

# **Test des systèmes d'intelligence artificielle**

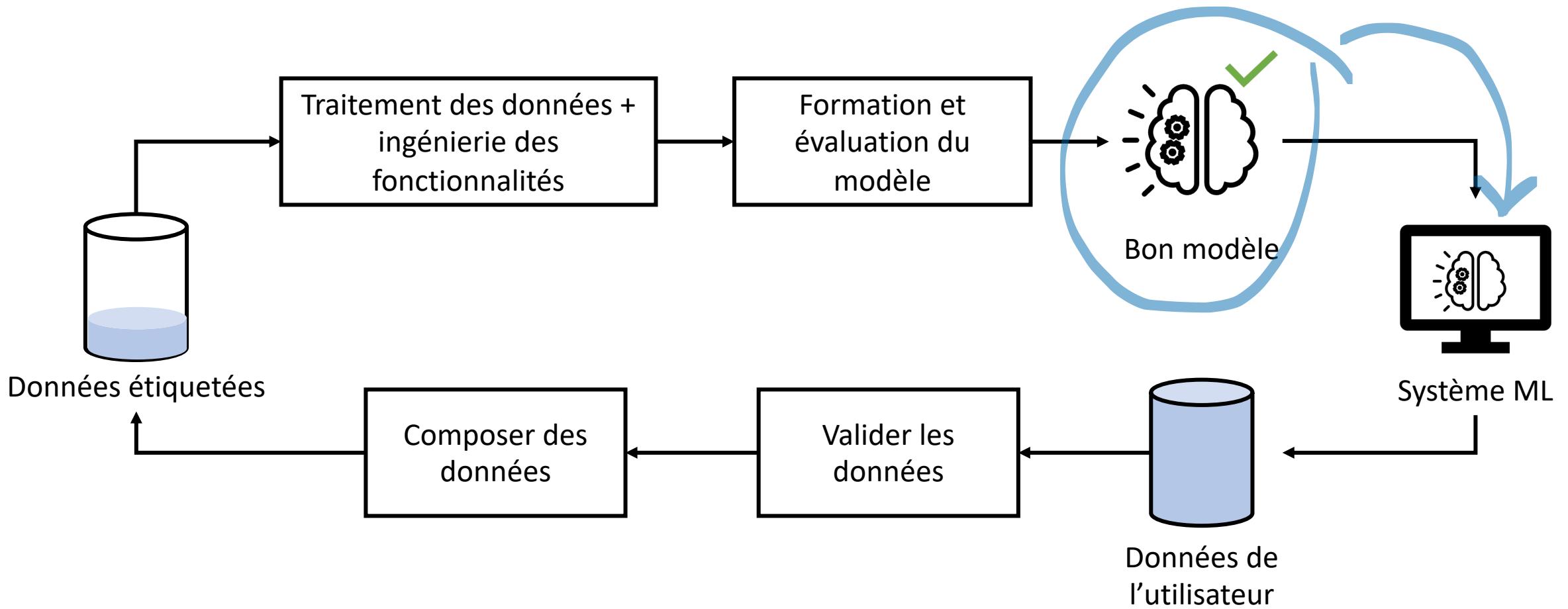
MGL 7811: Ingénierie logicielle des systèmes d'intelligence artificielle



Diego Elias Costa, PhD

Université du Québec à Montréal

# Pipeline de données des systèmes ML



# Les Notebooks ne sont que le début

Sélection des données  
Nettoyage des données  
Ingénierie des fonctionnalités  
Sélection du modèle  
...

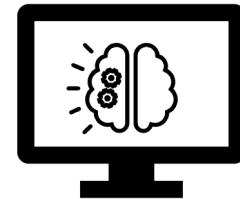
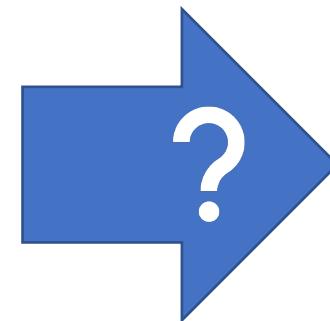
In [25]:

```
# Test la qualité de la prédiction du modèle

classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# Montre le résultat
print(f"Accuracy of our model's prediction {accuracy_score(y_test,y_pred)}")
```

Accuracy of our model's prediction 0.736



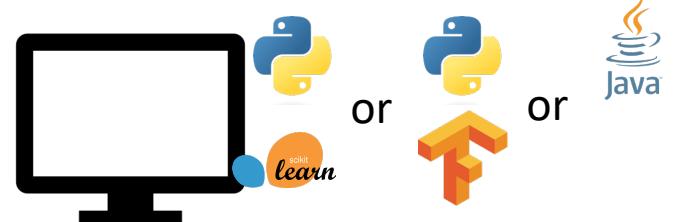
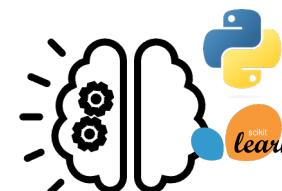
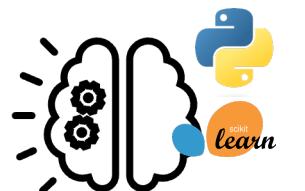
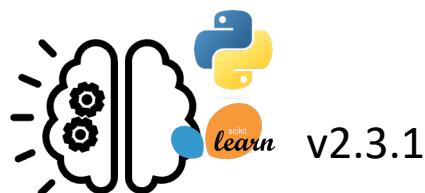
Système ML

# Persistance du modèle

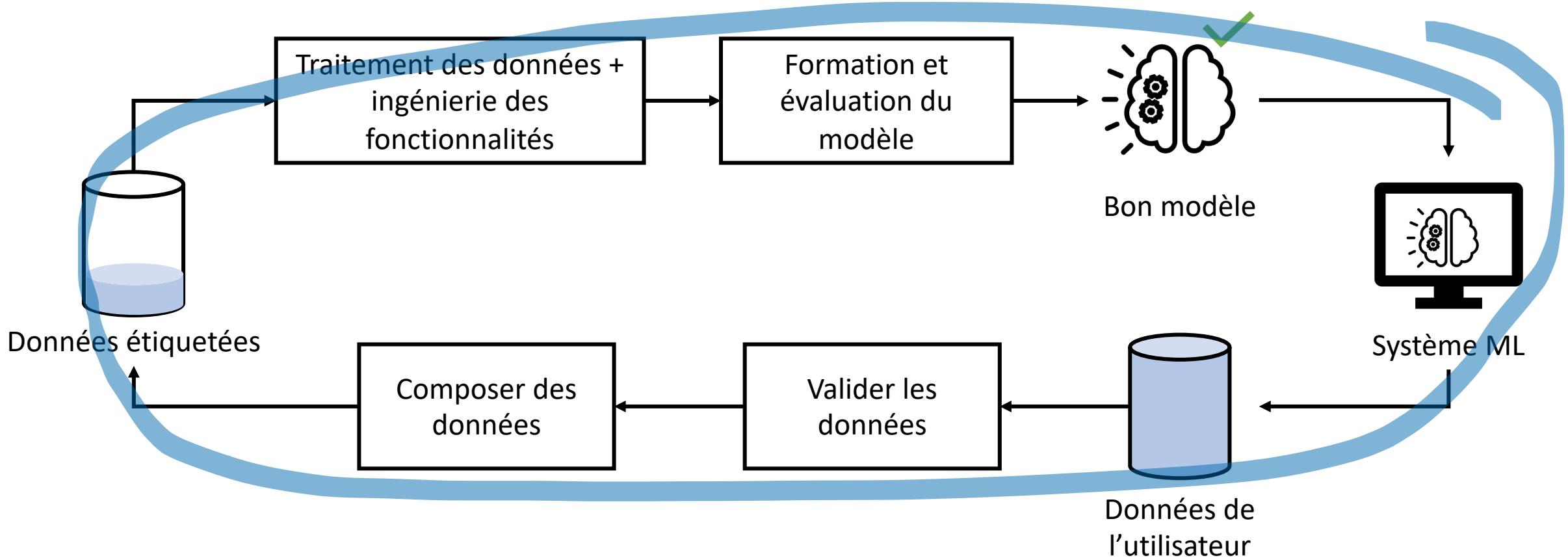
- Préparez votre modèle pour qu'il soit servi en production
  - Sérialiser le modèle entraîné
  - Incorporer le modèle dans le système
- Méthodes scikitlearn pour [model persistence](#)
  - Objets Python
  - Format sécurisé
  - Formats interopérables
    - Open Neural Network Exchange ([ONNX](#))
    - PMML

# Considérations sur les stratégies de persistance

- Quel stratégie utiliser ?



# Pipeline de données des systèmes ML



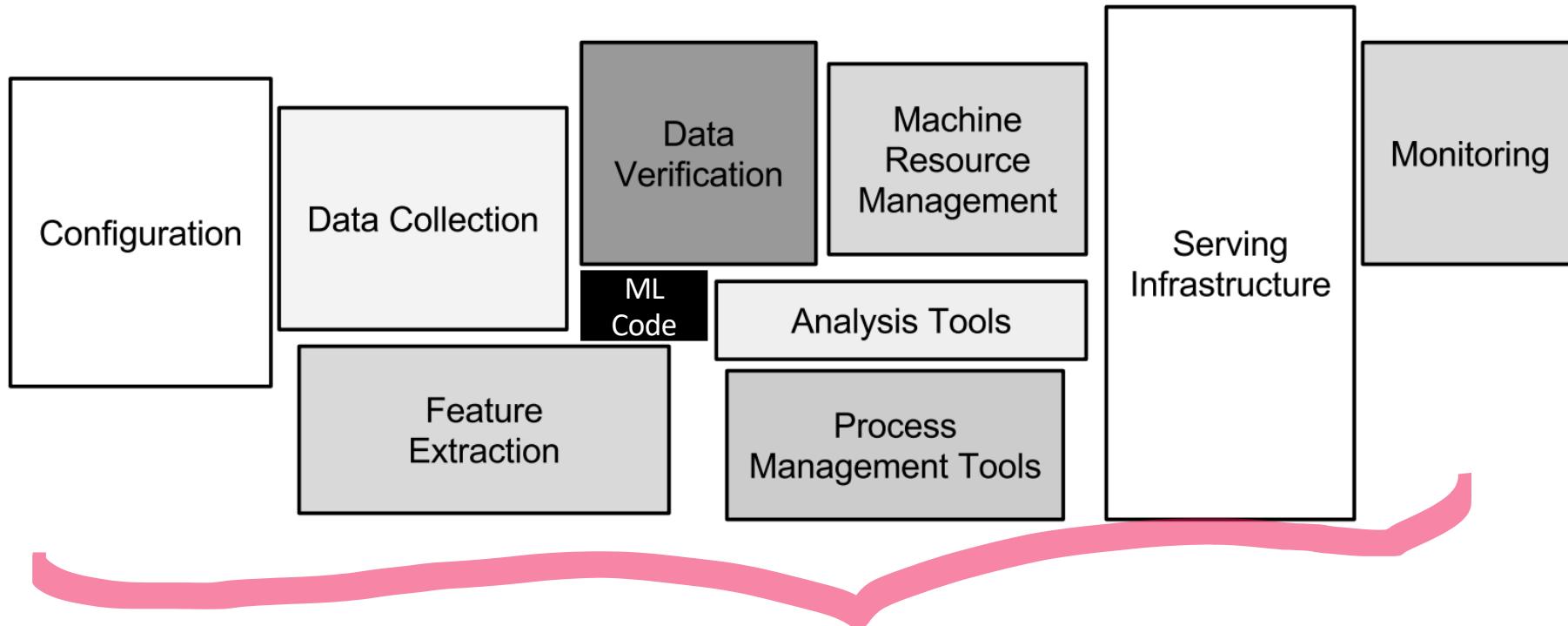
# Qu'est-ce que MLOps?

# ML Illusion de code



ML  
Code

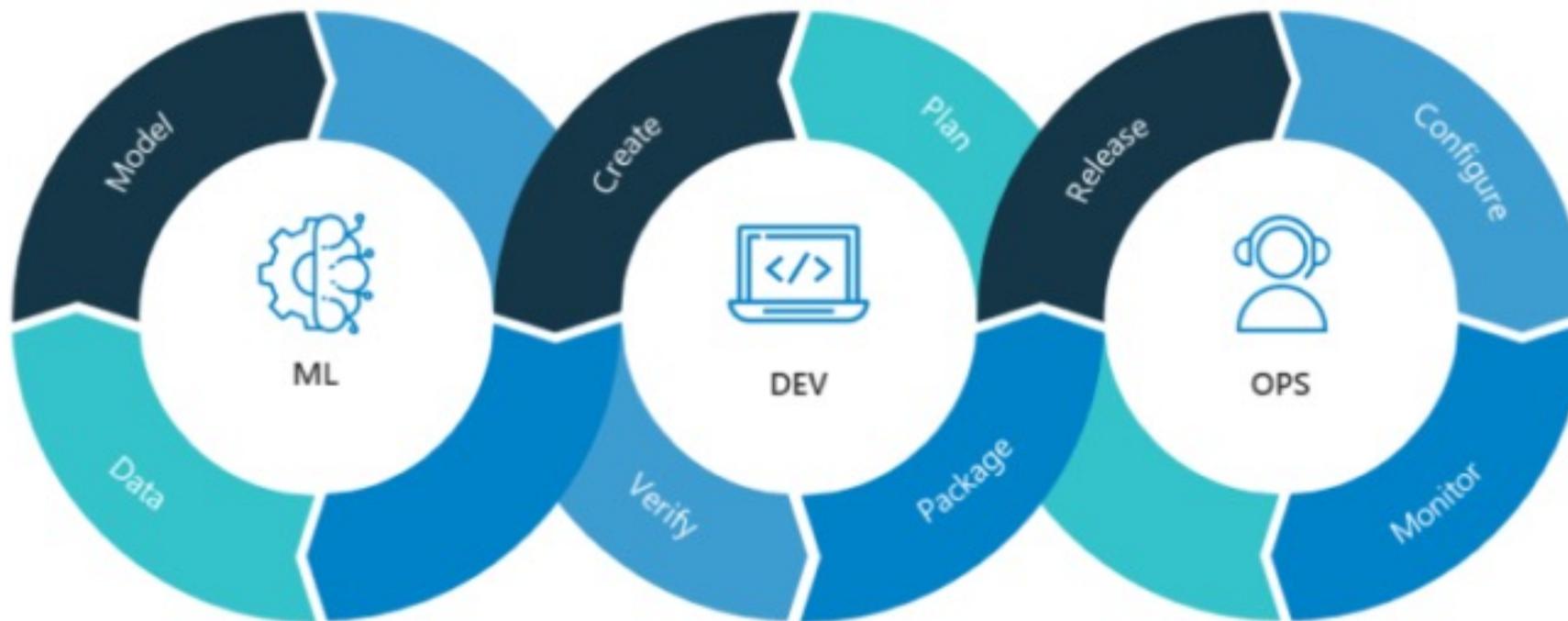
# Le code ML n'est qu'un petit composant d'un grand système



Des problèmes peuvent survenir dans toute l'infrastructure ML!

Testez l'ensemble de l'infrastructure ML

# Machine Learning Operations



<https://blogs.nvidia.com/blog/2020/09/03/what-is-mlops/>

# Problèmes et solutions

## ML Hidden Debts

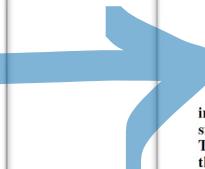
### Hidden Technical Debt in Machine Learning Systems

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips  
{dsculley, gholt, dg, edavydov, toddphillips}@google.com  
Google, Inc.

Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, Dan Dennison  
{ebner, vchaudhary, mwyong, jfcrespo, dennison}@google.com  
Google, Inc.

#### Abstract

Machine learning offers a fantastically powerful toolkit for building useful complex prediction systems quickly. This paper argues it is dangerous to think of these quick wins as coming for free. Using the software engineering framework of *technical debt*, we find it is common to incur massive ongoing maintenance costs in real-world ML systems. We explore several ML-specific risk factors to account for in system design. These include boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, configuration issues, changes in the external world, and a variety of system-level anti-patterns.



## ML Tests

### The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction

Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley  
Google, Inc.  
ebreck, cais, nielsene, msalib, dsculley@google.com

*Abstract*—Creating reliable, production-level machine learning systems brings on a host of concerns not found in small toy examples or even large offline research experiments. Testing and monitoring are key considerations for ensuring the production-readiness of an ML system, and for reducing technical debt of ML systems. But it can be difficult to formulate specific tests, given that the actual prediction behavior of any given model is difficult to specify *a priori*. In this paper, we present 28 specific tests and monitoring needs, drawn from experience with a wide range of production ML systems to help quantify these issues and present an easy to follow road-map to improve production readiness and pay down ML technical debt.

may find difficult. Note that this rubric focuses on issues specific to ML systems, and so does not include generic software engineering best practices such as ensuring good unit test coverage and a well-defined binary release process. Such strategies remain necessary as well. We do call out a few specific areas for unit or integration tests that have unique ML-related behavior.

*How to read the tests:* Each test is written as an assertion; our recommendation is to test that the assertion is true, the more frequently the better, and to fix the system if the assertion is not true.

# Technical Debt in ML Systems

---

- Rapports sur les problèmes liés à la maintenance des systèmes de ML

## Hidden Technical Debt in Machine Learning Systems

---

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips  
{dsculley, gholt, dg, edavydov, toddphillips}@google.com  
Google, Inc.

Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, Dan Dennison  
{ebner, vchaudhary, mwy, jfcrespo, dennison}@google.com  
Google, Inc.

### Abstract

Machine learning offers a fantastically powerful toolkit for building useful complex prediction systems quickly. This paper argues it is dangerous to think of these quick wins as coming for free. Using the software engineering framework of *technical debt*, we find it is common to incur massive ongoing maintenance costs in real-world ML systems. We explore several ML-specific risk factors to account for in system design. These include boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, configuration issues, changes in the external world, and a variety of system-level anti-patterns.

# Study Design

- Rapport industriel
  - Expérience de l'industrie
  - 10 auteurs de Google
- Publié dans NeurIPS en 2015
  - Neural Information Processing Systems
- **Objectif:**
  - Ne pas proposer de nouvel algorithme ou de nouvelle solution
  - **Accroître la sensibilisation aux problèmes**

# La maintenance des systèmes de ML est difficile

“Le développement et le déploiement de systèmes d'apprentissage automatique (ML) sont **relativement rapides** et **peu coûteux**.

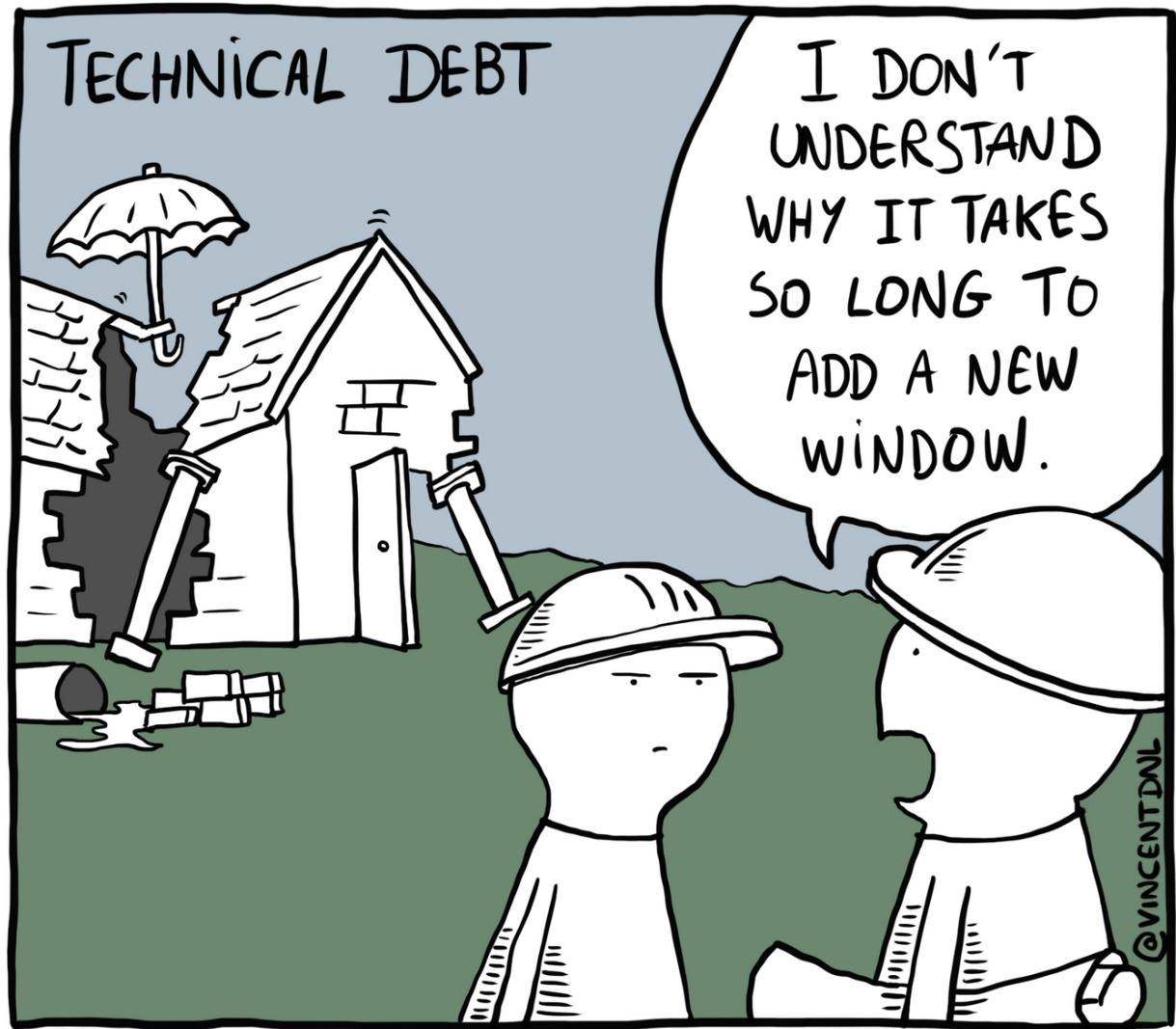
Cependant, la maintenance de ces systèmes sur le long terme est **difficile et coûteuse**.”

# Qu'est-ce que la dette technique (Technical Debt)?



# Dette technique

- Coûts **à long terme d'agir rapidement** en génie logiciel.
- Toutes les dettes ne sont pas mauvaises, mais elles doivent être payées
  - Refactorisation du code
  - Réduction des dépendances
  - Améliorer les tests



# **La dette technique (DT) est plus difficile dans les systèmes d'apprentissage automatique**

- Systèmes d'apprentissage automatique
  - Tous les DTs de logiciels traditionnels PLUS
  - Les DTs de l'apprentissage automatique
- Des dettes techniques peuvent exister au niveau du système
  - Plus difficile que les DTs au niveau du code

# Structure du papier

- Les modèles complexes érodent les frontières.
- Dépendances de données
- Boucles de retroaction (feedback loops)
- Antimodèles de système ML
- Dette de configuration
- Changements dans le monde externe
- Autres domaines de dette technique.



Technical  
Problems!

# Érosion des limites

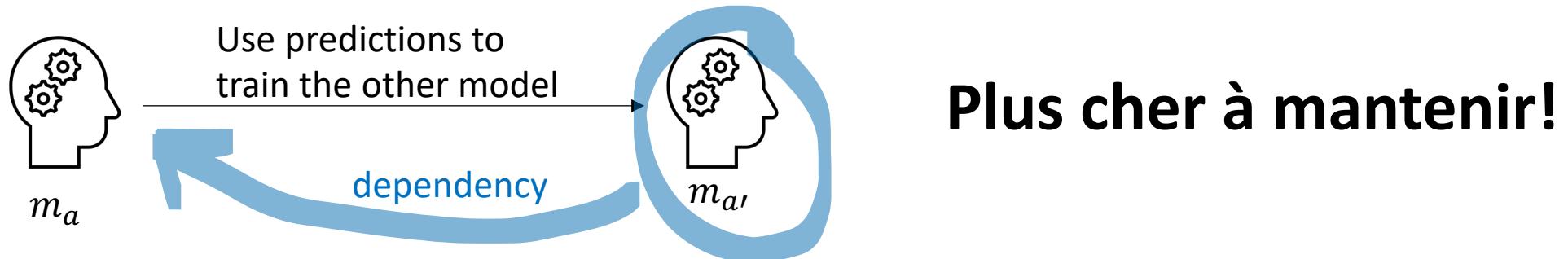
- Logiciel traditionnel
  - Les limites **d'encapsulation** et **d'abstraction** aident à créer un code maintenable.
- Il est difficile d'imposer des limites abstraites strictes dans les systèmes d'apprentissage automatique (ML).
  - Le comportement des systèmes d'apprentissage automatique dépend des données externes.
  - Les données du monde réel ne s'adaptent pas facilement aux limites d'encapsulation précises.

# Enchevêtrement (Entanglement)

- **CACE**
  - Changes Anything, Changes Everything
- L'ajout ou la suppression de fonctionnalités modifie la façon dont le modèle dépendait précédemment d'autres fonctionnalités.
- Mesures d'atténuation:
  - Les ensembles de serveurs essayent de capturer différents aspects du problème.
  - Déetecter les changements de comportement dès qu'ils se produisent.

# Correction Cascades

- Vous avez un modèle  $m_a$  pour un problème
- Vous devez résoudre un problème légèrement différent A'

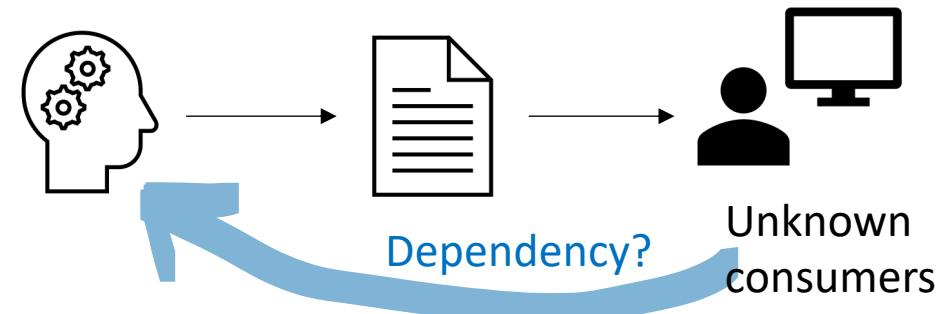


## Mesures d'atténuation:

- Augmentez  $m_a$  avec de nouvelles fonctionnalités pour résoudre A'
- Accepter les coûts de création d'un modèle complètement nouveau

# Undeclared Consumers

- Les prédictions des modèles d'apprentissage automatique sont rendues accessibles en interne
  - CSV files, log files



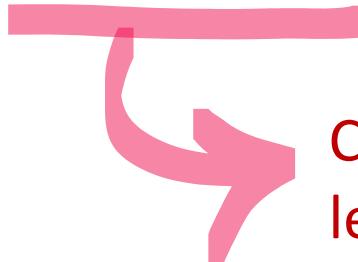
- Crée un couplage **invisible** entre les systèmes
- Mesures d'atténuation:
  - Protéger l'accès avec des accords de niveau de service stricts (SLA)

# Les dépendances de données **coûtent plus cher** que les dépendances de code

- SE traditionnel: La dépendance des données est un contributeur clé à la complexité du code
  - Les outils peuvent aider les développeurs
- Dans les systèmes de ML, la dépendance des données peut être tout aussi problématique mais **plus difficile à identifier**
  - Pas de bon outillage pour la dépendance de données ML

# Dépendances de données instables

- Il est courant d'utiliser des données d'autres systèmes en tant que fonctionnalité en entrée
  - Que se passe-t-il si les données sont instables ?
  - Changer de comportement au fil du temps
- **Stratégie d'atténuation:**
  - Créer une copie versionnée des signaux d'entrée



Comment conservez-vous  
les copies versionnées?

# Dépendances de données sous-utilisées

- Des données sous-utilisées peuvent **rendre le système vulnérable** aux changements sans offrir beaucoup d'avantages.
- **Legacy features:** Utile à un moment donné, mais plus maintenant
- **Bundled features:** Caractéristiques regroupées, mais n'ont pas été évaluées individuellement
- **$\epsilon$ -Features:** amélioration négligeable
- Mesures d'atténuation:
  - Feature importance, leave-one-feature-out evaluation

# Anti-patrons des systèmes de ML

- **Glue Code**
  - Contraint par les bibliothèques
- **Pipeline Jungle**
  - Jungle d'éraflures, jointures, échantillonnage...
- **Dead Experimental Codepaths**
  - Le coût de l'expérimentation est faible
  - Il est difficile de maintenir toutes les voies d'expérimentation
- **Abstraction Debt**
  - Quelle est la bonne abstraction pour décrire les données? Modèle? Prédiction?

# Odeurs courantes dans les systèmes ML

- **Plan-Old-Data**
  - Utilisation d'entités brutes sans codage ni métadonnées
- **Multiple languages**
  - L'utilisation de plusieurs langues augmente souvent le coût des tests et du transfert de propriété
  -
- **Prototype**
  - L'évolution du prototype vers une solution est coûteuse
  - La pression du temps pourrait pousser le prototype à être la solution

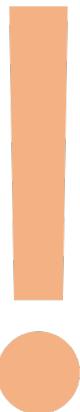
# Faire face aux changements dans le monde extérieur

- **Seuils fixes** dans les systèmes dynamiques
- Surveillance et Test
  - Les tests unitaires sont utiles, mais pas suffisants
  - Les données réelles changent et doivent être surveillées
  - **Prediction bias**
    - Distribution des étiquettes prévues equiv. étiquettes observes
  - **Action limits**
    - Appliquer des limites au système ML
  - **Up-stream producers**
    - Les producteurs de données devraient être constamment surveillés

# Conclusion: Measuring Debt and Paying it Off

- À quel point peut-on **facilement tester une nouvelle approche algorithmique à grande échelle** ?
- Dans quelle mesure peut-on **mesurer précisément l'impact d'un changement** apporté au système ?
- À quelle vitesse les **nouveaux membres** de l'équipe peuvent-ils être mis au courant ?

# Open Discussion



# The ML Test Score

- Un cadre pour la production de ML et la réduction de la dette technique

## **The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction**

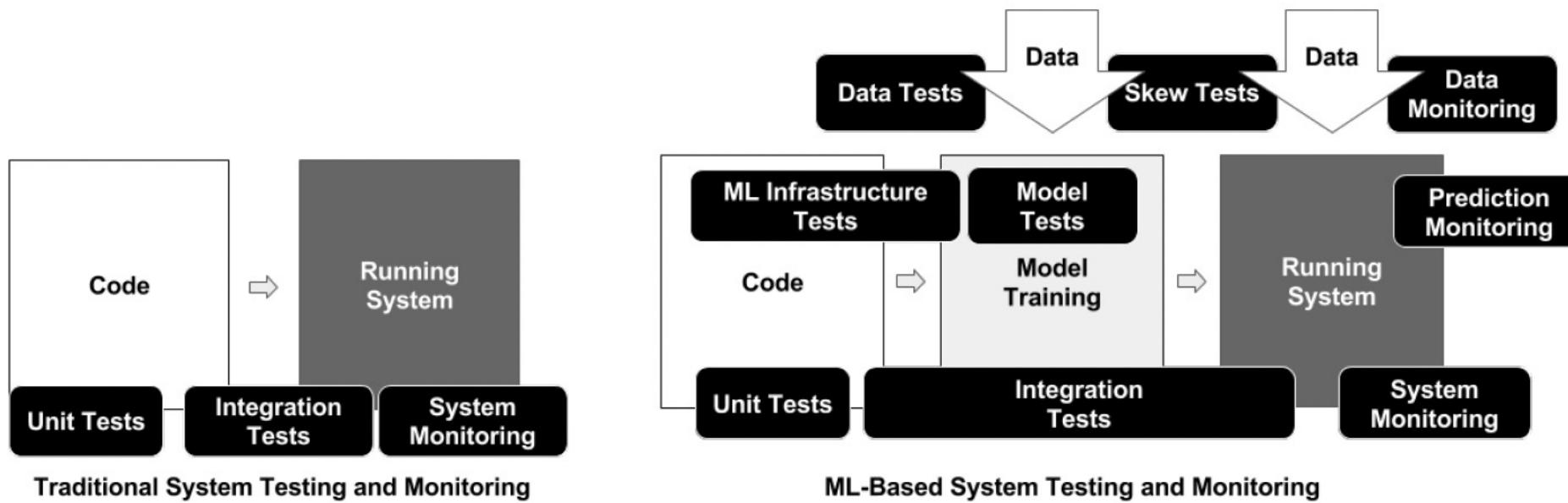
Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, D. Sculley  
Google, Inc.  
ebreck, cais, nielsene, msalib, dsculley@google.com

*Abstract*—Creating reliable, production-level machine learning systems brings on a host of concerns not found in small toy examples or even large offline research experiments. Testing and monitoring are key considerations for ensuring the production-readiness of an ML system, and for reducing technical debt of ML systems. But it can be difficult to formulate specific tests, given that the actual prediction behavior of any given model is difficult to specify *a priori*. In this paper, we present 28 specific tests and monitoring needs, drawn from experience with a wide range of production ML systems to help quantify these issues and present an easy to follow road-map to improve production readiness and pay down ML technical debt.

may find difficult. Note that this rubric focuses on issues specific to ML systems, and so does not include generic software engineering best practices such as ensuring good unit test coverage and a well-defined binary release process. Such strategies remain necessary as well. We do call out a few specific areas for unit or integration tests that have unique ML-related behavior.

*How to read the tests:* Each test is written as an assertion; our recommendation is to test that the assertion is true, the more frequently the better, and to fix the system if the assertion is not true.

# La motivation



Qu'est-ce qui devrait être testé?  
Combien est suffisant?

# Study Design

- Rapport industriel
  - Google
- Publié sur IEEE Big Data en 2017
- Goal:
  - Rubrique de 28 tests exploitables
  - Système de notation pour mesurer la préparation d'un système ML à la production

# Related work

Software testing is well studied, as is machine learning, but their intersection has been less well explored in the literature. [4] reviews testing for scientific software more generally, and cites a number of articles such as [5], who present an approach for testing ML algorithms. These ideas are a useful complement for the tests we present, which are focused on testing the use of ML in a production system rather than just the correctness of the ML algorithm per se.

Zinkevich provides extensive advice on building effective machine learning models in real world systems [6]. Those rules are complementary to this rubric, which is more concerned with determining how reliable an ML system is rather than how to build one.

Issues of surprising sources of technical debt in ML systems has been studied before [1]. It has been noted that the prior work has identified problems but been largely silent on how to address them; this paper details actionable advice drawn from practice and verified with extensive interviews with the maintainers of 36 real world systems.

Rules of machine learning – Talk in 1996

Travail connexe assez court

Machine learning: The high interest credit card of technical debt (2014)

# Catégories de tests

- Tests pour les fonctionnalités et les données
- Tests pour le développement des modèles
- Tests pour l'infrastructure ML
- Tests de surveillance pour les systèmes ML

# Tests for Feature and Data

1	Feature expectations are captured in a schema.
2	All features are beneficial.
3	No feature's cost is too much.
4	Features adhere to meta-level requirements.
5	The data pipeline has appropriate privacy controls.
6	New features can be added quickly.
7	All input feature code is tested.

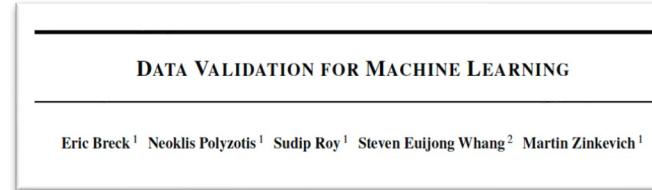


Table I  
BRIEF LISTING OF THE SEVEN DATA TESTS.

# #2 Toutes les fonctionnalités sont-ils bénéfiques?

- Chaque fonctionnalité a un coût d'ingénierie logicielle
  - Validez la valeur que chaque fonctionnalité fournit en puissance prédictive
- Comment?
  - Coefficients de corrélation
  - Entraînement des modèles avec des fonctionnalités individuelles
  - Entraînement des modèles avec k caractéristiques supprimées

# #4 Les fonctionnalités sont conformes aux exigences de méta-niveau

- Les projets peuvent imposer des **exigences** sur les données entrant dans le système
  - Caractéristiques protégées
  - Toutes les données doivent provenir d'une seule source
- Comment?
  - **Faire respecter ces exigences**
    - Tests de pipeline
    - Tests de modèle

# Tester le développement du modèle

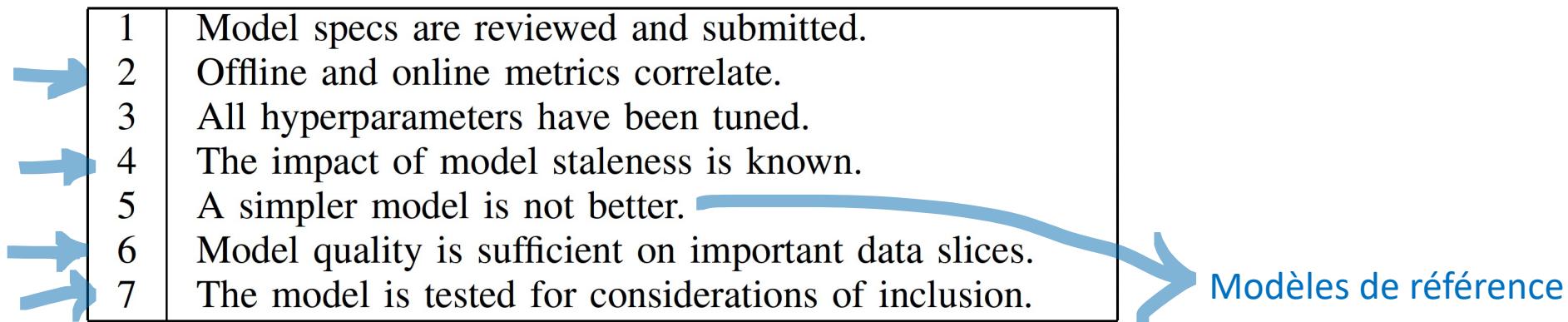


Table II  
BRIEF LISTING OF THE SEVEN MODEL TESTS

# #2 Les mesures hors ligne et en ligne sont corrélées

- Métriques ML
  - Log-loss, accuracy, precision, AUC ROC
- Métriques utilisateur
  - L'engagement, bonheur de l'utilisateur, les revenus;
- Comment?
  - Effectuer des tests A/B pour évaluer l'impact de meilleurs modèles (ou intentionnellement pires)



Comment les mesures ML traduisent-elles en mesures centrées sur l'utilisateur ?

# #4 Impact de l'obsolescence

- Qu'est-ce que **l'obsolescence** du modèle?
  - Un modèle qui n'est pas suffisamment mis à jour
- Testez l'impact d'un modèle obsolet
  - Vous aide à déterminer la fréquence des mises à jour, les plans d'urgence
- Comment?
  - Effectuer des tests A/B avec des modèles dépassés par différentes plages de dates

# #6 La qualité du modèle est bonne pour toutes les tranches de données importantes

L'impact Mondial d'un changement n'est peut-être pas uniforme

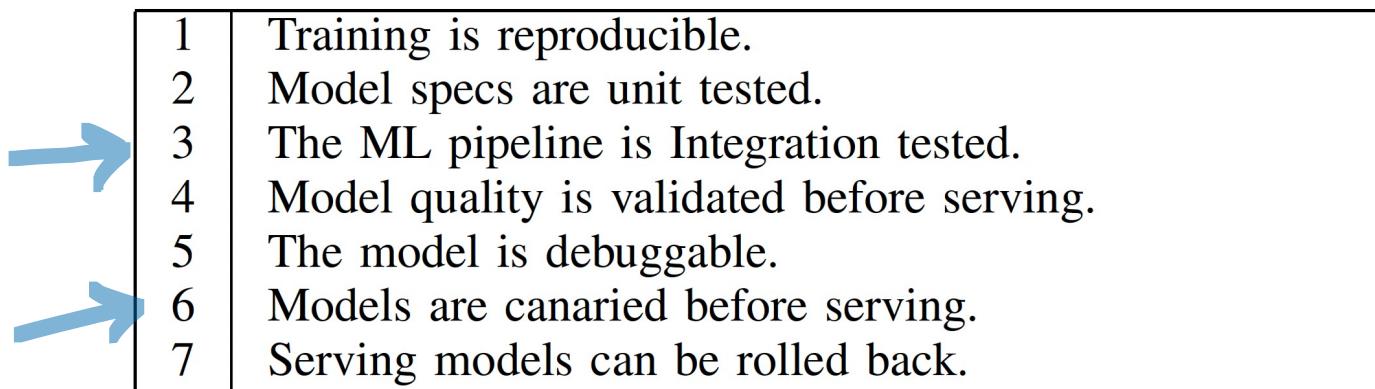
- Précision globale +1%, mais...
  - pour un pays -50%
  - pour les jeunes -30%
- Probablement causé par des flux de données défectueux...
- Comment?
  - Inclure des tests avec différentes tranches de données dans le processus de publication

# #7 Modèle testé pour les considérations d'inclusion

- ML Fairness
- Les biais négligés dans l'ensemble de données d'entraînement influencent le comportement du système
- Comment?
  - Tests avec des entités en entrée x caractéristiques protégées
  - Post-traitement pour minimiser les pertes disproportionnées pour certains groupes

# Tests for ML Infrastructure

Table III  
BRIEF LISTING OF THE ML INFRASTRUCTURE TESTS



1	Training is reproducible.
2	Model specs are unit tested.
3	The ML pipeline is Integration tested.
4	Model quality is validated before serving.
5	The model is debuggable.
6	Models are canaried before serving.
7	Serving models can be rolled back.

# #3 Le pipeline de ML est testé d'intégration

- Tests d'automatisation qui s'exécute régulièrement et exerce l'ensemble du pipeline
  - Collecting data -> Feature Extraction -> Model training -> Model Verification -> Deployment
- Comment
  - Les tests d'intégration doivent être exécutés en continu
  - Tests d'intégration plus rapides pour une rétroaction rapide
    - Données et modèles échantillonnés

# #6 Les modèles sont testés comme des canaris avant d'être mis en production



- Incompatibilité entre la modélisation et le service
  - Le code de modélisation a tendance à être mis à jour plus fréquemment
- Comment?
  - Tester si le modèle se charge avec succès en production
  - Gardez à la fois l'ancien et le nouveau modèle simultanément

# Monitoring Tests for ML

Table IV  
BRIEF LISTING OF THE SEVEN MONITORING TESTS

1	Dependency changes result in notification.
2	Data invariants hold for inputs.
3	Training and serving are not skewed.
4	Models are not too stale.
5	Models are numerically stable.
6	Computing performance has not regressed.
7	Prediction quality has not regressed.

La prochaine session

# #1 Les modifications de dépendance entraînent une notification

- Les systèmes ML consomment des données provenant de nombreux services
  - Pannes partielles...
  - ... mises à niveau de version
  - ... et d'autres changements peuvent interrompre le flux d'entrées
- "Les erreurs dans les données d'entrée ne doivent pas être ignorées."
- Comment?
  - Surveiller toutes les dépendances
    - S'abonne aux annonces
    - Avoir une liste complète des sources d'entrée

# #6 Les performances informatiques n'ont pas régressé

- Les performances de calcul (CPU, mémoire) sont une préoccupation clé à l'échelle
  - Les DNNs peuvent être lents au moment de l'entraînement et de l'inférence
  - Le temps de réponse lent est terrible pour l'efficacité de l'entreprise
- Comment?
  - Surveiller le rendement
  - Slice a surveillé les performances par données et versions de modèle

# Le score du test ML

Le score final est calculé comme suit :

- 0,5 pour les tests manuels
- 1.0 pour les tests atomatisés
- Additionnez le score pour chaque catégorie individuellement
  - Feature, Model, Infrastructure, Monitoring
- Le score final est le **minimum** dans toutes les catégories

# Comment interpréter le score du test ML

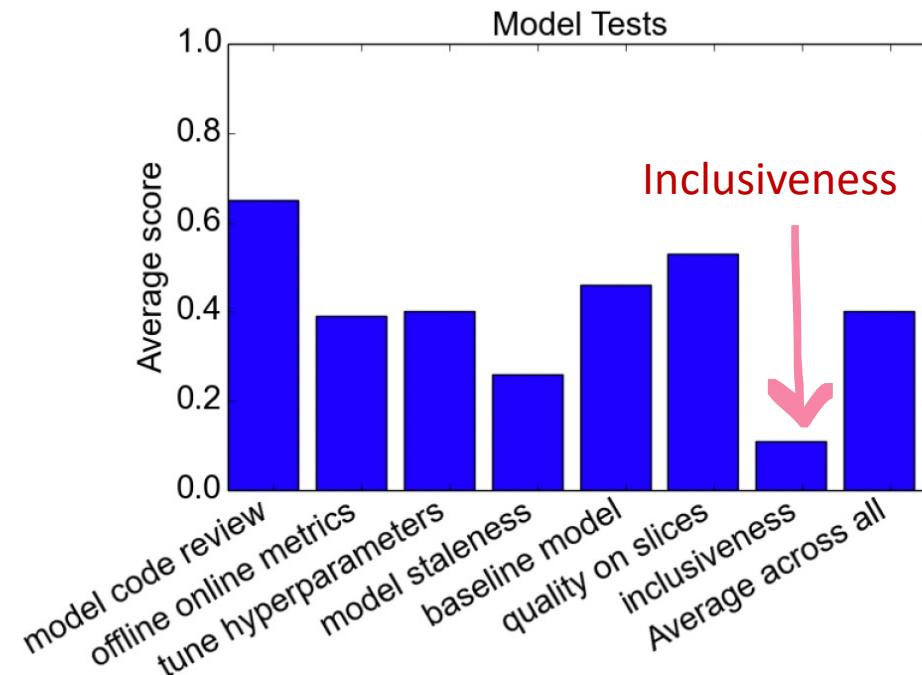
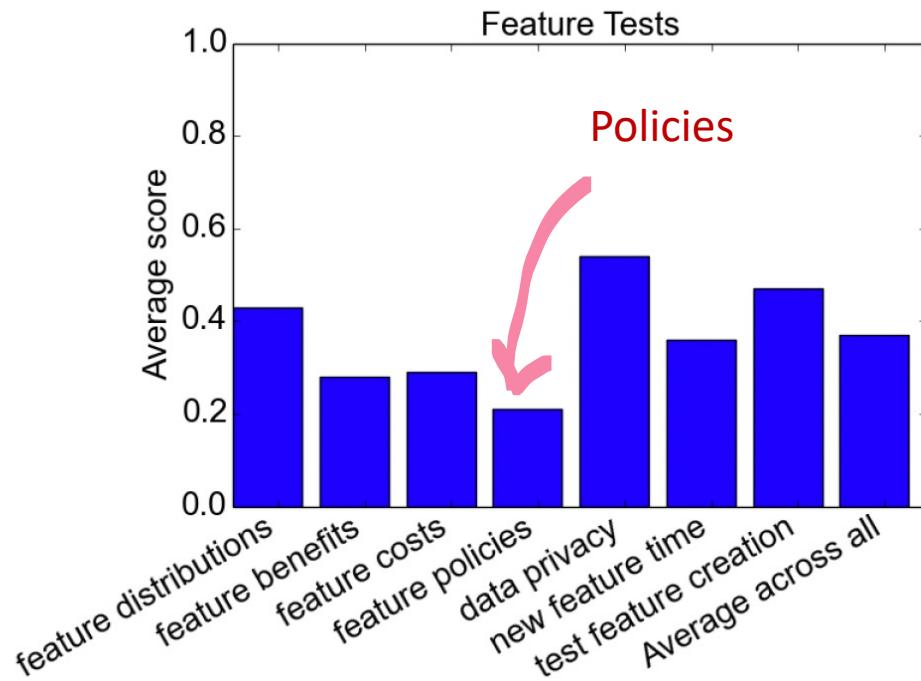
Points	Description
0	More of a research project than a productionized system.
(0,1]	Not totally untested, but it is worth considering the possibility of serious holes in reliability.
(1,2]	There's been first pass at basic productionization, but additional investment may be needed.
(2,3]	Reasonably tested, but it's possible that more of those tests and procedures may be automated.
(3,5]	Strong levels of automated testing and monitoring, appropriate for mission-critical systems.
> 5	Exceptional levels of automated testing and monitoring.

Table V

**Interpreting an ML Test Score.** THIS SCORE IS COMPUTED BY TAKING THE *minimum* SCORE FROM EACH OF THE FOUR TEST AREAS. NOTE THAT DIFFERENT SYSTEMS AT DIFFERENT POINTS IN THEIR DEVELOPMENT MAY REASONABLY AIM TO BE AT DIFFERENT POINTS ALONG THIS SCALE.

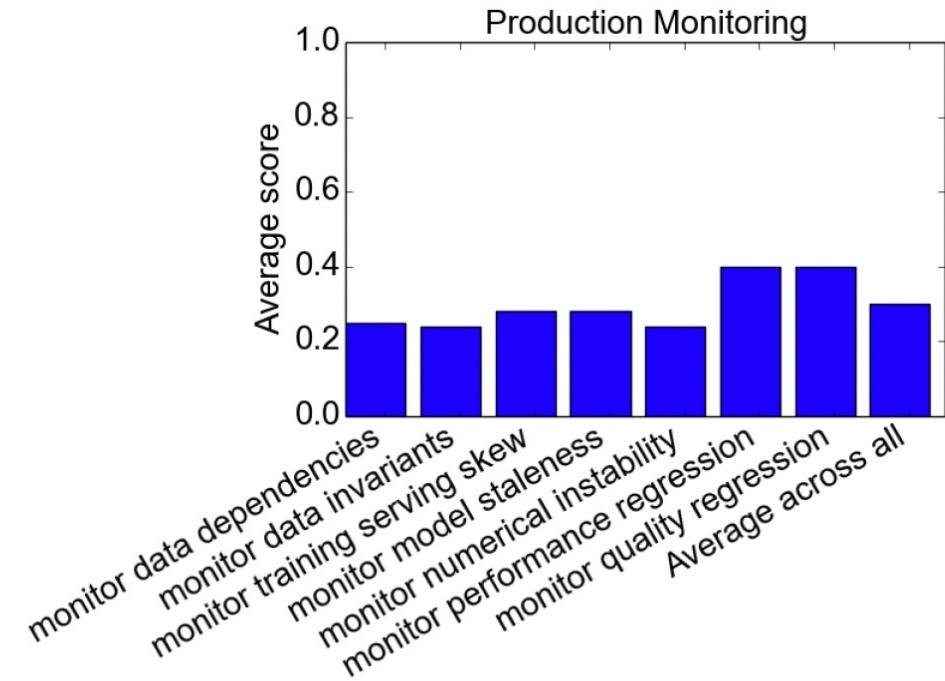
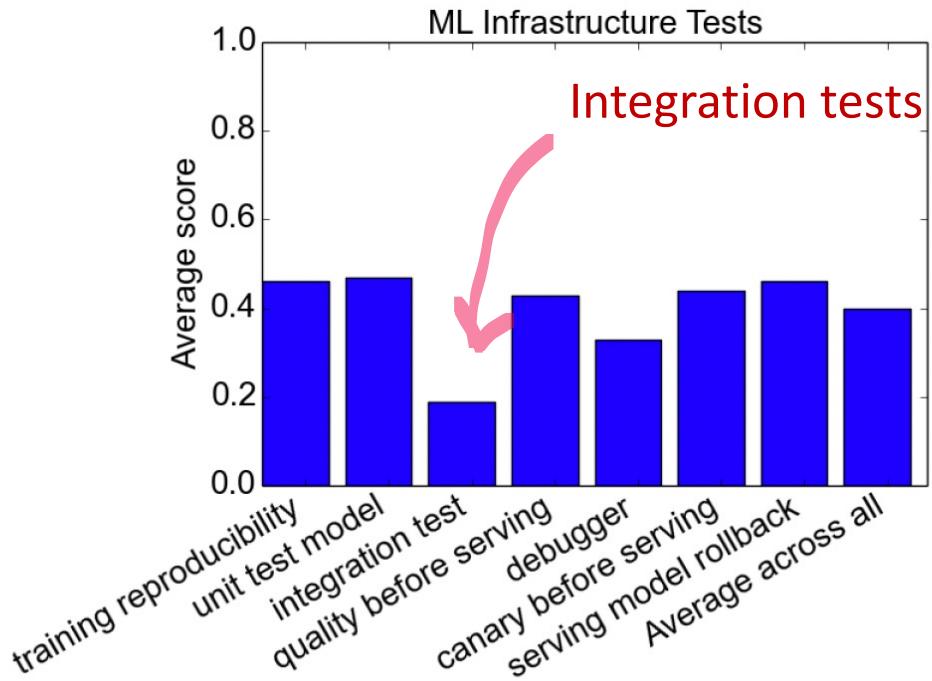
# Application du score du test ML

- 36 équipes chez Google



# Application du score du test ML

- 36 équipes chez Google



# Discussion ouverte



!

# Chapter 5 - Course Project

- Réfléchir aux dettes techniques potentielles de votre système
  - Dépendances de données
  - Anti-patrons
  - Smells
- Discutez les causes potentielles des TDs pertinentes.
  - Comment chaque TD pourrait-il affecter les qualités de **votre système**?
  - Quelles stratégies/processus seraient efficaces pour aider à les atténuer?

# Chapter 5 - Course Project

Discutez des tests de rubrique pour tester les données et le modèle (Tableaux 1 et 2)

Sélectionnez **au moins 2 de chaque test** à traiter dans le rapport et dans la mise en œuvre:

- Testing the data and Features
  - Recommandé: Tests 1,2, and 4
- Testing the Model Development
  - Recommandé: Tests 3, 5, and 6

