines by chaining together multiple microservices, each of which listens for the arrival of some data and performs its designated task.	2, http://bit.ly/2OZDuXH
Lambda Architecture	Real-time data processing requires scalability, fault tolerance, predictability, and other qualities. It must be extensible.	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, merges it.	2, http://bit.ly/33DTKTe

Results – Data Lake



Stores raw structured and unstructured data

https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/

Results – Design Patterns

Table 2. Extracted ML Design Patterns (NP: Number of participants who used the pattern.)

Pattern Name	Problem (excerpt)	Solution (excerpt)	NP and Source
ML Versioning	ML models and their several versions may change the behaviour of the overall ML applications.	Record the ML model structure, training data, and training system to ensure a reproducible training process.	4, [7]
Wrap Black-Box Packages into Common APIs	Using generic, independent ML frameworks often results in different glue code for each framework, for which a massive amount of supporting code is written to get data into and out of the framework from and to the rest of the application.	Wrap black-box packages into common APIs to make supporting infrastructure more reusable and to reduce the cost of changing packages.	4, [4]
Test Infrastructure Independently from ML	It is difficult to identify errors when infrastructure and machine learning are mixed.	Ensure that the infrastructure is testable and the learning parts of the system are encapsulated so that everything around it can be tested.	3, http://bit.ly/34zt2wx
Handshake (Hand Buzzer)	A ML system depends on inputs delivered outside of the normal release process.	Create a handshake normalization process, regularly check for significant changes, and send ALERTS.	2, http://bit.ly/2qdsWvG
Isolate and Validate Output of Model	Machine learning models are known to be unstable and vulnerable to adversarial attacks and to noise in data and data drift overtime.	Encapsulate ML models within rule-base safeguards and use redundant and diverse architecture that mitigates and absorbs the low robustness of ML models.	2, [8]

Results – Anti-Patterns

Table 3. Extracted ML Anti-Patterns

Pattern Name	Problem (excerpt)	Source
Big Ass Script Architecture	When all code is placed in one big ass script, it becomes difficult to reuse in future analysis, understand how it works, and debug.	http://bit.ly/35QPb9N
Abstraction Debt	For distributed learning, widely accepted abstractions are lacking.	[4]
Dead Experimental Codepaths	The code-paths accumulated by individual changing can create a growing debt due to the increasing difficulties of maintaining backward compatibility.	[4]
Glue Code	Glue code is costly in the long term because it tends to freeze a system to the peculiarities of a specific package.	[4]
Multiple-Language Smell	Using multiple languages increases the cost of effective testing and can increase the difficulty of transferring ownership to other individuals.	[4]
Pipeline Jungles	The system to prepare data in an ML-friendly format may become a pipeline jungle, and managing these pipelines is difficult and costly.	[4]
Plain-Old-Data Type Smell	The rich information used and produced by ML systems is often encoded with plain data types like raw floats and integers.	[4]
Undeclared Consumers	Undeclared consumers are dangerous because they create a hidden tight coupling of model MA to other parts of the stack.	[4]

Final Exam Question

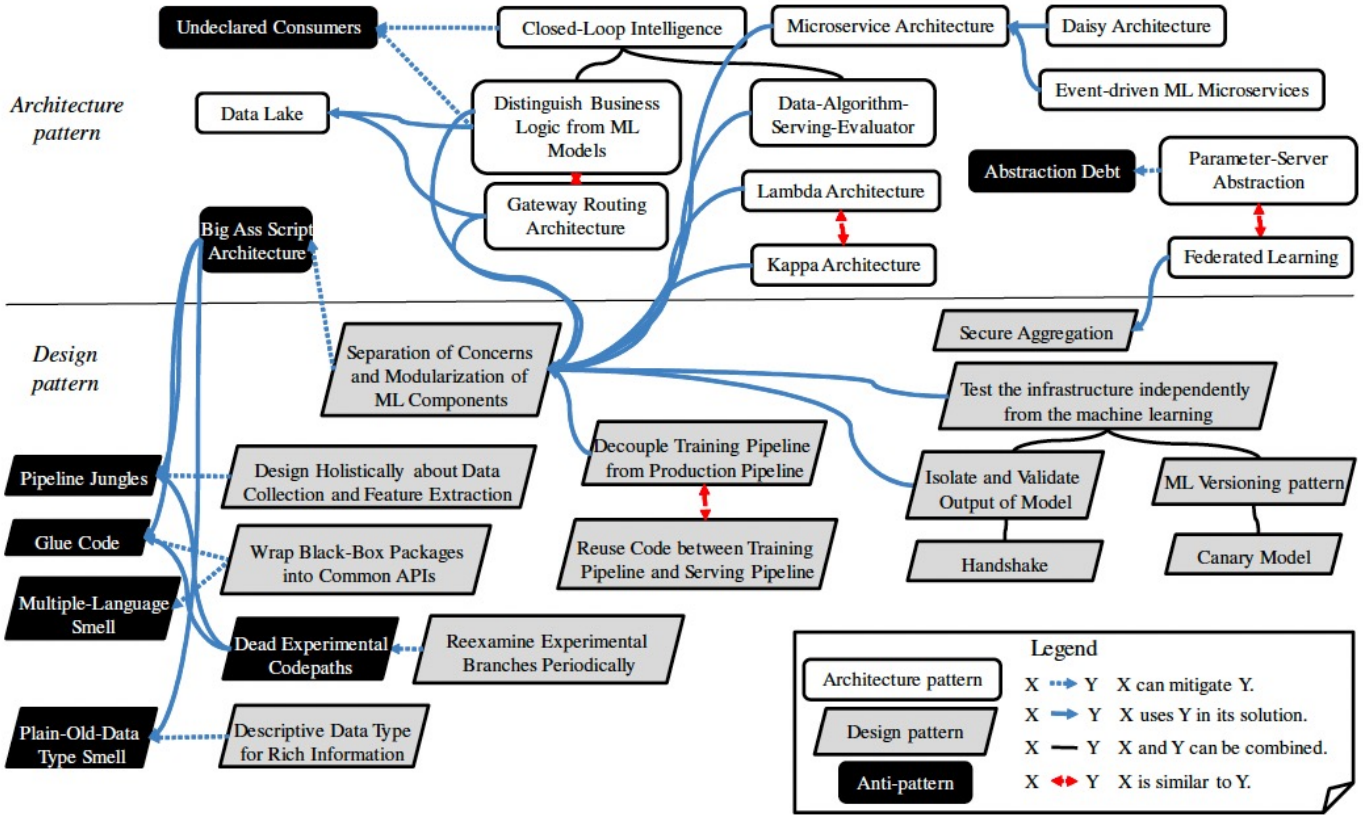


Figure 4. Pattern Map showing classifications and relationships among ML patterns

Open Discussion

Homework

- Two papers are posted on Moodle
- For one of the papers, write a summary (aprox. 1/3 of a page)
- For the other paper, write a critique, which includes a summary, at least 3 strong points and at least 3 weaknesses (aprox. 1 page).
- Submit your summary and critique on Moodle by Friday, Feb. 11th at 5pm

Backups

Results - SA

Nr.	Category	Challenges	Solutions	References
1	Reqs.	At design time the information available is insufficient to understand the customers or the projects.	Run simulations to gather data. Use past experience. Measure and document uncertainty sources.	[10, 17, 31, 41, 42]
2	Reqs.	ML components lack functional requirements.	Use metrics as functional requirements. Include understandability and explainability of the outputs.	[10, 17, 20, 31, 42]
3	Reqs.	ML projects have regulatory restrictions and may be subject to audits.	Analyse regulatory constraints up-front. Adopt an AI code of conduct. Design audit trails.	[24, 34, 49, 64]
4	Data	Data preparation may result in a jungle of scrapes, joins, and sampling steps, often with intermediate outputs.	Design separate modules/services for data collection and data preparation. Integrate external tools.	[20, 40, 60]
5	Data	Data quality is hard to test, and may have unexpected consequences.	Design separate modules/services for data quality assessment. Integrate external tools.	[20, 40, 45, 54, 76]
6	Design	Separate concerns between training, testing, and serving, but reuse code between them.	Standardise model interfaces. Use one middleware. Reuse virtualisation, infrastructure and test scripts.	[2, 73, 77]
7	Design	Distinguish failures between ML components and other business logic.	Separate business logic from ML components. Standardise interfaces and use one middleware between them.	[55, 74]
8	Design	ML components are highly coupled, and errors can have cascading effects.	Design independent modules/services for ML and data. Standardise interfaces and use one middleware. Relax coupling heuristics between ML and data.	[29, 51, 67]
9	Design	ML components bring inherent uncertainty to a system.	Use n-versioning. Design and monitor uncertainty metrics. Employ interpretable models/human intervention.	[3, 29, 51, 63, 65]
10	Design	ML components can fail silently. These failures can be hard to detect, isolate and solve.	Use metric monitoring and alerts to detect failures. Use n-versioning. Employ interpretable models.	[11, 65, 72]
11	Design	ML components are intrinsically opaque, and deductive reasoning from the architecture artefacts, code or metadata is not effective.	Instrument the system to the fullest extent. Use n-versioning. Employ interpretable models. Design log modules to aggregate/visualise metrics.	[29, 51, 59, 77]
12	Design	Avoid unstructured components which link frameworks or APIs (e.g., glue code).	Wrap components in APIs/modules/services. Use standard interfaces and one middleware. Use virtualisation.	[60]
13	Design	Automation and understanding of ML tasks is difficult (AutoML).	Version configuration files. Design the log and versioning systems to support AutoML data retrieval.	[40, 57, 64, 67, 73]
14	Testing	ML testing goes beyond programming bugs to issues that arise from model, data errors, or uncertainty.	Design model and data tests. Use CI/CD. Use integration and unit tests. Use data ownership for test modules.	[2, 4, 50, 56, 76]
15	Testing	Validation of ML components for production is difficult.	Use metrics and CI/CD for validation. Use alerts, visualisations, human intervention. Design release processes.	[58]
16	Ops.	ML components require continuous maintenance, retraining and evolution.	Design for automatic continuous retraining. Use CI/CD. Use automatic rollback. Use infrastructure-as-code. Adopt standard release processes.	[8, 41, 51, 58, 67, 69, 76]
17	Ops.	Manage the dependencies and consumers of ML applications.	Encapsulate ML components in identifiable modules/services. Use authentication and access control. Log consumers of ML components.	[7, 23, 29, 60, 74]
18	Ops.	Balance latency, throughput, and fault-tolerance, needed for training and serving.	Design for batch processing (training) and stream processing (serving), i.e., lambda architecture. Physically isolate the workloads. Use virtualisation.	[16, 40, 47, 68, 73]
19	Ops.	Trace back decisions to models, data and reproduce past results.	Design for traceability and reproducibility; log pointers to versioned artefacts, version configurations, models and data.	P10
20	Org.	ML applications use heterogeneous technology stacks which require diverse backgrounds and skills.	Form multi-disciplinary teams. Adopt an AI code of conduct. Define processes for decision-making. Raise awareness about ML risks within the team.	P1

Results – Architectural Patterns

Table 1. Extracted Architecture Patterns (NP: Number of participants who used the pattern.)

Pattern Name	Problem (excerpt)	Solution (excerpt)	NP and Source
Data Lake	We cannot foresee the kind of analyses that will be performed on the data and which frameworks will be used to perform these analyses.	The data ranging from structured data to unstructured data should be stored as "raw" as possible and the centralized data repository should allow parallel analyses of different kinds and with different frameworks.	5, http://bit.ly/33DTKTe
Distinguish Business Logic from ML Models	The overall business logic should be isolated as much as possible from the ML models so that they can be changed/overridden when necessary without impacting the rest of the business logic.	Separate the business logic and the inference engine, loosely coupling the business logic and ML-specific dataflows.	4, [3]
Microservice Architecture	ML applications may be confined to some "known" ML frameworks and miss opportunities for more appropriate frameworks.	Data scientists working with or providing ML frameworks can make these frameworks available through microservices.	4, http://bit.ly/2DyHGrV
Data-Algorithm-Serving-Evaluator	Prediction systems should connect different pieces in the data processing pipeline into one coherent system and prototyping predictive model.	Separate the following like MVC for ML: data (data source and data preparator), algorithm(s), serving, and evaluator.	2, http://bit.ly/2r6edmu
Event-driven ML Microservices	Due to frequent prototyping of ML models and constant changes, development teams must be agile to build, deploy, and maintain complex data pipelines.	Construct pipelines by chaining together multiple microservices, each of which listens for the arrival of some data and performs its designated task.	2, http://bit.ly/2OZDuXH
Lambda Architecture	Real-time data processing requires scalability, fault tolerance, predictability, and other qualities. It must be extensible.	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, merges it.	2, http://bit.ly/33DTKTe
Parameter-Server Abstraction	For distributed learning, widely accepted abstractions are lacking.	Distribute both data and workloads over worker nodes, while the server nodes maintain globally shared parameters, which are represented as vectors and matrices.	2, [4]
Daisy Architecture	The ability to scale content production processes must be acquired via the use of ML. Then the coverage of that tooling must be extended over as much of their remaining content.	Utilize Kanban, scaling, and microservice to realize pull-based, automated, on-demand, and iterative processes.	1, http://bit.ly/2DyHGrV
Gateway Routing Architecture	When a client uses multiple services, it can be difficult to set up and manage individual endpoints for each service.	Install a gateway before a set of applications, services, or deployments and use application layer routing requests to the appropriate instance.	1, [3]
Kappa Architecture	It is necessary to deal with huge amount of data with less code resource.	Support both real-time data processing and continuous reprocessing with a single stream processing engine.	1, http://bit.ly/37Xkguc
Closed-Loop Intelligence	It is necessary to address big, open-ended, time-changing or intrinsically hard problems.	Connect machine learning to the user and close the loop. Design clear interactions along with implicit and direct outputs.	0, http://bit.ly/2L8ZpdB
Federated Learning	Standard machine learning approaches require centralizing the training data on one machine or in a datacenter.	Employ Federated Learning, which enables mobile phones to collaboratively learn a shared prediction model while keeping all the training data on the device.	0, http://bit.ly/2qaRJk3

Results – Design Patterns

Table 2. Extracted ML Design Patterns (NP: Number of participants who used the pattern.)

Pattern Name	Problem (excerpt)	Solution (excerpt)	NP and Source
ML Versioning	ML models and their several versions may change the behaviour of the overall ML applications.	Record the ML model structure, training data, and training system to ensure a reproducible training process.	4, [7]
Wrap Black-Box Packages into Common APIs	Using generic, independent ML frameworks often results in different glue code for each framework, for which a massive amount of supporting code is written to get data into and out of the framework from and to the rest of the application.	Wrap black-box packages into common APIs to make supporting infrastructure more reusable and to reduce the cost of changing packages.	4, [4]
Test Infrastructure Independently from ML	It is difficult to identify errors when infrastructure and machine learning are mixed.	Ensure that the infrastructure is testable and the learning parts of the system are encapsulated so that everything around it can be tested.	3, http://bit.ly/34zt2wx
Handshake (Hand Buzzer)	A ML system depends on inputs delivered outside of the normal release process.	Create a handshake normalization process, regularly check for significant changes, and send ALERTS.	2, http://bit.ly/2qdsWvG
Isolate and Validate Output of Model	Machine learning models are known to be unstable and vulnerable to adversarial attacks and to noise in data and data drift overtime.	Encapsulate ML models within rule-base safeguards and use redundant and diverse architecture that mitigates and absorbs the low robustness of ML models.	2, [8]
Canary Model	A surrogate ML that approximates the behavior of the best ML model must be built to provide explainability.	Run the canary inference pipeline in parallel with the primary inference pipeline to monitor prediction differences.	1, http://bit.ly/35U0C0i
Decouple Training Pipeline from Production Pipeline	It is necessary to separate and quickly change the ML data workload and stabilize the training workload to maximize efficiency.	Physically isolate different workloads to different machines. Then optimize the machine configurations and the network usage.	1, [7]
Descriptive Data Type for Rich Information	The rich information used and produced by ML systems is often encoded with plain data types like raw floats and integers.	Design a robust system, where the model parameter knows if it is a log-odds multiplier or a decision threshold, and a prediction knows information about the model.	1, [4]
Design Holistically about Data Collection and Feature Extraction	The system to prepare data in an ML-friendly format may become a pipeline jungle. Managing these pipelines is difficult and costly.	Avoid pipeline jungles by thinking holistically about data collection and feature extraction that can dramatically reduce ongoing costs.	1, [4]
Reexamine Experimental Branches Periodically	The code-paths accumulated by individual changes can create a growing debt due to the increasing difficulties of maintaining backward compatibility.	Reexamine each experimental branch periodically to see what can be removed to eliminate glue code and pipeline jungles.	1, [4]
Reuse Code between Training Pipeline and Serving Pipeline	Training-serving skew can be caused by a discrepancy between how data in the training and serving pipelines are handled.	Reuse code between training pipeline and serving pipeline by preparing objects that store results in an understandable way for humans.	0, http://bit.ly/34zt2wx
Separation of Concerns and Modularization of ML Components	ML applications must accommodate regular and frequent changes to their ML components.	Decouple at different levels of complexity from simplest to most complex.	0, [9]
Secure Aggregation	The system needs to communicate and aggregate model updates in a secure, efficient, scalable, and fault-tolerant way.	Encrypt data from each mobile device in Federated learning and calculate totals and averages without individual examination.	0, http://bit.ly/2qaRJk3