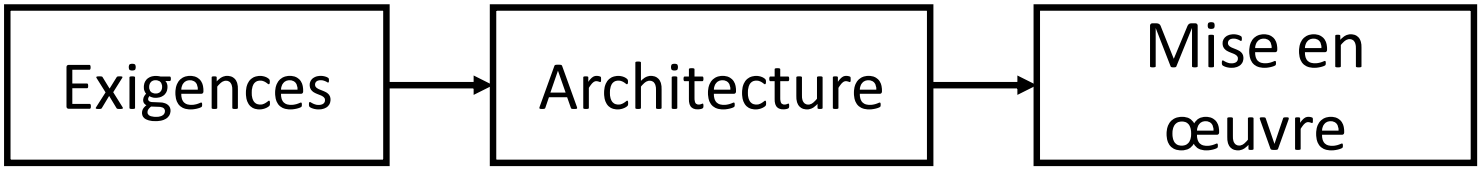
Architecture logicielle de systèmes d'IA

MGL7320 : Ingénierie logicielle des systèmes d'intelligence artificielle

Des exigences à la mise en œuvre



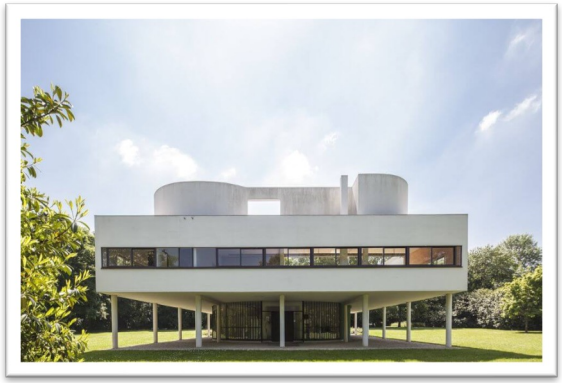
- Jusqu'à présent...
  - Identifiez les objectifs du système.
  - Mesures de succès
  - Considérations relatives à…
    - les exigences en matière de données
    - explicabilité
    - libre de la discrimination

Architecture logicielle des systèmes basés sur l'IA

Qu’est-ce que l’architecture?

Style gothique

Style méso-américain



Style moderne

Qu'est-ce que l'architecture logicielle ?

« L'architecture logicielle est l'organisation fondamentale d'un système, incarné dans ses **composants**, leurs **relations** les uns avec les autres et l'environnement, et les **principes** qui guident sa conception et son évolution. »

IEEE Standard 1471-2000

- Définit la façon dont le système est organisé
- Divise l'ensemble du système en un ensemble de modules de communication

Qu'est-ce que l'architecture logicielle ?

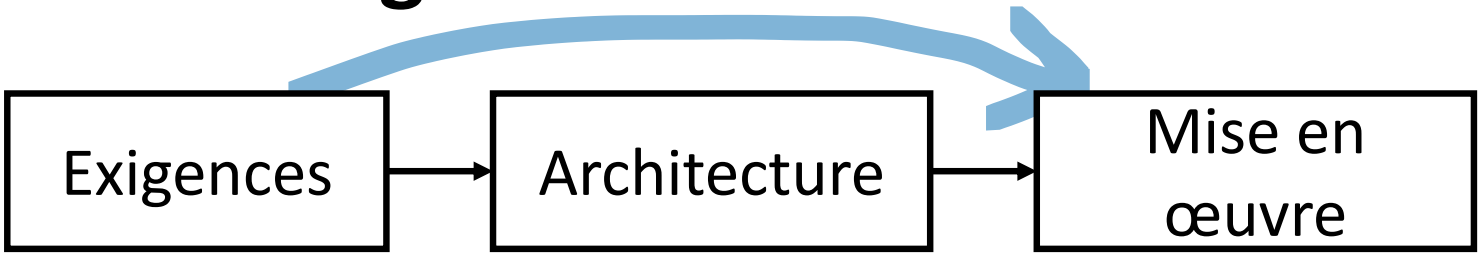
L'architecture dans les petits:

- Au niveau d'un « système »
- Définit comment un grand système est décomposé en sous-systems

L'architecture dans le grand:

- Au niveau de la « collection de systèmes »
- Définit comment un système de systèmes est décomposé en systèmes individuels

Pourquoi avons-nous besoin d'une architecture logicielle?

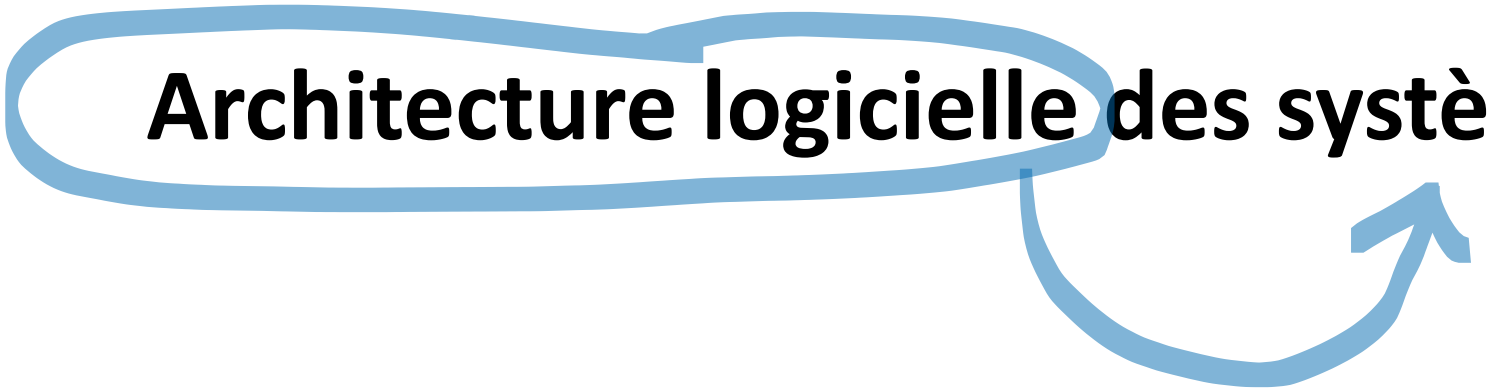


- Nous savons quoi construire, mais **comment**?

- **L'architecture:**
  - Principales décisions de conception
  - Très tôt
  - Axé sur les **qualités clés**

Les décisions architecturales sont **difficiles** à changer plus tard!

Architecture logicielle des systèmes d'IA



Comment définir une architecture d’un système d’IA?

- *Réfléchissez avant de coder*
  - Concentrez-vous tôt sur les qualités les plus importantes.
  - Laissez de l’espace pour la flexibilité autant que possible.

Trade-offs

Mettre l'accent sur les qualités clés du système

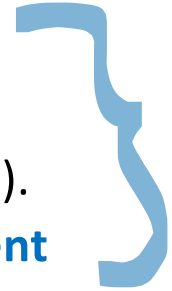
Guidé par les exigences, identifiez les qualités les plus importantes.

- Exemples:
  - **Coût :** Coût de développement, coût opérationnel, temps de libération
  - **Performances :** évolutivité, disponibilité, temps de réponse, débit
  - **Robustesse :** Sécurité, sûreté, facilité d'utilisation, équité
  - **Maintenabilité:** Facilité de modifications et de mises à jour
  - **ML:** Précision, capacité à collecter des données, latence d'entraînement

Considérations lors de la création de l'architecture des systèmes d'IA

Où mettre le modèle?

- Latence dans la **recyclage** du modèle
- Latence dans l'**exécution** du modèle
- Le **coût d'exécution** du modèle (distribution, inférence).
- Que se passe-t-il lorsque les utilisateurs se **déconnectent**



Comment les composants sont-ils organisés??

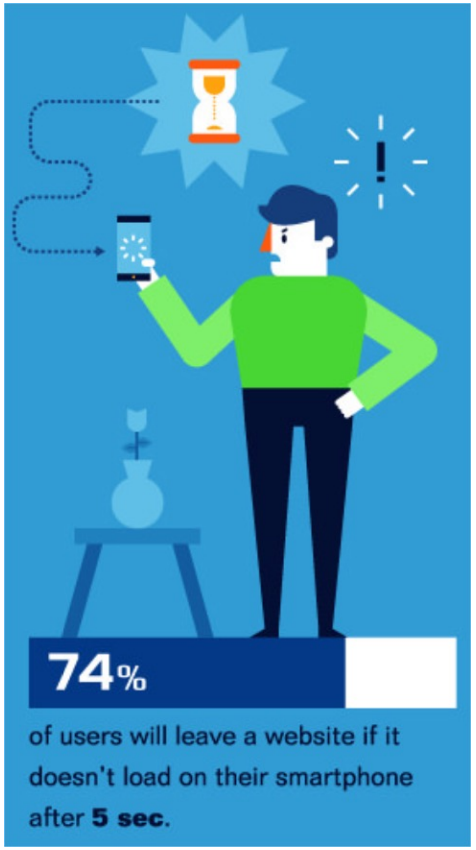
- Styles architecturaux

Latence dans le renouvellement du modèle

- À quelle fréquence votre modèle doit-il être renouvelé ?
  - La qualité du modèle change rapidement
    - P. ex., le déploiement d'un produit complètement nouveau
  - Le problème évolue rapidement.
    - P. ex., top-charts chansons, des attaques de sécurité
  - Risque d'erreurs coûteuses
    - P. ex., vies humaines, décisions à enjeux élevés

Latence dans l'exécution du modèle

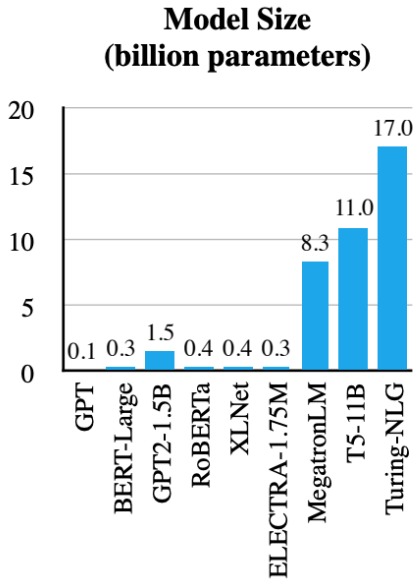
- La latence peut ruiner l'expérience utilisateur
  - Rend le système lent
- La latence a également un impact sur le choix du modèle
  - Temps d'inférence du modèle



Le coût d'exécution du modèle

- Coût de distribution du modèle
  - Le recyclage et distribution des modèles coûtent beaucoup d'argent.

- Coût d'exécution du modèle
  - Frais de location de serveurs



Formation de grands modèles le coût entre le

$80k - $1.6m!

(Modèle de paramètres de 1,5 milliard)

OpenAI's ChatGPT Reportedly Costs $100,000 a Day to Run

https://arxiv.org/pdf/2004.08900.pdf

Que se passe-t-il lorsque les utilisateurs se déconnectent

- Le système doit-il fonctionner déconnecté d'internet ?
  - Les systèmes moins critiques peuvent s'abstenir de faire des prédictions s'ils sont hors ligne.

- Les systèmes critiques doivent être disponibles 24/7.
  - Les modèles doivent être du côté du client
  - Les modèles mis en cache peuvent servir de solutions temporaires

Considérations lors de la création de l'architecture des systèmes d'IA

Où mettre le modèle?

- Latence dans la **recyclage** du modèle
- Latence dans l'**exécution** du modèle
- Le **coût d'exécution** du modèle (distribution, inférence).
- Que se passe-t-il lorsque les utilisateurs se **déconnectent**

Comment les composants sont-ils organisés??

- Styles architecturaux

Où mettre le modèle?

- Modèle statique dans le client (pas de recyclage)
  - Le modèle est construit et déployé une fois dans le produit
- Côté client
  - Le modèle est déployé dans le client
- Côté serveur
  - Le modèle est déployé sur un serveur centralisé
- Hybride
  - Les modèles sont principalement servis à partir d'un serveur
  - Les versions sont mises en cache sur le client.

Modèle statique dans le produit

Rassemblez un tas de données d'entraînement, produisez un modèle, regroupez-le avec votre logiciel et expédiez-le.

- Comme la construction d'un programme traditionnel

Latence dans le redéploiement : Pauvre
Latence dans l'exécution du modèle : Excellent
Le coût d'exécution du modèle : Pas cher
Opération hors ligne : Yes

Résumé des désavantages :
Risque d'erreurs non mesurées.
Aucune donnée pour améliorer le modèle.

Architecture de modèle côté client

- Le modèle s'exécute complètement sur le client
  - Mise à jour périodique

Latence dans le redéploiement : Variable
Latence dans l'exécution du modèle : Excellent
Le coût d'exécution du modèle : Pas cher / Basé sur le taux de mise a jour
Opération hors ligne : Yes

Résume des désavantages :
Difficile de garder les clients synchronisés.
Les ressources des clients peuvent être limitées.
Expose le modèle (sécurité)

Architecture de modèle côté serveur

- Modèle en tant que service
  - Le client obtient le contexte et les données d'entrée
  - Envoie au serveur qui exécute la prédiction du modèle

Latence dans le redéploiement : **Excellent**
Latence dans l'exécution du modèle : Variable
Le coût d'exécution du modèle : Infrastructure de service
Opération hors ligne : Non

Résume des désavantages :
Latence dans l'exécution
Coût de l'exécution des serveurs.
Pas de modèle hors ligne.

Architecture hybride

Plusieurs façons de combiner des modèles côté client et serveur

- prédictions bon marché (client), prédictions coûteuses (serveur)
- prédictions triviales (client) et décisions à enjeux élevés (serveur)

Latence dans le redéploiement : Variable
Latence dans l'exécution du modèle : Variable
Le coût d'exécution du modèle : Basé sur le volume d'utilisation
Opération hors ligne : Partiel

Résume des désavantages :
Plus complexe à construire et à orchestrer.

Considérations lors de la création de l'architecture des systèmes d'IA

Où mettre le modèle?

- Latence dans le **redéploiement** du modèle
- Latence dans l'**exécution** du modèle
- Le **coût d'exécution** du modèle (distribution, inférence).
- Que se passe-t-il lorsque les utilisateurs se **déconnectent**

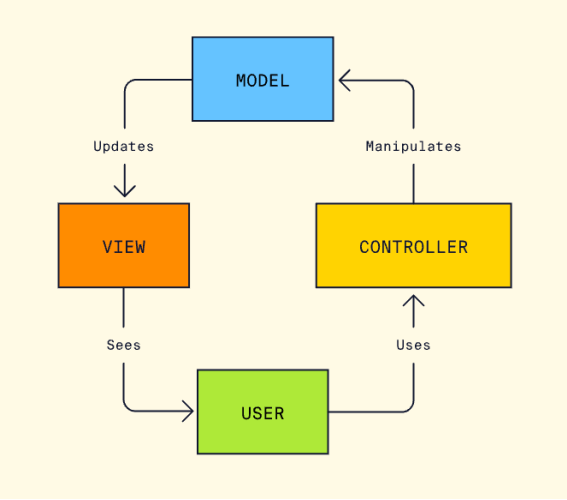
Comment les composants sont-ils organisés??

- Styles architecturaux

Que sont les styles architecturaux?

- Un style architectural définit une **famille de systèmes** en termes de modèle d'organisation structurelle.

Systèmes traditionnels



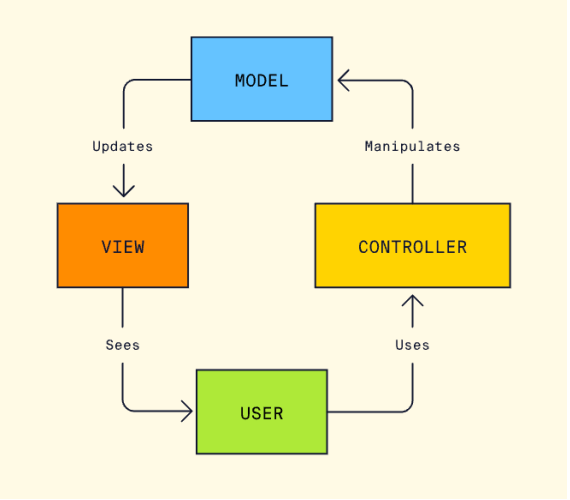
Systèmes d'IA

Où mettons-nous le modèle?
Et les données de formation / service?
Comment séparer les données système et les données ML?

Pourquoi les styles architecturaux

- Plus facile à **communiquer** aux intervenants
- Documentation des premières décisions de conception
- Permettre la **réutilisation** et le **transfert** de qualités vers des systèmes similaires

- Principes des composants
  - Encapsulation
  - Découplage
  - Communication par les interfaces

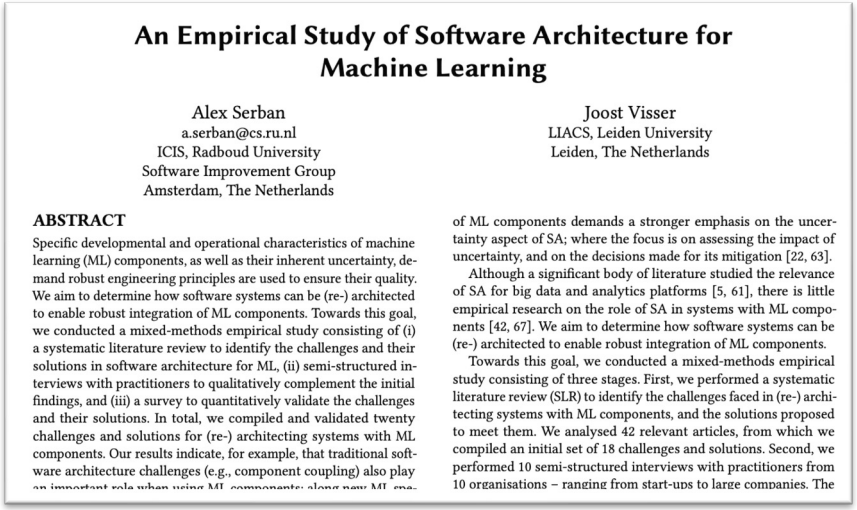


Empirical Study of Software Architecture for Machine Learning

Objectif:

- Quels sont les **défis** de l'architecture de systèmes logiciels avec ML?

- Quelles **solutions**, tactiques ou modèles répondent avec succès à ces défis?



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

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

Conception de l'étude

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

Résultats (aperçu)

- **Exigences** - incapacité de comprendre un projet et d'estimer l'effort dès le départ. Les restrictions réglementaires sont apparues comme étant difficiles.
- **Données** – la préparation et la qualité des données sont importantes.
- **Design** – gestion du couplage, incertitude inhérente, et l'intégration du ML à la logique du système.
- **Testing** – test du modèle, validation pour la production (accuracy, bias, robustness).
- **Operational** – Le maintien du ML est basé sur le recyclage et le redéploiement.

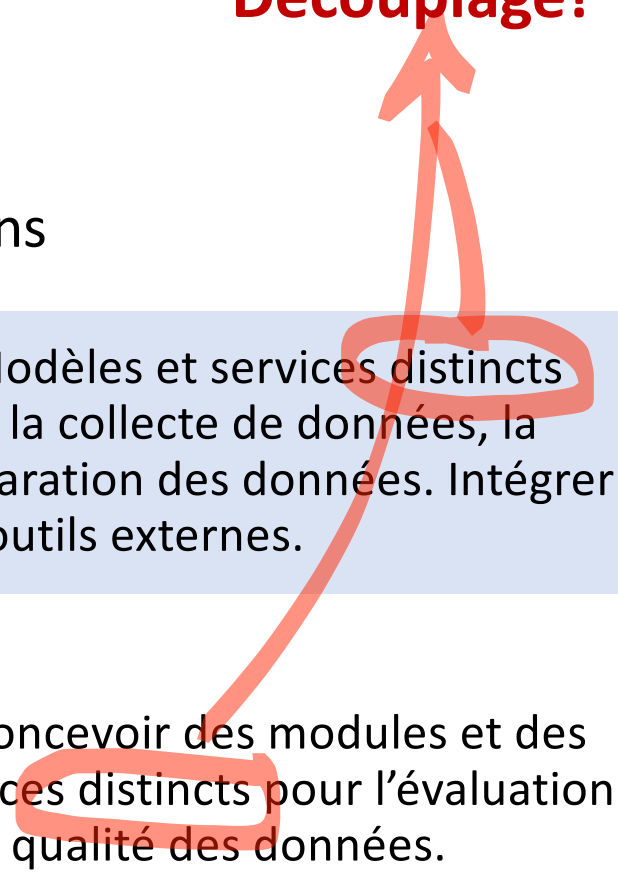
Exigences

• Problèmes	• Solutions
• #2 Les composants ML n'ont pas d'exigences fonctionnelles	• #2 Utilisez les mesures comme exigences fonctionnelles. • Inclure la compréhensibilité + l'explicabilité.
• #3 les projets ML ont des restrictions réglementaires	• #3 Analyser les contraintes réglementaires dès le départ. • Code de conduite de l'IA.

Données

- Problèmes
  - #4 La préparation des données peut donner lieu à un **jungle of scrapes**, joins, sampling, etc.
  - #5 La qualité des données est difficile à tester
- Solutions
  - #4 Modèles et services distincts pour la collecte de données, la préparation des données. Intégrer des outils externes.
  - #5 Concevoir des modules et des services distincts pour l'évaluation de la qualité des données.

Découplage!



Conception

Plusieurs équipes indépendantes travaillant sur des programmes construisent à partir de la même spécification.

- Problèmes
  - #7 Distinguer les défaillances des composants ML
  - #9 Les composantes ML apportent de l'incertitude
  - #10 Les composants ML peuvent échouer silencieusement
- Solutions
  - #7 Séparez la logique métier des composants ML. Normaliser les interfaces et la communication.
  - #9/#10/#11 Utilisez n-versioning.
  - Surveiller les métriques.
  - Utiliser des modèles interprétables.
  - Concevoir des modules de journal pour visualiser les métriques.

Testing

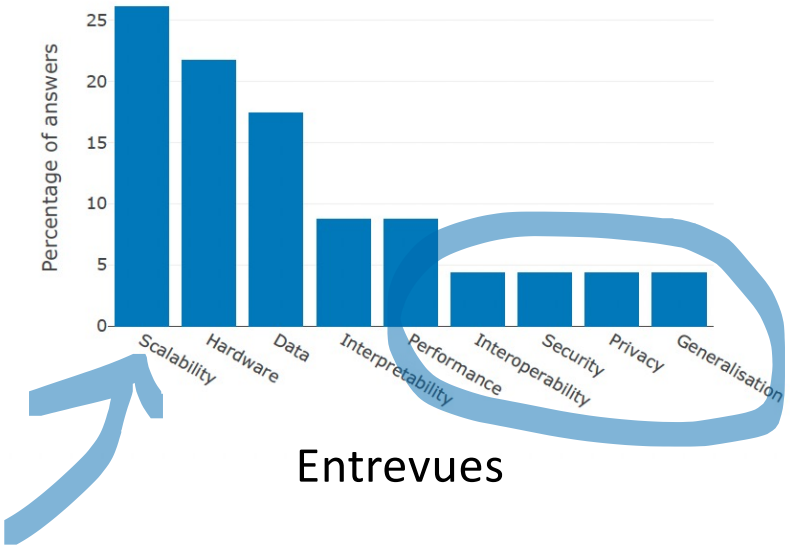
- Problèmes
  - #14 Les tests ml vont au-delà de la programmation.
  - Les problèmes découlent des données, du modèle et de l'incertitude.
- Solutions
  - #14 Concevoir des modèles et des tests de données.
  - Utiliser CI/CD.
  - Créer des tests d'intégrations et unitaires.

Opérations

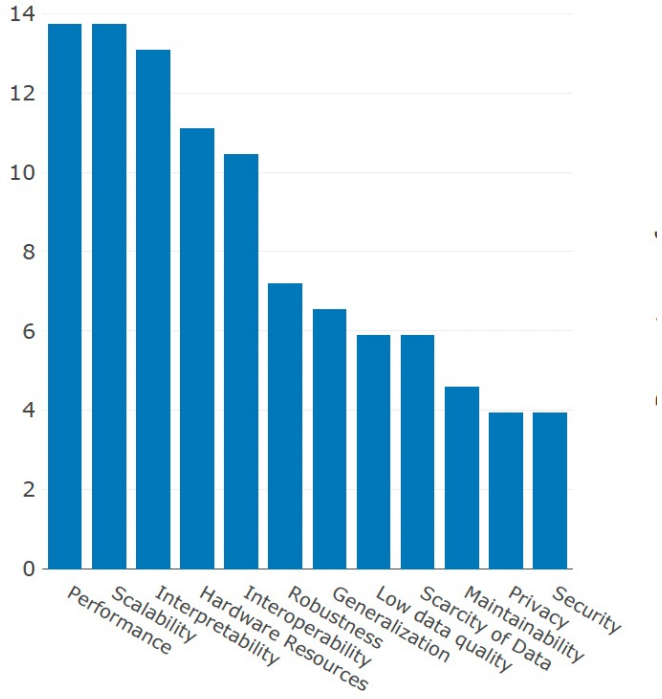
• Problèmes	• Solutions
• #14 Les composants ML nécessitent de maintenance continue, un recyclage, une évolution.	• #14 Conception pour le recyclage continu automatique. • Utilisez CI/CD. Restauration automatique.
• #19 Retracer les décisions aux modèles, aux données. • Reproduire les résultats antérieurs.	• #19 Conception pour la traçabilité et la reproductibilité. • Enregistrement des artefacts, des configurations de version, des modèles et des données.

Qu’est-ce qui motive la décision architecturale?

Figure 7: Distribution of architectural decision drivers from interviews.



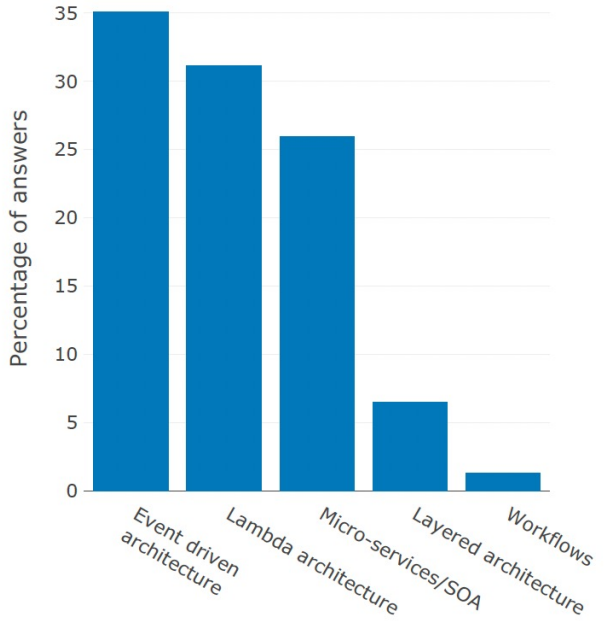
Entrevues



(b) SA decision drivers.

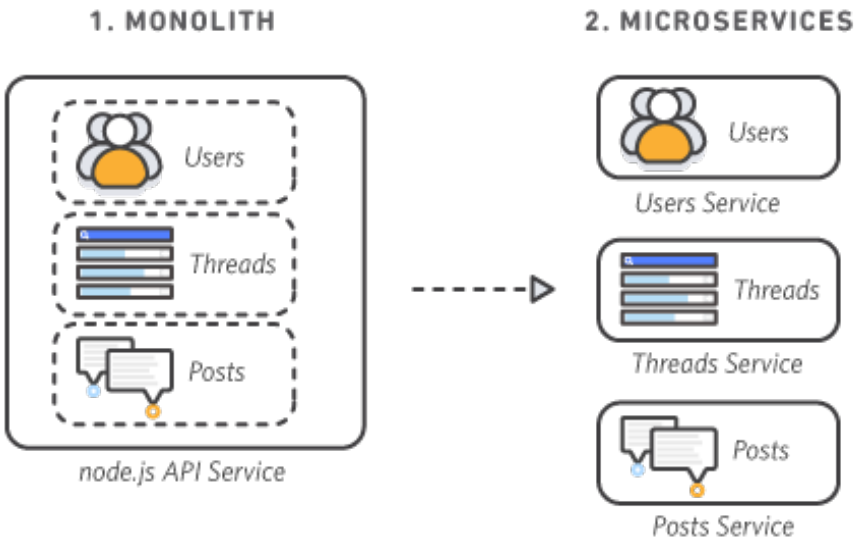
Enquête (survey)

Quels sont les styles architecturaux les plus courants?



(c) Architectural styles.

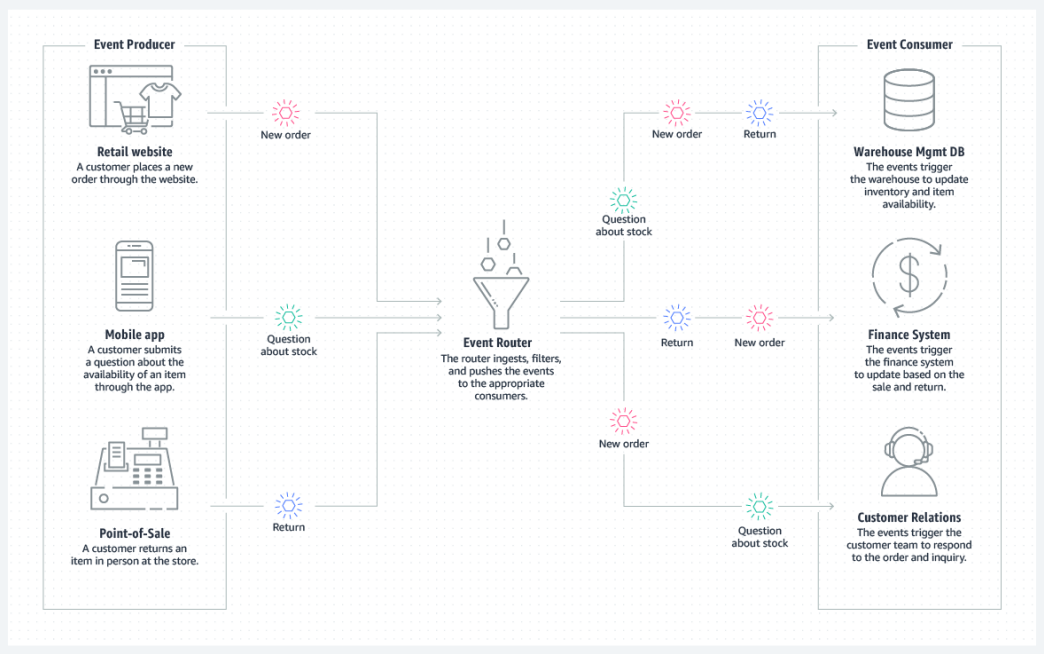
Microservices



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

Models can be deployed to respond to (micro)services.

Architecture pilotée par les événements (Event-driven Architecture)

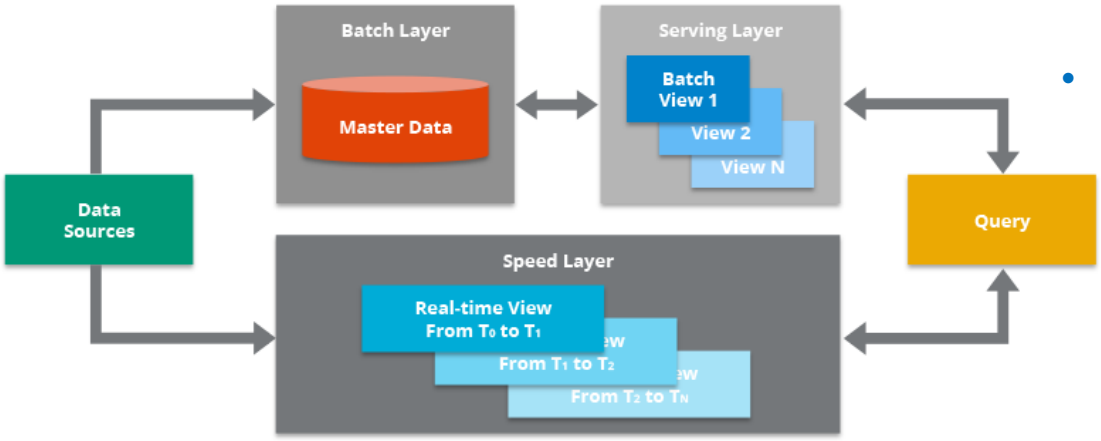


Chaining services/microservices based on events.

- The same model can service different events

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

Lambda Architecture



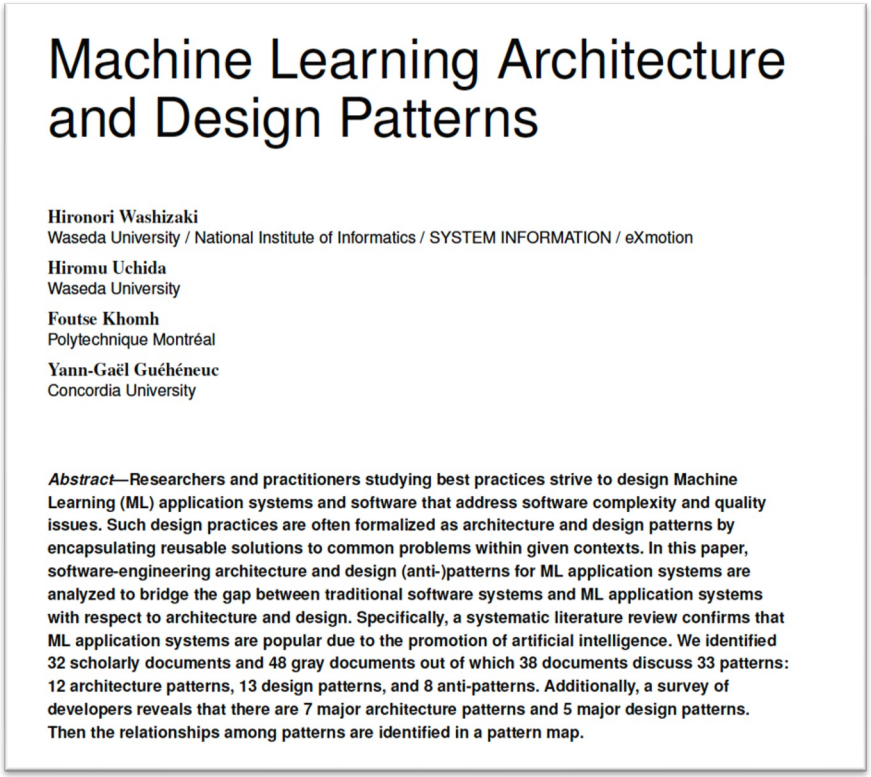
https://hazelcast.com/glossary/lambda-architecture/

Architecture pour le traitement des données en temps réel:

- Batch processing
  - le recyclage de modèles
- Stream processing (temps réel)
  - Prédictions en temps reel
- Servicing layer orchestre le système

Machine Learning Architecture and Design Patterns

Objectif: Trouver des solutions réutilisables à des problèmes courants d'architecture.



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.**

Conception de l'étude

- Deux auteurs ont lu la moitié des documents et extrait des **motifs**
- Les autres auteurs ont vérifié les **motifs**
- Identifié 69 **motifs** et 33 sont restés après vérification
  - **ML architecture patterns**: résoudre les problèmes architecturaux récurrents
  - **ML design patterns**: résoudre les problèmes récurrents de conception

Styles architecturaux

Data Lake

- Problem: Nous ne pouvons pas **prévoir** le type d'analyse qui sera effectuée sur les données.

- Solution:
  - Gardez les données aussi brutes (raw) que possible.
  - Centralisé.
  - Permettre une analyse parallèle.

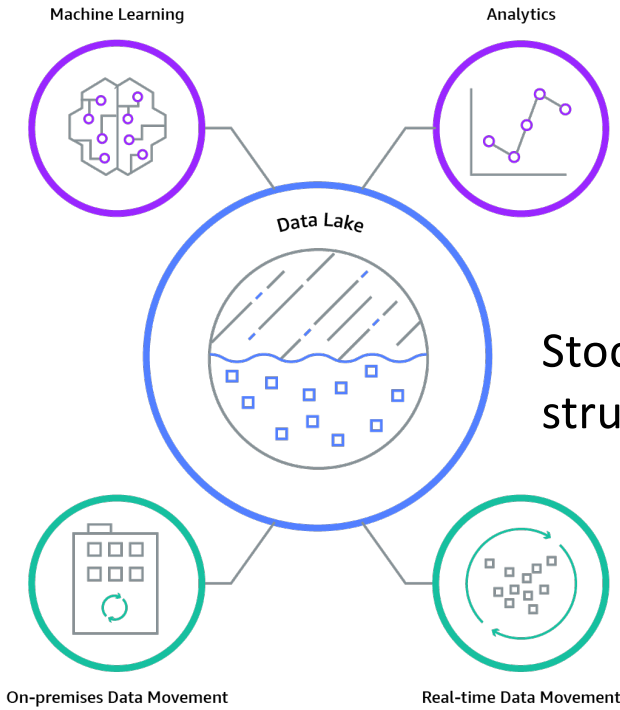
Microservices,

Lambda,

Event-driven

(...)

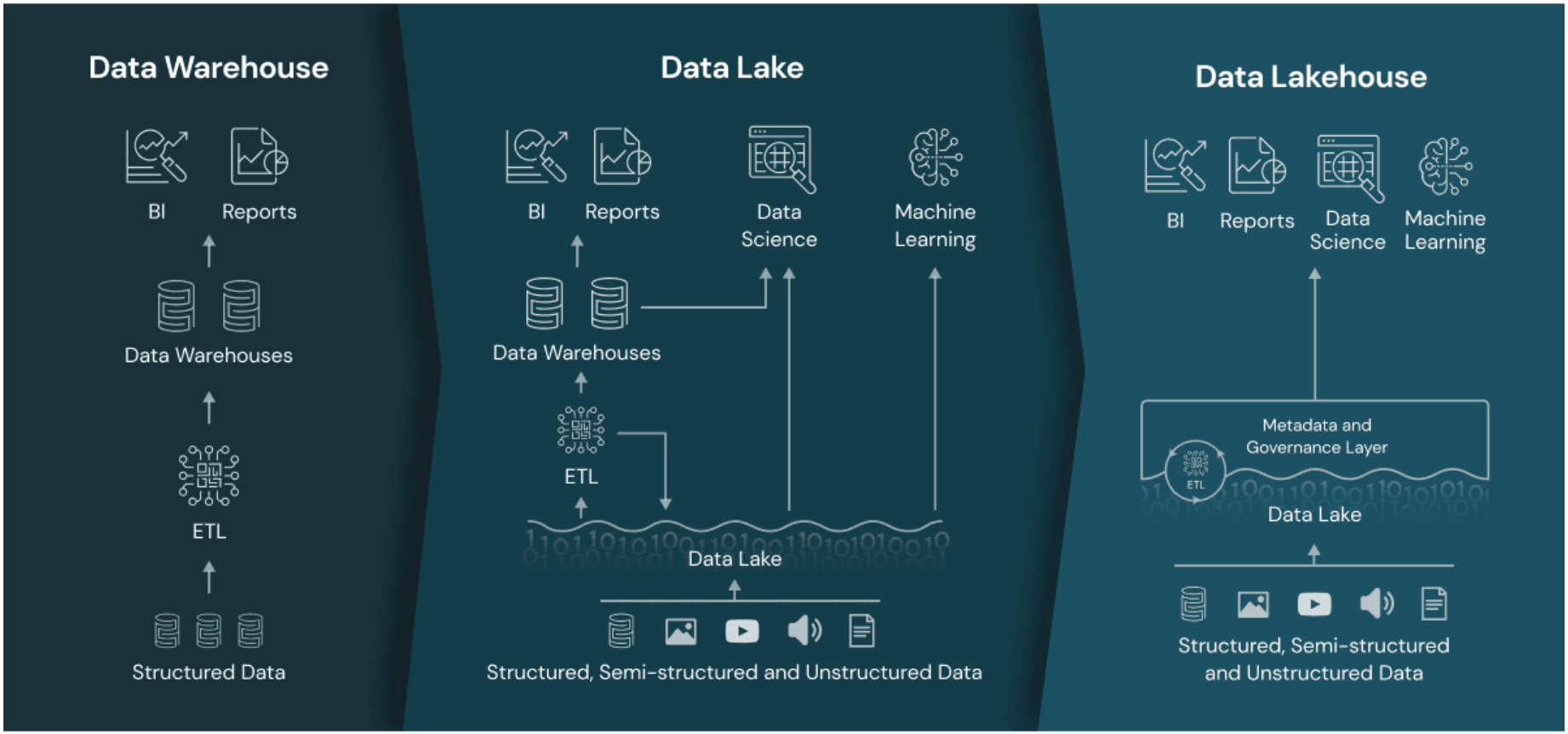
Data Lake



Stocke les données brutes structurées et non structurées

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

Data Lakehouse



https://www.databricks.com/fr/glossary/data-lakehouse

Design Patterns

ML Versioning

- Différentes versions de modèles peuvent modifier le comportement de l'application

- Enregistrer les données + modèle + configuration pour assurer un processus reproductible.

Independent Testing Infra

- Difficile de trouver des erreurs lorsque l'infrastructure et le ML sont mélangés

- Assurez-vous que le système est testable.
- Les modèles doivent être encapsulés.

Antipatterns

- Big-Ass Script Architecture
  - Tout le code est placé dans un grand script
  - Composants difficiles à réutiliser
- Glue code
  - Gèle un système aux particularités d'un paquet, bibliothèque
- Pipeline jungles
  - La collecte de données, la préparation est effectuée sur une séquence d'opérations de pipeline de données couplées.

Distinguer la logique métier du modèle ML

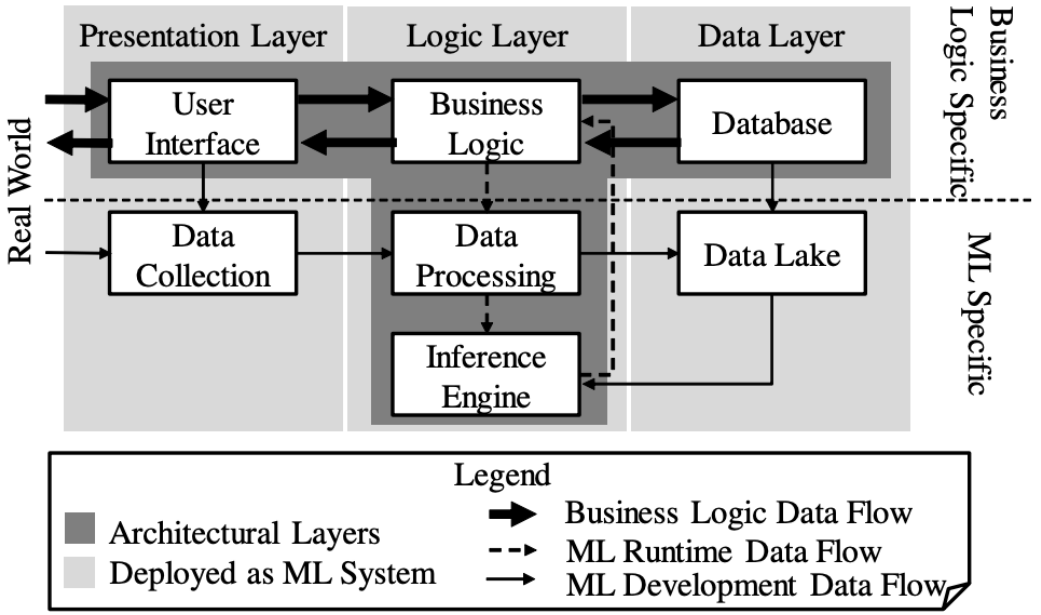


Figure 2. Structure of Distinguish Business Logic from ML Model pattern [3]

Question de l'examen final

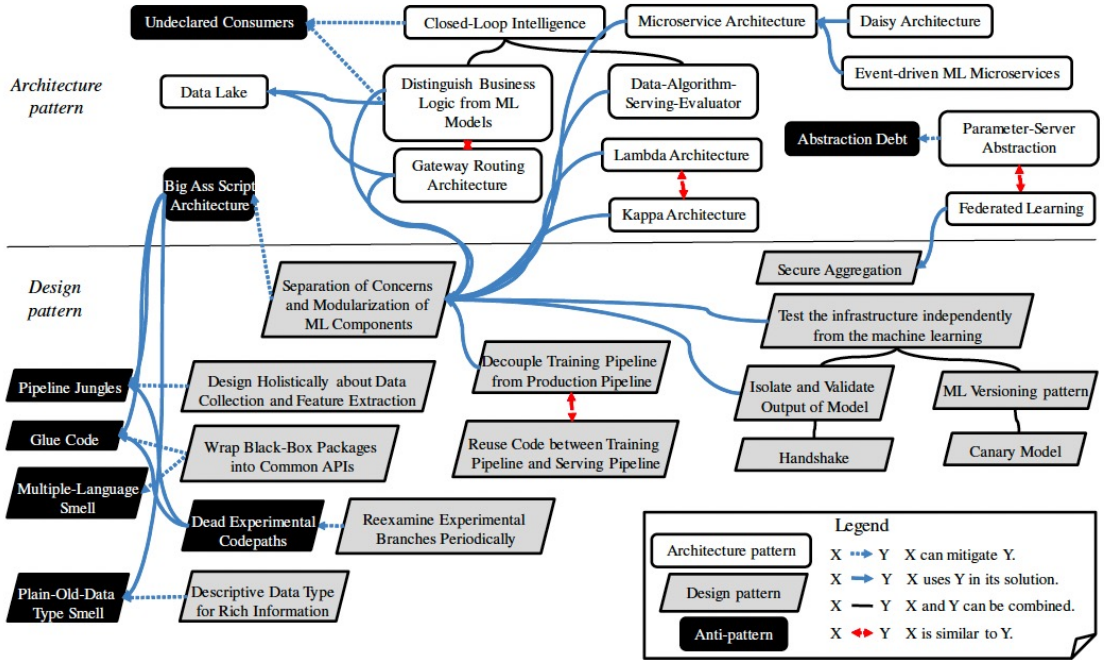


Figure 4. Pattern Map showing classifications and relationships among ML patterns

Références

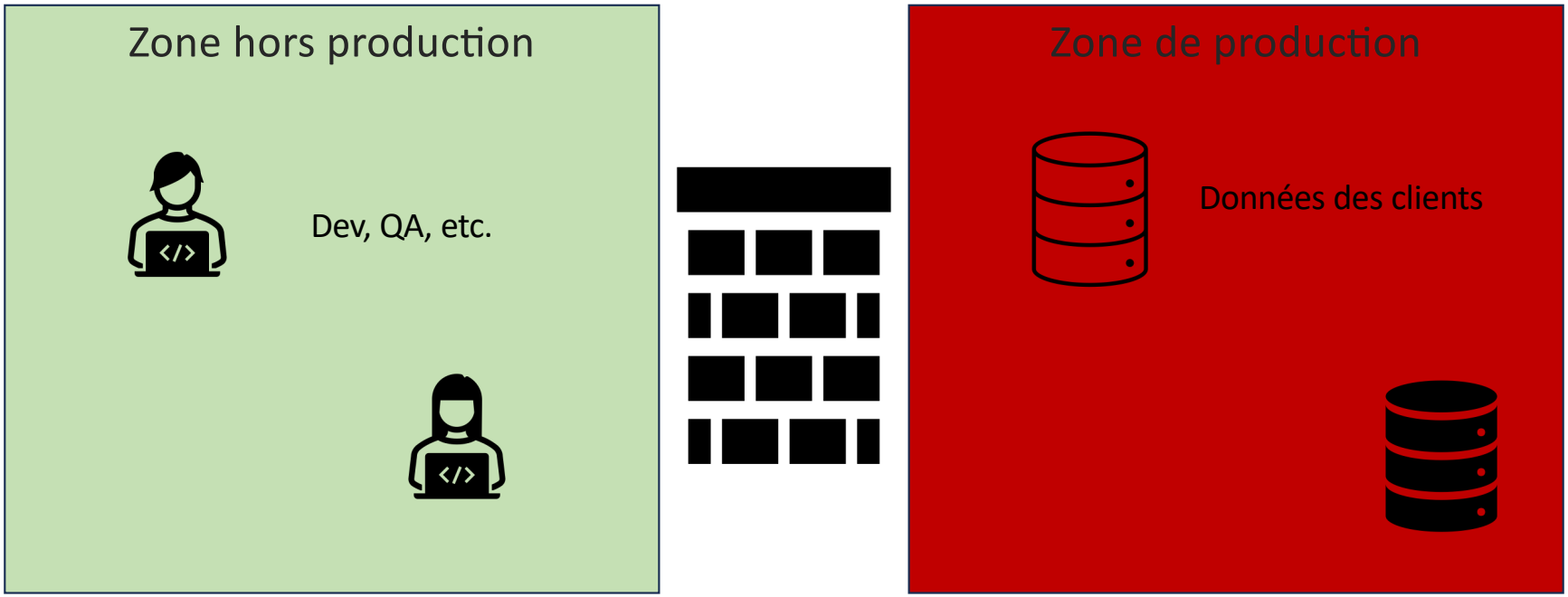
- Christian Kaestner: Machine Learning in Production, Carnegie Mellon University, 2022

- Sharir, O., Peleg, B., & Shoham, Y. (2020). The Cost of Training NLP Models: A Concise Overview. *ArXiv*.

- Geoff Hulten (2018). Building Intelligent Systems: A Guide to Machine Learning Engineering

Prod vs Non Prod =
le principal obstacle des architectures IA

Architecture TI traditionnelle



Architecture IA