Anomaly Detection in Microservice-based Applications: A Comparative Study of Machine Learning Models

Darius TEMGOUA
Université de Yaoundé I
Yaoundé, Cameroun
steves.temgoua@facsciences-uy1.cm

Valéry MONTHE
Université de Yaoundé I
Yaoundé, Cameroun
valery.monthe@facsciences-uy1.cm

***Abstract*—In a constantly evolving landscape where modern software systems are increasingly adopting distributed architectures, microservices are emerging as a crucial solution. However, their distributed nature and high interconnectivity make monitoring and anomaly detection particularly challenging, as traditional methods do not always provide effective resilience. This thesis proposes a comparative study of different machine learning approaches applied to anomaly detection in microservice-based systems. The objective is to highlight the real capabilities of models such as the multi-layer perceptron (MLP), the LSTM autoencoder, and a hybrid model combining sequential representation and supervised classification.**

The experimental study is based on a dataset in which anomaly data were injected to simulate the behavior of applications at the service level and the global application level. After rigorous data preprocessing, the models were trained and then evaluated according to classic metrics (accuracy, precision, recall, F1-score) and the results reveal that despite the complexity of some models, the overall performances remain close, suggesting that in a low-noise and low-dimensional environment, simple models can be sufficient to detect anomalies effectively.

This study is part of a DevOps context in order to propose monitoring tools capable of contributing to the resilience of distributed systems.

***Keywords*—DevOps, Distributed monitoring, Microservices, Anomaly detection, Machine learning, Comparative study.**

I. Introduction

L'essor des architectures microservices, désormais standard dans le développement d'applications distribuées, offre modularité, résilience et flexibilité, mais complexifie considérablement la surveillance. Chaque service produit ses propres métriques, journaux et traces, rendant difficile la détection précoce d'anomalies, surtout dans des environnements DevOps à cycles de déploiement rapides en l'occurence avec les outils CI/CD. Les approches traditionnelles, basées sur des seuils ou règles statiques, peinent à capter les dynamiques séquentielles des données et manquent de flexibilité face à l'évolution constante des systèmes.

La détection proactive d'anomalies revêt une importance stratégique : elle limite les interruptions de service, réduit les pertes financières, renforce la satisfaction client et garantit la continuité d'activité, particulièrement dans des secteurs critiques comme les télécommunications ou la santé. Dans un contexte DevOps, elle permet de diminuer significativement le [1] **Mean Time to Detect (MTTD)** et le **Mean Time to Repair (MTTR)**, améliorant à la fois la productivité et la qualité de service.

Face à ces enjeux, l'apprentissage automatique (**machine learning**) s'impose comme une piste prometteuse. Ce travail explore une question centrale : ***dans quelle mesure la complexité d'un modèle améliore-t-elle la détection d'anomalies dans un environnement microservices ?*** En particulier, *les modèles avancés comme les autoencodeurs LSTM offrent-ils un avantage tangible par rapport à des approches plus simples telles que les perceptrons multicouches ?*

L'objectif est de proposer une solution pratique et reproductible de détection d'anomalies adaptée aux environnements microservices, en comparant les performances de modèles de complexité variable à partir de données simulées réalistes, et en analysant l'impact de cette complexité sur la qualité de détection.

II. Related Works

Plusieurs travaux de recherche récents ont proposé des approches innovantes pour la détection d'anomalies dans les systèmes microservices, en s'appuyant sur différents types de données et techniques d'apprentissage automatique.

- Grambow et al. [2] (2020) ont exploré l'utilisation d'un autoencodeur LSTM pour détecter des anomalies à partir des métriques de services dans des architectures Kubernetes. Leur étude a montré la pertinence des modèles séquentiels dans la capture des dépendances temporelles au sein des services.
- Bogatinovski et al. [3] (2020) ont proposé une méthode d'apprentissage auto-supervisée basée sur les traces distribuées. Leur approche encode les séquences d'appels de services pour construire une représentation latente, utilisée ensuite pour identifier les anomalies sans supervision explicite.
- Nobre et al. [4] (2023) ont réalisé une étude approfondie sur les défis de la détection d'anomalies dans les architectures microservices. Leur travail met en avant l'efficacité du Perceptron Multi-Couche (MLP) pour la détection au

niveau service, en montrant des scores F1 élevés sur des jeux de données injectés de fautes. Ils insistent sur la nécessité de disposer de jeux de données réalistes pour mieux entraîner les modèles.

- Silva et al. [5] (2022) ont développé un système de détection d'anomalies en exploitant des autoencodeurs dans un environnement Kubernetes. En utilisant des outils comme eBPF pour collecter des métriques noyau fines, ils ont montré que les autoencodeurs peuvent détecter efficacement les anomalies dans des contextes dynamiques à forte volumétrie.
- Raeiszadeh et al. [6] (2023) ont introduit une approche de détection en temps réel des anomalies à partir des traces distribuées dans les systèmes cloud à base de microservices. Leur modèle repose sur l'encodage des séquences de services et l'analyse en ligne des écarts de comportement, avec une attention particulière portée à la latence de détection.

Chacune de ces approches souligne l'importance d'adapter la méthode de détection aux types de données disponibles (logs, métriques, traces), aux contraintes du système (temps réel, volumétrie, distribution) et aux objectifs visés (performance, sécurité, fiabilité).

III. Methodology

A. Dataset Description

Les expérimentations de ce travail s'appuient sur un jeu de données proposé par [4], spécialement conçu pour l'étude de la détection d'anomalies dans des environnements microservices. Ce jeu de données présente l'avantage d'être **scientifiquement contrôlé** et **reproductible**, ce qui le rend particulièrement adapté à la comparaison objective de modèles d'apprentissage automatique.

1) Structure et granularité: Le jeu de données est structuré selon deux niveaux d'analyse.

- **Niveau service** : chaque observation correspond aux mesures d'un service individuel à un instant donné. Les principales variables incluent :
  - `Mean` : temps de réponse moyen (en ms) ;
  - `50th quantile` : médiane des temps de réponse, reflétant la performance typique ;
  - `99th quantile` : temps de réponse du pire cas observé ;
  - `2xx` : nombre de requêtes HTTP réussies ;
  - `4xx` / `5xx` : nombre de réponses HTTP erronées côté client ou serveur.
- **Niveau application** : données agrégées provenant de plusieurs services, offrant une vision globale du comportement du système.

2) Processus de génération et de collecte: L'environnement de collecte mis en place par [4] comprend.

- un **cluster Kubernetes** hébergeant un ensemble de microservices simulant une application complète ;
- un outil de monitoring (*Prometheus*) collectant les métriques toutes les 15 secondes ;
- un **mécanisme d'injection d'anomalies** permettant de simuler différents scénarios :
  - augmentation artificielle de la latence ;
  - génération d'erreurs HTTP (4xx et 5xx) ;
  - arrêt ou indisponibilité de services.

Les anomalies sont injectées de manière isolée ou combinée, reproduisant des situations réalistes tout en conservant une maîtrise expérimentale.

3) Étiquetage des données: Chaque observation est associée à une variable binaire `IsError` définissant l'état du système :

$$y = \begin{cases} 0, & \text{comportement normal} \\ 1, & \text{comportement anormal} \end{cases}$$

Les données se présentent ainsi sous la forme :

$$X \in \mathbb{R}^{n \times p}, \quad y \in \{0,1\}^n$$

où n est le nombre d'observations et p le nombre de variables.

B. Preprocessing

Le prétraitement des données est une étape déterminante, car la qualité de cette phase conditionne directement les performances et la stabilité des modèles d'apprentissage automatique. Les opérations suivantes ont été effectuées pour garantir des entrées propres, homogènes et exploitables.

1) Nettoyage et préparation:

- **Conversion des timestamps** : transformation des horodatages en format temporel exploitable pour l'indexation et la synchronisation des données ;
- **Harmonisation des types de données** : conversion explicite des colonnes numériques et catégorielles dans des formats appropriés ;
- **Gestion des valeurs manquantes** :
  - suppression des enregistrements lorsque l'impact sur l'information est faible ;
  - imputation (moyenne, médiane, interpolation) lorsque la donnée est jugée essentielle.

2) Normalisation des variables: Les variables présentant des échelles différentes (par exemple, temps en millisecondes et taux d'erreurs en pourcentage) ont été ramenées sur une échelle commune via un *Min-Max Scaling* :

$$x_{\text{scaled}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Cette transformation garantit que toutes les variables contribuent de manière équilibrée à l'apprentissage, en évitant qu'une seule domine le processus.

C. Models Evaluated

Afin de couvrir différentes approches d'apprentissage, trois modèles ont été retenus dans le cadre de cette étude :

1) un modèle supervisé classique pour données tabulaires ;
2) un modèle non supervisé adapté aux séries temporelles ;
3) un modèle hybride combinant deux paradigmes.

1) Perceptron Multicouche (MLP): Le Perceptron Multicouche (*Multi-Layer Perceptron*, MLP) a été implémenté à l'aide de la bibliothèque `scikit-learn`. Ce réseau de neurones *feedforward* est structuré comme suit :

- **Couche d'entrée** : reçoit les variables explicatives prétraitées ;
- **Couches cachées** : entre 4 et 16 neurones avec des fonctions d'activation non linéaires telles que `ReLU` ou `tanh` ;
- **Couche de sortie** : un seul neurone activé par une fonction `sigmoïde`, produisant une probabilité d'appartenance à la classe `anomalie`.

L'optimisation a été réalisée avec les solveurs `adam` (rapide et adaptatif) et `lbfgs` (plus stable pour petits ensembles de données). Une recherche d'hyperparamètres (`GridSearchCV`) a été menée pour ajuster la profondeur, le taux d'apprentissage et la régularisation. En raison du déséquilibre des classes, le paramètre `class_weight='balanced'` a été activé.

2) Autoencodeur LSTM: L'Autoencodeur LSTM (*Long Short-Term Memory Autoencoder*) a été développé avec `Keras/TensorFlow` afin d'exploiter les dépendances temporelles présentes dans les données microservices. Ce modèle, de nature non supervisée, fonctionne en trois étapes :

1) **Encodage** : une ou plusieurs couches LSTM (32 à 64 unités) compressent la séquence d'entrée dans une représentation latente compacte ;
2) **Décodage** : reconstruction de la séquence initiale à partir de cette représentation ;
3) **Évaluation** : l'erreur de reconstruction est utilisée comme score d'anomalie, et un seuil empirique (déterminé sur un ensemble de validation) permet de classer l'observation.

Les fonctions d'activation `tanh` et la perte `MSE` (*Mean Squared Error*) ont été utilisées. L'entraînement s'est déroulé uniquement sur des séquences normales, sur 20 à 50 époques, avec une taille de lot (*batch size*) de 16 et l'optimiseur `Adam`.

3) Modèle Hybride MLP + LSTM: Le modèle hybride combine deux sous-architectures :

- une **branche LSTM** dédiée à l'analyse des variables séquentielles (métriques temporelles collectées dans le temps) ;
- une **branche dense (MLP)** pour traiter les variables statiques ou dérivées ne possédant pas de structure temporelle.

Les sorties de ces deux branches sont concaténées, puis transmises à une couche dense finale avec activation `sigmoïde`. L'entraînement est réalisé sur 18 à 50 époques, avec un `dropout` compris entre 0,2 et 0,5 pour limiter le surapprentissage. Ce type d'architecture permet de capturer simultanément :

- les **dynamiques temporelles** via la branche LSTM ;
- les **relations instantanées** via la branche dense.

Cette combinaison est particulièrement pertinente dans le contexte microservices, où certaines anomalies se manifestent par des ruptures dans les tendances temporelles tandis que d'autres sont visibles à un instant donné.

D. Experimental Setup

Cette section décrit l'environnement logiciel et matériel utilisé pour les expérimentations, ainsi que les métriques et outils d'évaluation mobilisés.

1) Environnement logiciel: Les expérimentations ont été menées en **Python 3.x**, avec les bibliothèques suivantes :

- `Pandas`, `NumPy` : manipulation et structuration des données ;
- `Matplotlib`, `Seaborn` : visualisation et analyse graphique ;
- `Scikit-learn` : implémentation du MLP, prétraitement des données et calcul des métriques d'évaluation ;
- `TensorFlow/Keras` : implémentation des modèles LSTM Autoencodeur et Hybride.

2) Configuration matérielle: Les tests ont été exécutés sur une machine disposant de la configuration suivante :

- Processeur : Intel Core i7 (10^{e} génération) ;
- Mémoire vive : 32 Go RAM ;
- Carte graphique : GPU NVIDIA avec 12 Go de mémoire dédiée.

3) Métriques d'évaluation: Plusieurs métriques complémentaires ont été utilisées pour évaluer la performance des modèles. Elles permettent d'apprécier la capacité des modèles à distinguer correctement les comportements normaux des anomalies, en particulier dans un contexte de classes déséquilibrées.

- **Accuracy** : proportion totale de prédictions correctes sur l'ensemble du jeu de données.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

où TP (True Positives) représente les anomalies correctement détectées, TN (True Negatives) les observations normales bien classées, FP (False Positives) les fausses alertes et FN (False Negatives) les anomalies non détectées.

- **Precision** : proportion d'anomalies correctement identifiées parmi celles prédites comme telles (réduction des fausses alertes).

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall (Sensibilité)** : proportion d'anomalies correctement détectées parmi toutes les anomalies présentes (réduction des anomalies manquées).

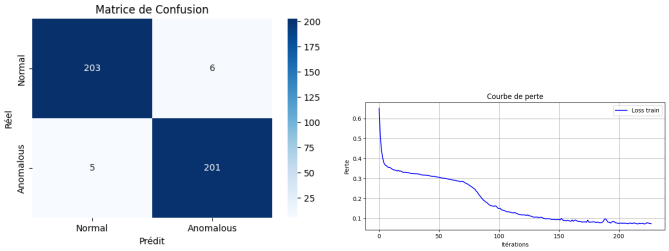
$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-score** : moyenne harmonique entre la précision et le rappel, offrant un compromis entre les deux.

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

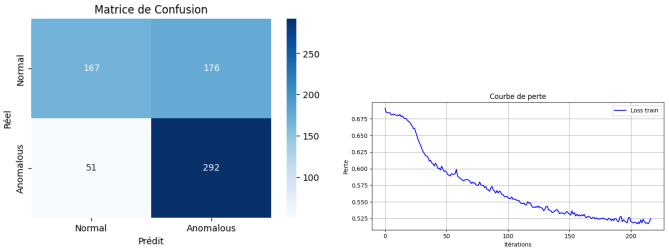
Ces métriques sont complétées par des outils visuels pour interpréter les résultats :

- **Matrices de confusion** : représentation de la répartition des vraies et fausses prédictions par classe ;



(a) Matrice de confusion (b) Courbe de perte

Figure 1. MLP — **niveau service**.



(a) Matrice de confusion (b) Courbe de perte

Figure 2. MLP — **niveau application**.

- **Courbes d'apprentissage** (perte et précision) : suivi de la convergence des modèles au fil des époques ;
- **Distribution des erreurs de reconstruction** : spécifique aux Autoencodeurs LSTM, permettant d'évaluer la séparation entre séquences normales et anormales.

Ces éléments assurent une évaluation complète, combinant analyse chiffrée et interprétation visuelle, afin d'identifier les forces et limites de chaque modèle dans le contexte de la détection d'anomalies sur architectures microservices.

IV. Results and Discussion

A. Résultats expérimentaux

Les modèles ont été évalués selon des métriques standards de classification : *accuracy*, *precision*, *recall* et *F1-score*. Les matrices de confusion permettent de visualiser les proportions de vrais positifs (VP), vrais négatifs (VN), faux positifs (FP) et faux négatifs (FN), tandis que les courbes de perte et d'apprentissage offrent un aperçu de la convergence des entraînements et des risques de surapprentissage. L'analyse des courbes est essentielle pour interpréter la stabilité de l'apprentissage, la vitesse de convergence et la capacité du modèle à généraliser.

*1) **Perceptron Multicouche (MLP)**:*

a) Configuration.: Le MLP comporte deux couches cachées avec activation `tanh`. Au **niveau service**, la configuration $(16, 8)$ est retenue ; au **niveau application**, $(8, 16)$. Les hyperparamètres ont été ajustés via `GridSearchCV` pour maximiser la performance globale.

b) Niveau service.: **Résumé des métriques.**

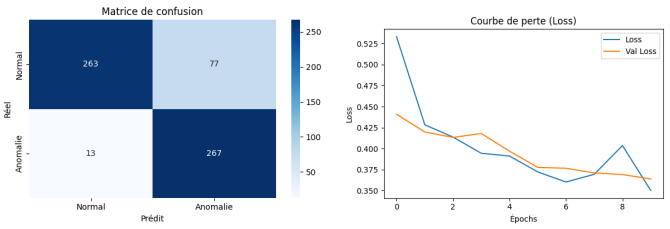
- Accuracy = 0,97 Precision = 0,97 Recall = 0,98 F1-score = 0,97

Analyse. La matrice (*fig 1.(a)*) montre très peu de faux positifs/négatifs, confirmant une excellente robustesse. La courbe (*fig 1.(b)*) décroît rapidement puis se stabilise sans oscillations marquées, indiquant un apprentissage stable et un surajustement limité.

c) Niveau application.: **Résumé des métriques.**

- Accuracy = 0,52 Precision = 0,62 Recall = 0,12 F1-score = 0,20

Analyse. La matrice (*fig 2.(a)*) révèle un grand nombre de faux négatifs (rappel très faible) : le modèle manque une large partie des anomalies. Malgré une perte finale basse (*fig 2.(b)*), la généralisation reste médiocre, suggérant un surapprentissage sur des motifs faiblement discriminants.



(a) Matrice de confusion (b) Courbe de perte

Figure 3. LSTM autoencodeur — **niveau service**.

d) Lecture croisée.: Au **niveau service**, la granularité fine (latences spécifiques, distribution détaillée des codes HTTP, variabilité locale) fournit des signaux nets que le MLP sépare efficacement. Au **niveau application**, l'agrégation dilue ces signaux et masque les anomalies locales ; le modèle apprend des motifs peu généralisables, expliquant le *recall* très bas malgré une courbe de perte visuellement "propre".

*2) **Autoencodeur LSTM**:* L'autoencodeur LSTM a été conçu pour apprendre uniquement la structure des données normales. En phase d'inférence, les observations présentant une erreur de reconstruction supérieure à un seuil prédéfini sont considérées comme des anomalies. Cette approche est particulièrement adaptée aux séries temporelles, car les couches LSTM capturent les dépendances temporelles et les dynamiques séquentielles.

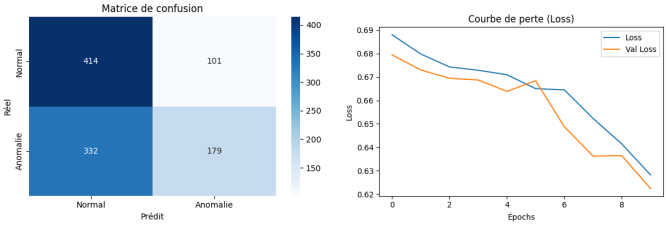
a) Niveau service.: **Résumé des métriques.**

- Accuracy = 0,85 Precision = 0,78 Recall = 0,95 F1-score = 0,86

Analyse. La matrice de confusion (*fig 3.(a)*) montre que sur 550 échantillons normaux, 263 sont correctement classés et 77 faussement signalés comme anomalies. Pour les anomalies, 267 sur 280 sont détectées correctement, expliquant le **rappel élevé (95 %)**. Cette sensibilité réduit le risque de laisser passer des anomalies, tandis que la précision de 0,78 traduit encore 22 % de faux positifs (fluctuations normales interprétées comme anomalies).

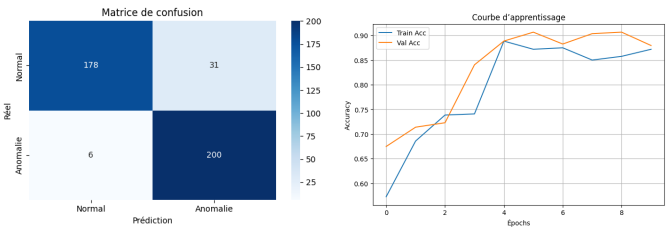
Courbe de perte. La courbe (*fig 3.(b)*) illustre un apprentissage stable :

1) **Phase initiale (0–2 époques)** : forte baisse (de $\approx 0{,}69$ à $\approx 0{,}67$), capture rapide des structures majeures.
2) **Phase intermédiaire (3–6 époques)** : décroissance plus lente, ajustement sur des motifs subtils.



(a) Matrice de confusion (b) Courbe de perte

Figure 4. LSTM autoencodeur — **niveau application**.



(a) Matrice de confusion (b) Courbe d'apprentissage

Figure 5. Modèle hybride — **niveau service**.

3) **Phase finale (7–9 époques)** : perte $\approx 0{,}63$ avec validation parallèle, convergence sans surapprentissage notable.

Interprétation. Le couple *rappel élevé / précision modérée* correspond au compromis classique en détection d'anomalies : on privilégie la sensibilité au prix de davantage de fausses alertes — acceptable en supervision critique.

b) Niveau application.: **Résumé des métriques.**

- Accuracy = 0,58 Precision = 0,64 Recall = 0,35 F1-score = 0,45

Analyse. La matrice (*fig 4.(a)*) révèle que sur 515 échantillons normaux, 414 sont correctement classés mais 101 sont faussement détectés comme anomalies. Sur 511 anomalies réelles, seules 179 sont détectées, laissant 332 faux négatifs — d'où le **rappel faible (35 %)** — signe de la difficulté à distinguer anomalies et fluctuations normales au niveau agrégé.

Courbe de perte. Le profil (*fig 4.(b)*) diffère du niveau service :

1) **Phase initiale (0–1 époque)** : perte passant de $\approx 0{,}53$ à $\approx 0{,}43$.
2) **Phase intermédiaire (2–6 époques)** : poursuite de la baisse avec fluctuations, apprentissage moins stable.
3) **Phase finale (7–9 époques)** : convergence vers $\approx 0{,}35$ sans gain de rappel, suggérant un surajustement aux motifs d'entraînement.

Interprétation. Les signaux applicatifs, plus bruités et moins corrélés temporellement, font perdre à l'autoencodeur LSTM son avantage séquentiel. Faute de motifs discriminants nets après agrégation, une proportion importante d'anomalies n'est pas détectée.

*3) **Modèle hybride LSTM + MLP**:* Le modèle hybride vise à combiner deux paradigmes complémentaires :

- **LSTM** : exploite les dépendances temporelles pour analyser les variations séquentielles et contextuelles des métriques.
- **MLP** : traite efficacement les variables indépendantes ou faiblement corrélées dans un cadre non séquentiel, permettant d'intégrer des signaux plus statiques.

La logique sous-jacente est que les métriques issues d'un environnement microservices contiennent à la fois des patterns dynamiques (latence, throughput) et des informations plus stables (répartition des codes HTTP, moyennes glissantes). En combinant ces deux approches, le modèle cherche à capturer simultanément la dimension temporelle et la dimension statique.

a) Niveau service.: **Performances globales.**

- Accuracy = 0,91 Precision = 0,87 Recall = 0,97 F1-score = 0,92

Analyse. La matrice (*fig 5.(a)*) montre un **rappel élevé (97 %)** : les anomalies sont détectées presque intégralement. Le volume de faux positifs est inférieur à celui du LSTM seul, ce qui suggère que la composante MLP filtre des fluctuations normales.

Courbe d'apprentissage. La courbe (*fig 5.(b)*) présente une montée rapide puis une convergence stable, avec peu d'écart entraînement/validation — *surapprentissage limité*.

Interprétation. Au niveau service, des signaux stables et structurés permettent au LSTM de capter la dynamique, tandis que le MLP stabilise la décision finale. Le compromis *sensibilité/spécificité* est favorable : moins de fausses alertes sans perte des vraies anomalies.

b) Niveau application.: **Performances globales.**

- Accuracy = 0,50 Precision = 0,51 Recall = 0,14 F1-score = 0,22

Analyse. La matrice (*fig 6.(a)*) met en évidence une **chute du rappel (14 %)** : la majorité des anomalies échappent à la détection, avec de nombreux faux négatifs.

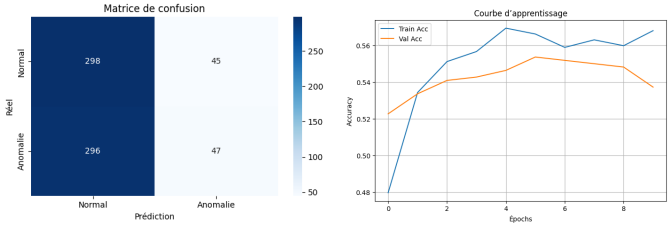
Courbe d'apprentissage. La courbe (*fig 6.(b)*) progresse lentement puis atteint un *plateau prématuré*. La faible marge entraînement/validation indique que la limite vient surtout de la nature des données (signaux dilués) plutôt que d'un manque de capacité du modèle.

Interprétation. Au niveau application, l'agrégation dilue les signaux temporels et masque les anomalies locales : la composante LSTM perd en pertinence et le MLP, privé de variables distinctives, ne compense pas. Le gain de l'hybride n'apparaît que si les données contiennent des informations discriminantes suffisantes — ce qui est vrai au niveau service, beaucoup moins au niveau application.

4) Résumé comparatif et tendances:

a) Observation générale :: Trois tendances structurantes se dégagent de l'analyse :

1) **Supériorité systématique des performances au niveau service** : Les trois modèles obtiennent des scores nettement plus élevés lorsqu'ils sont appliqués sur les données désagrégées par service. Cette différence s'explique par la richesse des signaux capturés à ce niveau : latences spécifiques, variations locales de charge, distribution fine



(a) Matrice de confusion (b) Courbe d'apprentissage

Figure 6. Modèle hybride — **niveau application**.

des codes HTTP. En revanche, au niveau application, l'agrégation des métriques dilue ces signaux et atténue les ruptures temporelles, ce qui rend la tâche de détection plus complexe. Les modèles doivent alors inférer des anomalies à partir de motifs beaucoup moins distinctifs.

2) **Limitation du LSTM par la granularité des données** : Le LSTM excelle dans la modélisation de séries temporelles lorsqu'il existe une structure séquentielle forte. Au niveau service, cette condition est remplie, et le modèle atteint un rappel élevé (95%), détectant presque toutes les anomalies. Cependant, lorsque les données sont agrégées au niveau application, les patterns temporels deviennent moins marqués, entraînant une chute significative du rappel (35%). Autrement dit, la valeur ajoutée du LSTM est directement proportionnelle à la clarté des dépendances temporelles dans les données d'entrée.
3) **Rendement conditionnel des modèles hybrides** : La combinaison LSTM + MLP offre un compromis solide au niveau service (F1-score = 0.92) grâce à la complémentarité entre :
   - le LSTM, qui capture les dynamiques temporelles ;
   - le MLP, qui filtre les variations normales et améliore la précision.

   En revanche, au niveau application, cette combinaison ne parvient pas à compenser le manque de signaux discriminants. Les performances restent proches, voire inférieures, à celles des modèles individuels. Cela illustre un principe clé : un modèle complexe ne surperforme que si les données contiennent des informations distinctives exploitables.

b) Lecture stratégique des résultats ::

- Pour un déploiement opérationnel où la granularité des métriques est fine (niveau service), le **MLP** reste la solution la plus efficace en précision globale, tandis que le **LSTM** et l'hybride sont préférés si la sensibilité aux anomalies (rappel élevé) est prioritaire.
- Dans des contextes où seules des métriques agrégées au niveau application sont disponibles, les performances sont limitées quelle que soit l'architecture utilisée. L'enjeu devient alors moins un choix de modèle qu'une optimisation de la collecte de données pour augmenter la qualité du signal.

Table I
RÉSUMÉ COMPARATIF DES PERFORMANCES DES MODÈLES SUR LES DEUX NIVEAUX D'ANALYSE.

Modèle	Niveau	Acc.	Prec.	Rec.	F1
MLP	Service	0.97	0.97	0.98	0.97
MLP	Application	0.52	0.62	0.12	0.20
LSTM	Service	0.85	0.78	0.95	0.86
LSTM	Application	0.58	0.64	0.35	0.45
Hybride	Service	0.91	0.87	0.97	0.92
Hybride	Application	0.50	0.51	0.14	0.22

- Ces résultats rappellent l'importance de l'alignement entre la nature des données et la conception du modèle : la sophistication algorithmique n'apporte un gain que si elle répond à une structure exploitable dans les données.

B. Discussion

Les résultats confirment un écart net entre **niveau service** et **niveau application**. Ce contraste s'explique d'abord par la granularité des signaux, avec des conséquences directes sur la détectabilité des anomalies et l'utilité relative des architectures.

Analyse par niveau:

a) Au niveau service.: Signaux temporels et statistiques *clairs* (latences, répartitions HTTP, variations locales) et variabilité modérée $\Rightarrow$ modèles facilités.

- **MLP** : sépare efficacement des relations non linéaires sans dépendances séquentielles complexes.
- **LSTM** : capte les motifs séquentiels, *rappel* très élevé (détection quasi exhaustive).
- **Hybride** : combine sensibilité (LSTM) et stabilité décisionnelle (MLP), moins de faux positifs.

Conclusion : des signaux forts et localisés expliquent des *accuracy* souvent $> 90\%$.

b) Au niveau application.: Agrégation multi-services $\Rightarrow$ dilution des signaux et bruit statistique ; une dégradation locale peut être masquée par le reste.

- **MLP** : frontières de décision floues, *rappel* très faible (p. ex. 12%).
- **LSTM** : quelques motifs globaux captés, mais *rappel* modeste faute de structure temporelle marquée.
- **Hybride** : peu d'apport, la limite vient surtout de l'information disponible, pas du choix d'architecture.

Conclusion : la granularité des données conditionne la sensibilité ; l'agrégation excessive la réduit fortement.

Forces et limites des approches:

- **MLP** — *Forces* : simple, rapide, robuste avec signaux statiques/forts. *Limites* : ne capture pas les dynamiques séquentielles, sensible au bruit agrégé.
- **Autoencodeur LSTM** — *Forces* : excellente sensibilité aux anomalies subtiles (modélisation séquentielle), utile quand les FN coûtent cher. *Limites* : seuil d'erreur crucial, davantage de FP, entraînement plus coûteux.
- **Hybride LSTM+MLP** — *Forces* : couvre variables temporelles et statiques, bon compromis quand les signaux sont variés. *Limites* : complexité accrue sans gain si l'information discriminante est faible.

Implications pratiques:

- Privilégier la **surveillance au niveau service** pour des détections rapides et localisées.
- Conserver une **vue application** pour les tendances globales et pannes étendues, en acceptant une sensibilité moindre.
- **Combiner les deux niveaux** en production (collecte fine + agrégée) pour maximiser couverture et réactivité.

V. Conclusion and Future Work

Ce travail a exploré et comparé trois approches d'apprentissage automatique pour la détection d'anomalies dans des environnements microservices : un perceptron multicouche (MLP), un autoencodeur LSTM et un modèle hybride LSTM+MLP. Les expérimentations, menées aux niveaux *service* et *application* et évaluées via l'accuracy, la précision, le rappel et le F1-score, ont permis de dégager plusieurs constats.

Les résultats montrent une nette supériorité du suivi au niveau *service*, avec un F1-score atteignant 0.97 pour le MLP, alors que les performances chutent fortement au niveau *application* en raison de la perte de granularité et de la dilution des signaux. Les modèles plus complexes, tels que le LSTM ou l'hybride, n'apportent un gain significatif que lorsque des motifs temporels marqués coexistent avec des variables statiques informatives.

Ces observations soulignent que la granularité et la qualité des données influencent davantage les performances que la seule complexité algorithmique, et que le choix du modèle doit être aligné avec la nature des signaux surveillés.

Les perspectives de ce travail incluent : (i) l'enrichissement des données avec des journaux applicatifs, des traces distribuées et des métriques métier ; (ii) l'exploration d'architectures intégrant des mécanismes d'attention pour mieux capter les dépendances à long terme ; (iii) l'intégration de méthodes d'explicabilité telles que SHAP ou LIME afin d'améliorer la transparence et l'adoption par les équipes DevOps ; et (iv) la validation de ces approches en environnement de production.

Ces résultats ouvrent la voie à des systèmes de détection d'anomalies plus précis, interprétables et adaptés aux contraintes opérationnelles des architectures microservices modernes.

References

[1] M. A. Morais, "Real-time incident detection in public bus systems using machine learning," in *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2023.

[2] K. Grambow and et al., "Using lstm autoencoders for anomaly detection in kubernetes," in *IEEE International Conference on Cloud Engineering*, 2020, pp. 172–178.

[3] J. Bogatinovski and et al., "Self-supervised anomaly detection in microservices using distributed traces," in *IEEE International Conference on Cloud Computing*, 2020, pp. 180–187.

[4] J. Nobre, E. J. S. Pires, and A. Reis, "Anomaly detection in microservice-based systems," *Applied Sciences*, vol. 13, p. 7891, 2023, dataset available at: https://doi.org/10.6084/m9.figshare.22726298;. [Online]. Available: https://doi.org/10.3390/app13137891

[5] M. de Silva, S. Mahadura, S. Daniel, L. Rupasinghe, M. Kumarapeli, and C. Liyanapathirana, "Anomaly detection in microservice systems using autoencoders," in *2022 4th International Conference on Advancements in Computing (ICAC)*. IEEE, 2022, pp. 488–493. [Online]. Available: https://doi.org/10.1109/ICAC57685.2022.10025259

[6] M. Raeiszadeh, A. Ebrahimzadeh, A. Saleem, R. H. Glitho, J. Eker, and R. A. F. Mini, "Real-time anomaly detection using distributed tracing in microservice cloud applications," in *2023 IEEE International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2023.

[7] M. Youssfi, "Blog 1 : D'une architecture monolithique vers une architecture micro-services," *Adria Business and Technology*, 2022. [Online]. Available: https://adria-bt.com/en/challenges-et-bonnes-pratiques-de-mise-en-oeuvre-des-architectures-micro/

[8] J. Lewis and M. Fowler, "Microservices: A definition of this new architectural term," *Scientific Research*, 2014. [Online]. Available: https://martinfowler.com/microservices/

[9] P. Wizenty, J. Sorgalla, F. Rademacher, and S. Sachweh, "Magma: Build management-based generation of microservice infrastructures," in *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings*, 2017, pp. 61–65.

[10] L. Baresi and M. Garriga, "Microservices: The evolution and extinction of web services?" in *Microservices: Science and Engineering*, 2020, pp. 3–28.

[11] L. Giamattei, A. Guerriero, and R. P. et al, "Monitoring tools for devops and microservices: A systematic grey literature review," *The Journal of Systems and Software*, vol. 208, p. 111906, 2024. [Online]. Available: https://doi.org/10.1016/j.jss.2023.111906

[12] Z. Chen *et al.*, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 26th ACM SIGKDD*, 2020.

[13] G. Muruti, F. A. Rahim, and Z.-A. bin Ibrahim, "A survey on anomalies detection techniques and measurement methods," in *2018 IEEE Conference on Applications, Information and Network Security (AINS)*. IEEE, 2023.

[14] A. Behera, C. R. Panigrahi, and R. Patel, "A comparative analysis of anomaly detection from microservice generated unstructured logs," *International Journal of Advanced Research in Computer Science*, vol. 14, no. 6, pp. 54–59, 2023. [Online]. Available: https://www.researchgate.net/publication/377448760