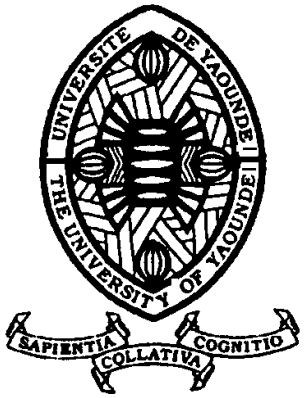

Université de Yaoundé 1
Département d'Informatique

DÉTECTION D„ANOMALIES DANS LES APPLICATIONS BASÉES SUR LES ARCHITECTURES MICROSERVICES : Étude comparative de modèles d'apprentissage automatique

---

Mémoire de fin de cycle Présenté et soutenu par :

TEMGOUA PASSO Darius Steves

Matricule :

16U2076

En vue de l'obtention du :

DIPLÔME DE MASTER RECHERCHE EN INFORMATIQUE

Option :

GENIE LOGICIEL ET SYSTÈME D'INFORMATION

Sous la direction de :

Dr. Valéry MONTHE

Chargé de cours, Université de Yaoundé 1

ANNÉE ACADÉMIQUE 2024-2025

Dédicaces

À mon feu papa **TEMGOUA Jean Paul** et mon fils **SOUNGMO TEMGOUA Kâaris**...

Remerciements

Avant tout, je tiens à exprimer ma profonde gratitude envers mes ancêtres, les esprits, la nature et toutes les personnes qui m'ont guidé tout au long de cette aventure académique. Leur présence m'a apporté sagesse et inspiration, me permettant de traverser les défis avec sérénité. Chaque échange a enrichi mon expérience et façonné ma compréhension du travail à accomplir et du monde qui m'entoure.

Je cite, entre autres :

- **Dr. Monthé Valéry**, qui a accepté de me prendre sous son encadrement et dont les conseils et la disponibilité constants m'ont permis de surmonter les difficultés et d'acquérir une meilleure compréhension du travail à réaliser, me permettant ainsi de développer au mieux mes compétences.
- Mes parents, **M. et Mme TEMGOUA**, qui m'ont permis, dès le bas âge, d'avoir accès à l'éducation et qui ont toujours su être présents pour me prodiguer les conseils nécessaires chaque fois que j'en avais besoin, me permettant d'arriver là où j'en suis aujourd'hui.
- Mes mamans, **Clarisse** et **Antoinette Ngoumnaï**, pour leur présence constante, leurs conseils et leur soutien multiforme.
- Papa **Aghoagni Jean**, pour sa présence constante, ses conseils et son soutien multiforme.
- **Dr. Aminou Halidou**, chef du département d'informatique, ainsi que tout le corps enseignant du département, pour le cadre mis en place afin d'assurer notre formation, et dont les parcours et les conseils ont été une source de motivation.
- Toute l'équipe de recherche **Monthe Team Research**, pour leurs contributions, leurs suggestions et les moments d'échange durant toutes nos activités.
- Mes amis **Nitcheu Hermann**, **Kouagnang Franck** et **Manfouo Ridano**, pour leurs conseils et leur soutien multiforme.
- Mes frères et sœurs : **Ngoumnaï Boris**, **Fouegap Maryline**, **Temgoua Pavel**, **Ngoumnaï Loetitia**, **Temgoua Loïc**, **Ngoumnaï Sydselle**, **Soungmo Kerry**, **Ngoune Antony**, pour leur soutien multiforme.
- Ma fiancée, **Ngo Gweth**, pour son soutien inconditionnel et son amour constant. Merci d'avoir cru en moi et d'avoir toujours été là pour m'encourager.
- Tous mes autres amis et toutes les personnes ayant participé, de près ou de loin, à ce travail.

Résumé

Dans un paysage en constante évolution où les systèmes logiciels modernes tendent de plus en plus vers des architectures distribuées, les microservices s'imposent comme une solution incontournable. Toutefois, leur nature répartie et leur forte interconnectivité rendent la surveillance et la détection d'anomalies particulièrement difficiles car les méthodes traditionnelles ne permettent pas toujours d'apporter une résiliance éfficace. Ce mémoire propose une étude comparative de différentes approches d'apprentissage automatique appliquées à la détection d'anomalies dans les systèmes basés sur les microservices.

L'objectif est de mettre en lumière les capacités réelles de modèles tels que le perceptron multicouche (MLP), l'autoencodeur LSTM et un modèle hybride combinant représentation séquentielle et classification supervisée.

L'étude expérimentale repose sur un jeu de données dans lequel les données d'anomalies ont été injectées pour simuler le comportement des applications niveau service et niveau application globale. Après un prétraitement rigoureux des données, les modèles ont été entrainé puis sont évalués selon des métriques classiques (accuracy, précision, rappel, F1-score) et les résultats révèlent que malgré la complexité de certains modèles, les performances globales restent proches, suggérant que dans un environnement à faible bruit et à faible dimensionnalité, des modèles simples peuvent suffire à détecter efficacement les anomalies.

Cette étude s'inscrit dans un contexte DevOps afin de proposer des outils de monitoring capables de contribuer à la résilience des systèmes distribués.

Mots-clés : DevOps, Monitoring distribué, Microservices, Détection d'anomalies, Apprentissage automatique, Étude comparative.

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.

Table des matières

Dédicace i

Remerciements ii

Résumé iii

Abstract iv

1 Introduction 1

1.1 Contexte et motivation 1
1.2 Intérêt socio-économique et pratique 2
1.3 Problématique 2
1.4 Objectifs de recherche 3
1.5 Démarche méthodologique 3
1.6 Structure du document 3

2 État de l'art 4

2.1 Les architectures microservices 4
2.1.1 Origine 4
2.1.2 Définition et caractéristiques 5
2.1.3 Représentation Architecturale 6
2.1.4 Challenges et défis des microservices 6
2.2 Surveillance et monitoring dans les systèmes distribués 7
2.3 Détection d'anomalies : généralités 7
2.3.1 Définitions et enjeux de la détection d'anomalies 7
2.3.2 Approches de détection d'anomalies 8
2.3.3 Évaluation des performances : métriques essentielles 9
2.4 Détection d'anomalies dans les systèmes à microservices 11
2.4.1 Spécificités des architectures microservices 11
2.4.2 Types d'anomalies observées dans les microservices 11

2.4.3 Méthodes appliquées dans le contexte microservices 11
2.5 Travaux connexes 12

3 Méthodologie **14**
3.1 Présentation du jeu de données 14
3.2 Prétraitement des données 16
3.3 Environnement d'expérimentation 17
3.4 Description des modèles et configurations 17
3.4.1 Perceptron Multicouche (MLP) 18
3.4.2 Autoencodeur LSTM 18
3.4.3 Modèle Hybride MLP + LSTM 19
3.5 Méthodes d'évaluation 19

4 Résultats et discussions **21**
4.1 Résultats expérimentaux 21
4.1.1 Perceptron Multicouche (MLP) 21
4.1.2 Autoencodeur LSTM 25
4.1.3 Modèle hybride LSTM + MLP 27
4.1.4 Résumé comparatif et tendances 30
4.2 Discussion 31

5 Conclusion **34**

Table des figures

2.1 Comparaison entre une architecture monolithique et microservices. 5
2.2 Description simplifié d'une architecture microservice. 6
2.3 Cartographie des techniques de détections d'anomalies [9]. 9

3.1 Architecture fonctionnelle de mise en oeuvre 15
3.2 Structure des colonnes principales du jeu de données aux niveaux service et application. 16

4.1 Matrice de confusion du MLP – niveau service. 22
4.2 Courbe de perte du MLP – niveau service. 22
4.3 Matrice de confusion – MLP niveau application. 23
4.4 Courbe de perte – MLP niveau application. 24
4.5 Matrice de confusion et courbe de perte – LSTM autoencodeur niveau service. 25
4.6 Résultats du LSTM autoencodeur – niveau application. 26
4.7 Matrice de confusion et courbe d'apprentissage – Modèle hybride, niveau service. 27
4.8 Matrice de confusion et courbe d'apprentissage – Modèle hybride, niveau application. 28

Liste des tableaux

2.1 Comparaison des approches de détection d'anomalies dans les systèmes microservices 13

4.1 Résumé comparatif des performances des modèles sur les deux niveaux d'analyse. 30

Chapitre 1

Introduction

Dans ce chapitre, nous travaillons à situer notre travail de recherche. Dans un premier temps, nous allons positionner le contexte dans lequel ce travail est réalisé, puis définir la problématique, ensuite la question de recherche et enfin les objectifs de recherche. Ce chapitre donne la structure de notre travail dans le but de mieux se situer dans le document.

1.1 Contexte et motivation

Suite à la crise du logiciel dans les années 60, la mise en pratique de l'ingénierie logicielle qui est l'application des principes de l'ingénierie à la conception, au développement, au test, ainsi qu'au déploiement et à la gestion de logiciels et des systèmes d'information a contribué à l'évolution des technologies ce qui à permis la naissance des applications web, mobile et de bureau. Son évolution croissante est liée à la complexité des systèmes d'information qui tendent de plus en plus a évoluer d'ou la naissance des architectures microservices qui aujourd'hui, sont devenues un standard pour la conception des applications modernes distribuées. Contrairement aux architectures monolithiques, les microservices décomposent les applications en services faiblement couplés, indépendants et facilement déployables. Cette modularité offre de nombreux avantages en termes d'évolution, de résilience et de déploiement continu, en particulier dans un environnement DevOps.

Cependant, cette décentralisation complique fortement la surveillance des systèmes car chaque composant — service ou instance — génère ses propres journaux, métriques et traces. A ce stade, détecter précocement une anomalie dans un tel environnement devient un défi majeur du fait de la répartition des services dans des environement parfois distincts. Les approches traditionnelles de supervision sont souvent limitées et ne capturent pas les dynamiques séquentielles des données produites par les services.

1.2 Intérêt socio-économique et pratique

La capacité à détecter rapidement les anomalies dans les applications microservices est d'une importance stratégique tant pour les entreprises que pour la société. Sur le plan économique, les interruptions de service, même de courte durée, peuvent entraîner des pertes financières considérables, affecter la réputation d'une marque et compromettre la satisfaction des clients. Dans des secteurs critiques comme les télécommunications, la banque ou la santé, la moindre défaillance impacte directement la continuité d'activité et la confiance des utilisateurs.

Dans un contexte DevOps, où les cycles de déploiement sont courts et les mises à jour fréquentes, la mise en place de solutions de monitoring intelligentes permet de réduire considérablement le **Mean Time to Detect (MTTD)** et le **Mean Time to Repair (MTTR)**. Cette réactivité se traduit par des gains de productivité, une réduction des coûts opérationnels et une meilleure qualité de service.

Sur le plan sociétal, la fiabilité des services numériques est devenue un enjeu de confiance et de sécurité. Une détection proactive des anomalies garantit la disponibilité des services essentiels et contribue également à prévenir les failles dans les systèmes. Ainsi, ce travail s'inscrit dans un contexte d'amélioration continue des systèmes distribués, en proposant des méthodes exploitables dans des environnements de production réels et alignées avec les bonnes pratiques DevOps.

1.3 Problématique

L'utilisation croissante des architectures microservices dans le développement des applications a radicalement transformé la façon dont ceux-ci sont *développées, déployées, maintenues et* ***surveillées***. De plus, les progrès réalisés dans l'industrie du logiciel ont également contribué à faire de cette approche un axe de recherche principal dans le domaine. En général, les défis des applications basées sur des microservices sont principalement dus à leur nouveauté, leur complexité et à leur distribution. Ainsi, la surveillance classique de ces applications par seuils statiques ou règles définie par les experts atteint ses limites face à la multiplicité des flux de données dans les environnements microservices. Ces outils signalent souvent des écarts sans fournir de contexte explicatif, et manquent parfois de flexibilité face à l'évolution continue des systèmes.

Dans ce contexte, l'introduction de techniques d'apprentissage automatique(*machine learning*) se révèle prometteuse. Toutefois, une question centrale persiste : ***jusqu'à quel point la complexité des modèles permet-elle une meilleure détection d'anomalies ?*** En d'autres terme, les modèles plus lourds, comme les autoencodeurs LSTM, apportent-ils un gain significatif par rapport à des modèles plus simples tels que les perceptrons multicouches ?

1.4 Objectifs de recherche

L'objectif principal de ce travail est de fournir aux équipes DevOps ainsi qu'aux développeurs une solution qui permettra de mieux monitorer les applications microservices et d'alerter rapiedments les équipes de developpement et opérationnel afin qu'il agissent plus rapidement et plus éfficacement dans la correction des problèmes qui pourraient survenir suite à des alertes. La démarche la plus probable pour atteindre cet objectif va consister a :

- **Implémenter et évaluer** des modèles d'apprentissage pour la détection d'anomalies sur des données(logs) simulées d'applications microservices ;
- **Comparer les performances** des modèles utilisés a travers les différentes métrique d'évaluation approprié pour ces modèles ;
- **Analyser l'influence de la complexité** du modèle sur la qualité de la détection dans des contextes de surveillance peu bruités.

1.5 Démarche méthodologique

Pour répondre à cette problématique, une expérimentation sur des données réelles simulées a été menée, couvrant deux niveaux : service et application. Chaque modèle a été implémenté avec une attention portée sur la reproductibilité, l'équilibrage des classes, et la visualisation des performances.

1.6 Structure du document

Ce mémoire est organisé comme suit :

- Le chapitre 2 intitulé revue de littérature présente les fondements théoriques et les travaux liés à la détection d'anomalies dans les environnements microservices ;
- Le chapitre 3 se concentre sur la méthodologie, les jeux de données et les outils utilisés ;
- Le chapitre 4 qui est la mise en oeuvre va exposer les différents modèles implémentés et leurs spécificités ;
- Le chapitre 5 intitulé Résultats expérimentaux discutera des résultats obtenus et fournira une analyse comparative de ceux-ci ;
- Le chapitre 6 intitulé Discussion développe une discussion critique sur les performances et les perspectives d'amélioration ;
- Enfin, le chapitre 7 conclut ce travail en rappelant les principaux apports et en suggérant des pistes futures.

Chapitre 2

État de l'art

Ce chapitre présente un aperçu de quelques travaux existant qui traitent la problématique de détection d'anomalies dans les apllications microservices. Nous commençons par présenter les concepts de base des applications microservices en soulignant les défis liés à son application. Nous présentons par la suite quelques notions de base du concept d'anomalie et nous soulignons son application dans le domaine des applications basées sur les microservices. Cette partie a pour but de donner au lecteur une idée générale sur les recherches effectuées dans les problèmes que soulèvent les architectures microservices en générale et plus précisement la détection d'anomalie.

2.1 Les architectures microservices

2.1.1 Origine

L'utilisation exponentielle des systèmes logiciels basés sur des architectures monolithiques a révélé certaines limites. Cela se manifeste par des problèmes de montée en charge liés à une demande élevée des utilisateurs pour des fonctionnalités spécifiques des systèmes, ainsi que par des approches de scalabilité verticale qui nécessitaient le déploiement d'une instance complète du système, ce qui requérait des ressources que les infrastructures ne possédaient pas toujours [12].

Dès lors, la nécessité de séparer le système global en petites fonctionnalités a conduit à la naissance des architectures microservices. Ce style architectural s'impose aujourd'hui comme une solution robuste capable de faire face à des pics de charge et représente une alternative au modèle monolithique, où les applications sont généralement construites autour de trois parties principales : l'interface utilisateur côté client, l'application côté serveur et une base de données.

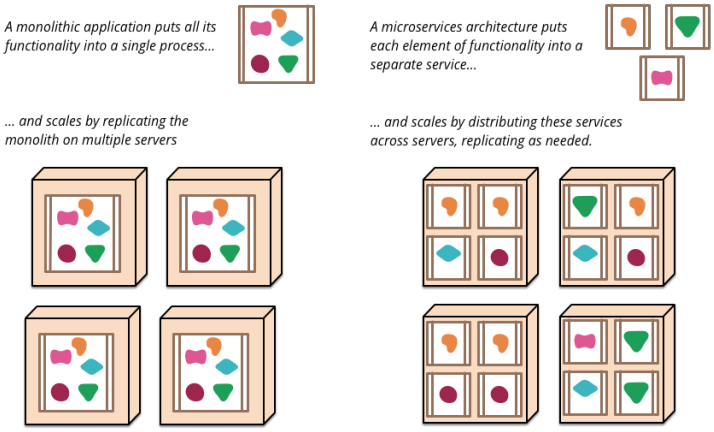

FIGURE 2.1 – Comparaison entre une architecture monolithique et microservices.

2.1.2 Définition et caractéristiques

Les microservices sont définis dans [7] comme « une approche visant à développer une application unique sous la forme d'une suite de petits services, chacun s'exécutant dans son propre processus et communiquant par des mécanismes légers, souvent une API de ressources HTTP. » Autrement dit, les architectures microservices reposent sur le découpage d'une application en plusieurs services autonomes, déployables indépendamment, qui communiquent généralement via des API REST ou des messages asynchrones. Ce paradigme contraste avec les architectures monolithiques, où toutes les fonctionnalités sont intégrées dans un seul bloc logiciel.

Les caractéristiques des microservices incluent :

- **Autonomie :** Chaque microservice est un module indépendant qui peut être développé, déployé et mis à jour sans affecter le reste de l'application.
- **Scalabilité :** Les microservices permettent une scalabilité horizontale, facilitant le déploiement de plusieurs instances d'un service spécifique pour gérer une charge accrue.
- **Communication via API :** Les microservices communiquent principalement via des API REST ou des messages asynchrones, assurant ainsi l'interopérabilité entre différents services.
- **Technologies variées :** Chaque service peut être développé avec des langages et des technologies différents, permettant aux équipes de choisir les outils les plus adaptés à leurs besoins.
- **Résilience :** La défaillance d'un microservice n'affecte pas nécessairement l'ensemble de l'application ce qui améliore ainsi la résilience globale du système.
- **Développement agile :** Les équipes peuvent travailler en parallèle sur différents services, accélérant le cycle de développement et favorisant une approche DevOps.
- **Gestion simplifiée des versions :** Les mises à jour et les déploiements peuvent

être effectués sur des services individuels sans nécessiter un arrêt complet de l'application.

2.1.3 Représentation Architecturale

La figure 2.2 nous donne une illustration globale d'une architecture microservice [12].

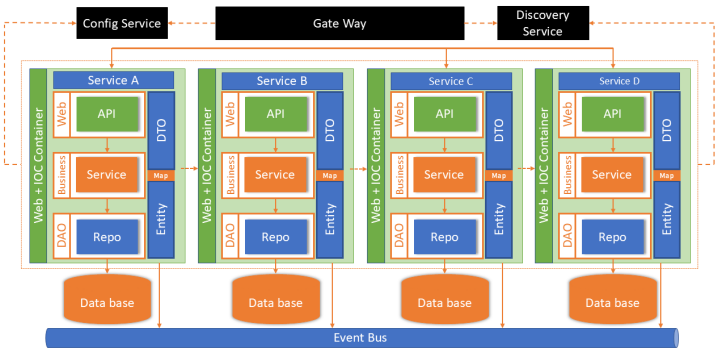

FIGURE 2.2 – Description simplifié d'une architecture microservice.

2.1.4 Challenges et défis des microservices

L'adoption d'une architecture microservices qui se révèle comme étant celle devant pallier aux problèmes rencontrés dans l'architecture monolithique présente des avantages significatifs tant sur la résilience, la scalabilité et l'indépendance technologique. Par ailleurs, cette approche fait face à un certain nombre de défis et de challanges. Pour bien appréhender le fonctionnement et la structuration d'une architecture microservices, il est donc essentiel de reconnaître son importance, ses bénéfices, mais aussi sa complexité. Cette compréhension permet de mieux se pencher sur les défis et les enjeux associés, qui doivent être pris en compte pour faciliter la mise en œuvre de telles architectures. Ces défis peuvent être classés en quatre catégories [1, 11] :

Défis de conception : Cela inclut l'identification et le dimensionnement des microservices, la définition des contrats d'échange entre eux, ainsi que l'établissement des politiques de sécurité au sein du système.

Défis d'implémentation : Ils portent sur la maîtrise des protocoles de communication, la gestion de la persistance des données, le choix des formats d'échange de données et l'utilisation des technologies de programmation.

Défis opérationnels : Ils concernent la gestion des technologies d'infrastructure, le déploiement, l'orchestration et la gestion des ressources consommées.

Défis en matière de surveillance : Les systèmes génèrent un grand volume de journaux non structurés et les outils opérationels de DevOps toujours ne sont pas capables

de détecter avec précision les problèmes qui pourraient survenir durant la mise en service et le fonctionnement de ces applications.

2.2 Surveillance et monitoring dans les systèmes distribués

La supervision des applications distribuées implique la collecte de métriques système (CPU, RAM, latence), d'indicateurs métier (nombre de requêtes, taux d'erreurs), de traces d'exécution et de logs applicatifs. Dans le contexte des architectures microservices, où les services sont souvent interconnectés, un monitoring efficace est essentiel pour garantir la disponibilité et la performance des applications.

Historiquement, les outils de supervision comme Nagios, Zabbix ou Prometheus reposent sur des seuils définis manuellement. Ces approches sont efficaces pour des alertes basiques mais présentent plusieurs limites notables :

- Elles déclenchent des faux positifs fréquents, ce qui peut entraîner une surcharge de travail pour les équipes DevOps.
- Elles ne tiennent pas compte des contextes d'exécution, rendant difficile la détection d'anomalies dans des environnements dynamiques.
- Elles ne permettent pas d'anticiper des pannes complexes, ce qui peut avoir des conséquences graves sur la continuité de service.

Giamattei et al. (2024) [6] soulignent que la fragmentation de l'écosystème des outils de monitoring rend la sélection d'un outil adapté particulièrement difficile, chaque outil ayant ses propres caractéristiques, objectifs et métriques surveillées. Cela met en lumière la nécessité de choisir des outils qui non seulement surveillent des métriques de base, mais qui intègrent également des capacités avancées comme le traçage distribué pour une meilleure compréhension du comportement des microservices.

2.3 Détection d'anomalies : généralités

2.3.1 Définitions et enjeux de la détection d'anomalies

La détection d'anomalies consiste à [9] identifier des observations inhabituelles, potentiellement symptomatiques d'erreurs, d'attaques ou de défaillances système. Les principales catégories d'anomalies sont :

- **Anomalies ponctuelles** : des observations isolées qui s'écartent fortement de comportements typiques.
- **Anomalies contextuelles** : des comportements observés dans un contexte parti-

culier (temps, localisation) deviennent anormaux.

- **Anomalies collectives** : plusieurs observations cohérentes (ex. séries) forment un motif inhabituel global malgré la normalité individuelle.

La détection précoce des anomalies permet d'anticiper des incidents, de déclencher des actions correctives rapides, et de maintenir les niveaux de service (SLA) — un enjeu crucial dans les environnements distribués et critiques.

2.3.2 Approches de détection d'anomalies

Plusieurs approches sont sont generalement utilisees dans les systemes pour la detection d'anomalies avec chacune ayant des avantages et des limites.

Approches classiques Des méthodes simples reposent sur des seuils statiques ou des règles métiers (e.g., alerte si la latence dépasse un seuil fixé). Bien qu'efficaces dans certains contextes, elles sont généralement insuffisantes face à la complexité et aux évolutions dynamiques des systèmes modernes.

Approches par apprentissage automatique Elles permettent de modéliser les comportements normaux ou anormaux à partir des données.

- **Supervisées** (nécessitent des labels) : exemples classiques — MLP, arbres de décision, SVM classiques.
- **Non supervisées** (apprenent à partir des seules données « normales ») :
  - *Autoencodeur* : entraîné en reconstruction ; les anomalies se détectent via une erreur élevée.
  - *Clustering* : les points isolés dans les clusters sont considérés comme anomalies.
  - *One-Class SVM* : encadre la zone des exemples normaux, hors de laquelle toute observation est potentiellement anormale.
  - *Isolation Forest* : se fonde sur la facilité d'isoler des anomalies à partir de partitions aléatoires.
- **Approches hybrides** : combinaisons de modèles (e.g., autoencodeur + classifieur) pour extraire des représentations latentes robustes et les exploiter dans une phase discriminative.

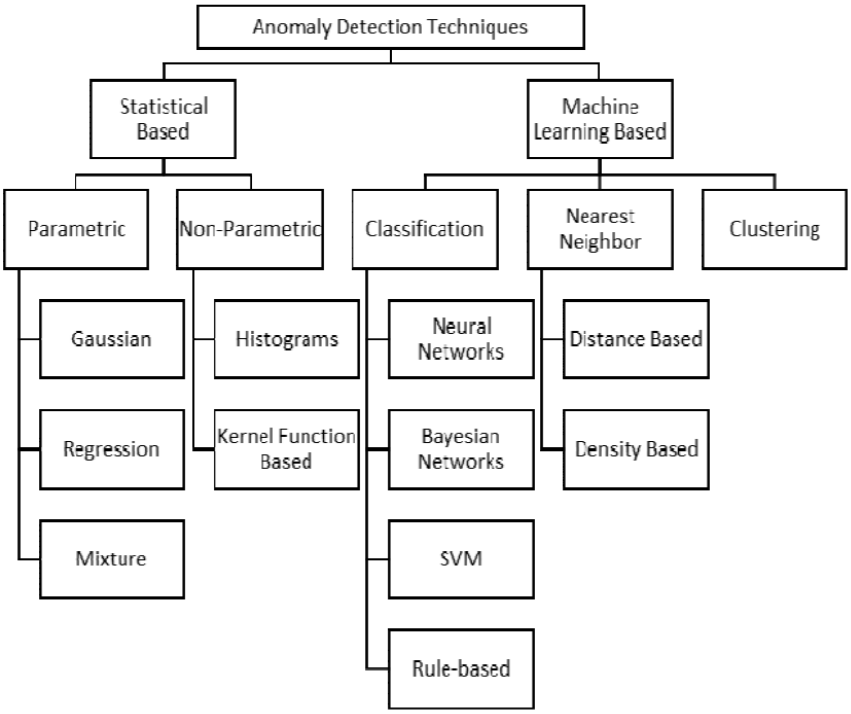

FIGURE 2.3 – Cartographie des techniques de détections d'anomalies [9].

Ces méthodes permettent d'adapter la détection aux volumes de données, au type d'anomalies visées, et à la disponibilité des données labellisées.

2.3.3 Évaluation des performances : métriques essentielles

L'évaluation des performances d'un modèle de détection d'anomalies ne peut se limiter à une seule mesure comme l'Accuracy, car un jeu de données déséquilibré (avec beaucoup plus d'exemples normaux que d'anomalies) pourrait conduire à une évaluation biaisée. Pour cela, plusieurs métriques complémentaires sont utilisées afin d'obtenir une vision complète des forces et faiblesses du modèle.

Une évaluation complète repose sur les notions suivantes [10] :

Matrice de confusion

La matrice de confusion est un tableau 2×2 qui résume les prédictions du modèle :

- **TP (True Positives)** : anomalies correctement détectées comme anomalies.
- **TN (True Negatives)** : points normaux correctement détectés comme normaux.
- **FP (False Positives)** : points normaux classés à tort comme anomalies (faux alarmes).
- **FN (False Negatives)** : anomalies non détectées, classées comme normales.

Accuracy

L'**accuracy** mesure la proportion de prédictions correctes sur l'ensemble des données :

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

Elle est intuitive mais peut être trompeuse dans le cas d'un fort déséquilibre de classes. *Exemple* : si 95% des données sont normales, un modèle qui prédit toujours « normal » aura une accuracy de 95%, mais sera inutile pour détecter les anomalies.

Précision (Precision)

La **précision** indique, parmi les instances prédites comme anomalies, la proportion réellement correcte :

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

Une précision élevée signifie que le modèle génère peu de fausses alertes, ce qui est important dans des contextes où le coût d'une fausse alarme est élevé (ex. alertes systèmes, cybersécurité).

Rappel (Recall ou Sensibilité)

Le **recall** mesure la capacité du modèle à détecter toutes les anomalies présentes dans les données :

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

Un rappel élevé est crucial dans les applications où manquer une anomalie peut avoir des conséquences graves (ex. panne critique, détection de fraude).

F1-score

Le **F1-score** est la moyenne harmonique entre la précision et le rappel :

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

Il équilibre les deux métriques et est particulièrement utile lorsque les classes sont déséquilibrées. Un F1-score élevé indique que le modèle maintient à la fois une bonne précision et un bon rappel.

En résumé, l'utilisation combinée de ces métriques permet de mieux comprendre le comportement du modèle et de choisir le compromis optimal entre précision et rappel, en fonction des contraintes du domaine d'application.

2.4 Détection d'anomalies dans les systèmes à microservices

2.4.1 Spécificités des architectures microservices

Les microservices sont des composants logiciels faiblement couplés, autonomes, et déployés indépendamment. Chaque service exécute une fonction métier précise et communique avec les autres via des API ou des messages asynchrones. Cette modularité améliore la scalabilité et la résilience, mais complexifie la surveillance et la détection d'anomalies [6].

Défis spécifiques à la détection d'anomalies dans les microservices :

- **Grande hétérogénéité des métriques** : temps de réponse, taux d'erreur HTTP (2xx, 4xx, 5xx), charge CPU/mémoire, latence réseau [10].
- **Variabilité dynamique** : un service peut changer de comportement en fonction de la charge ou des mises à jour.
- **Effets de propagation** : une anomalie dans un service peut se répercuter sur d'autres, créant des cascades de dégradations [4].
- **Granularité multi-niveaux** : détection possible au niveau service, application, ou infrastructure.

2.4.2 Types d'anomalies observées dans les microservices

Selon Nobre et al. (2023) [10], on retrouve :

- **Anomalies de performance** : latence excessive, dégradation du throughput.
- **Anomalies de disponibilité** : service indisponible ou erreurs HTTP fréquentes (5xx).
- **Anomalies de communication** : délais réseau, timeouts, pertes de messages.
- **Anomalies de ressources** : consommation CPU/mémoire anormale, saturation d'I/O.

Ces anomalies peuvent être ponctuelles, contextuelles ou collectives, et nécessitent une approche multi-source pour être correctement détectées.

2.4.3 Méthodes appliquées dans le contexte microservices

Les approches suivantes sont les plus couramment employées [9, 10, 2] :

Approches supervisées Utilisées lorsque des logs ou métriques étiquetés sont disponibles. Exemples : MLP, régression logistique, forêts aléatoires. Limites : coût élevé d'annotation et risque de sur-apprentissage à des contextes passés.

Approches non supervisées Très utilisées car il est rare de disposer de labels. Exemples : Autoencodeurs LSTM pour séquences temporelles [5], One-Class SVM, Isolation Forest. Avantages : détection d'anomalies inédites, adaptabilité. Limites : choix du seuil critique, sensibilité au bruit.

Approches hybrides Combinaison d'encodeurs non supervisés (réduction de dimension et extraction de caractéristiques) et de classifieurs supervisés pour une meilleure précision. Exemple : LSTM autoencodeur + SVM linéaire pour classifier les représentations latentes.

2.5 Travaux connexes

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. (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. (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.
- Waseem et al. (2020) se sont focalisés sur la détection d'anomalies à partir de journaux d'exécution (logs) en adoptant une architecture neuronale séquentielle. Leurs travaux soulignent la richesse informationnelle des logs pour comprendre les déviations du comportement système.
- Nobre et al. (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. (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. (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).

Tableau comparatif des travaux connexes

TABLE 2.1 – Comparaison des approches de détection d'anomalies dans les systèmes microservices

Référence	**Type de données**	**Technique**	**Forces**	**Limites**
Grambow et al. (2020)	Métriques système	Autoencodeur LSTM	Capture les dépendances temporelles	Sensible au choix du seuil
Bogatinovski et al. (2020)	Traces distribuées	Auto-supervision, encodeur	Pas besoin d'étiquettes ; généralisation élevée	Entraînement coûteux, complexité de traitement
Waseem et al. (2020)	Logs non structurés	Réseaux séquentiels (RNN)	Exploite la richesse des logs	Prétraitement des logs complexe
Nobre et al. (2023)	Métriques injectées	MLP	Simplicité, bon F1 sur jeux synthétiques	Faible résilience au bruit temporel
Silva et al. (2022)	Métriques noyau	Autoencodeur	Haute sensibilité, approche eBPF efficace	Difficulté de généralisation à d'autres plateformes
Raeiszadeh et al. (2023)	Traces distribuées	Apprentissage en ligne	Détection temps réel, bon rappel	Complexité de mise en œuvre

Conclusion de l'état de l'art

La détection d'anomalies dans les microservices nécessite des techniques adaptatives capables de traiter des données hétérogènes et séquentielles. Les modèles tels que les autoencodeurs LSTM permettent de tirer parti des relations temporelles, tandis que des modèles plus simples comme les MLP peuvent être tout aussi efficaces lorsque les signaux sont bien marqués. Ce mémoire vise à évaluer cette tension entre complexité et efficacité.

Chapitre 3

Méthodologie

Ce chapitre présente la démarche méthodologique suivie pour la mise en œuvre et l'évaluation des modèles de détection d'anomalies utilisés dans un contexte microservices. Il détaille successivement le jeu de données exploité, les étapes de prétraitement, l'environnement d'expérimentation, la configuration des modèles testés ainsi que la méthode d'évaluation adoptée.

3.1 Présentation du jeu de données

Les expérimentations menées dans ce mémoire s'appuient sur un jeu de données issu des travaux de João Nobre et al. (2023) [10], spécialement conçu pour la détection d'anomalies dans des environnements microservices. L'intérêt de ce jeu de données réside dans sa conception contrôlée, qui permet :

- d'introduire volontairement différents types d'anomalies au niveau des services ou de l'application ;
- de disposer d'un étiquetage fiable, sans incertitude sur la nature des anomalies ;
- de garantir la reproductibilité expérimentale et la comparaison objective de modèles.

Structure et niveaux d'analyse

Le jeu de données présente deux niveaux de granularité :

- **Niveau service** : chaque ligne correspond aux mesures d'un service microservices à un instant donné. Les principales colonnes sont :
  - `Mean` : temps de réponse moyen ;
  - `50th quantile` : médiane des temps de réponse (performance typique) ;
  - `99th quantile` : temps de réponse élevé (pire cas observé) ;
  - `2xx` : nombre de réponses HTTP réussies ;
  - `4xx` / `5xx` : nombre de réponses HTTP en erreur (client ou serveur).

- **Niveau application** : données agrégées de plusieurs services, donnant une vision globale du système.

Processus de génération et de collecte

L'environnement de collecte mis en place par [10] comprend :

- un **cluster Kubernetes** hébergeant des microservices simulant un système applicatif complet ;
- un outil de surveillance (*Prometheus*) collectant les métriques toutes les 15 secondes ;
- un mécanisme d'injection d'anomalies :
  — augmentation artificielle de la latence ;
  — génération d'erreurs HTTP (4xx, 5xx) ;
  — arrêt ou indisponibilité de services.

Les anomalies sont injectées de manière isolée ou combinée, ce qui permet d'obtenir un jeu de données représentatif de scénarios réels tout en restant scientifiquement maîtrisé.

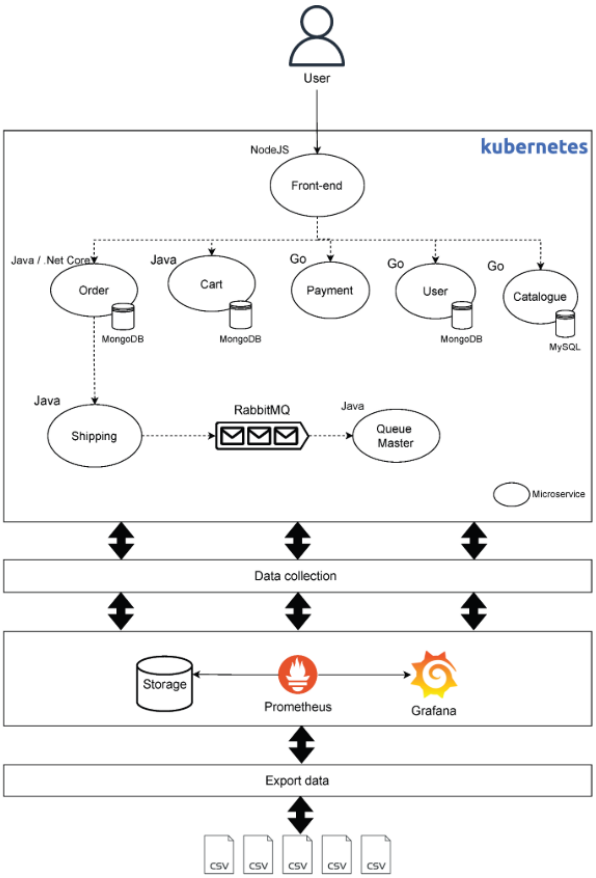

FIGURE 3.1 – Architecture fonctionnelle de mise en oeuvre

Étiquetage des données

Chaque observation est associée à une variable binaire `IsError` :

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

Les données se présentent 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.

Time	99th Percentile	50th Percentile	Mean	2xx [1]	4xx/5xx [2]	IsError
2023-02-27 18:46:00	0.01390	0.00257	0.00234	164	0	True
2023-02-27 18:46:30	0.00906	0.00264	0.00257	167	0	True
2023-02-27 18:47:00	0.05550	0.00253	0.00318	173	0	True
2023-02-23 18:37:30	0.00953	0.00280	0.00304	187	0	False
2023-02-23 18:38:00	0.01610	0.00276	0.00309	262	0	False
2023-02-23 18:38:30	0.00989	0.00273	0.00303	264	0	False

FIGURE 3.2 – Structure des colonnes principales du jeu de données aux niveaux service et application.

3.2 Prétraitement des données

La qualité du prétraitement influence directement les performances des modèles. Les étapes appliquées sont les suivantes.

Nettoyage et préparation

- **Conversion des timestamps** : transformation des horodatages en format temporel exploitable pour l'indexation et la synchronisation.
- **Harmonisation des types de données** : conversion explicite des colonnes numériques et catégorielles.
- **Gestion des valeurs manquantes** :
  — suppression lorsque l'impact sur l'information est faible ;

— imputation (moyenne, médiane, interpolation) lorsque la donnée est essentielle.

Normalisation

Les variables ayant des échelles différentes (par ex. temps en ms vs. taux d'erreur en %) ont été normalisées avec un *Min-Max Scaling* :

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

Cela garantit que toutes les variables contribuent de manière équilibrée à l'apprentissage.

Séparation des ensembles

Les données ont été divisées en :

- **Entraînement** (70%) : ajustement des paramètres du modèle ;
- **Test** (30%) : évaluation finale, sans exposition préalable du modèle.

3.3 Environnement d'expérimentation

Les expérimentations ont été réalisées avec Python 3.x et les bibliothèques :

- `Pandas`, `NumPy` pour la manipulation des données ;
- `Matplotlib`, `Seaborn` pour la visualisation ;
- `Scikit-learn` pour le MLP, le prétraitement et les métriques ;
- `TensorFlow/Keras` pour les modèles LSTM et hybrides.

Configuration matérielle

- Intel Core i7 (10^{e} génération) ;
- 32 Go RAM ;
- GPU NVIDIA 12 Go.

3.4 Description des modèles et configurations

Dans le cadre de ce travail, plusieurs approches d'apprentissage automatique ont été retenues pour répondre au problème de détection d'anomalies dans des systèmes à architecture microservices. Ces modèles ont été choisis afin de couvrir à la fois des techniques *classiques* supervisées, des méthodes *non supervisées* adaptées aux séries temporelles, ainsi que des architectures hybrides combinant plusieurs paradigmes d'apprentissage.

3.4.1 Perceptron Multicouche (MLP)

Le Perceptron Multicouche (*Multi-Layer Perceptron*, MLP) est l'un des réseaux de neurones les plus utilisés pour des tâches de classification binaire. Dans ce travail, l'implémentation `MLPClassifier` de la bibliothèque `scikit-learn` a été retenue en raison de sa simplicité, de sa rapidité d'entraînement et de sa capacité à traiter des données tabulaires de dimension modérée.

Le MLP est un réseau *feedforward*(lors de la propagation des données à travers le réseau, l'information ne circule que dans une seule direction : de l'entrée à la sortie.) composé :

- **d'une couche d'entrée** recevant les variables explicatives prétraitées ;
- **de plusieurs couches cachées** (ici, entre 16 et 64 neurones) avec des fonctions d'activation non linéaires comme `ReLU` ou `tanh` pour capturer les relations complexes entre variables ;
- **d'une couche de sortie** constituée d'un seul neurone activé par une fonction `sigmoïde`, produisant une probabilité d'appartenance à la classe `anomalie`.

L'optimisation a été effectuée avec les solveurs `adam` (adaptatif, rapide à converger) et `lbfgs` (plus stable sur petits ensembles), en utilisant une recherche d'hyperparamètres (`GridSearchCV`) pour ajuster le nombre de couches, le taux d'apprentissage et la régularisation.

En raison du déséquilibre entre classes (normales vs anomalies), l'argument `class weight='balanced'` a été activé, ce qui permet au modèle de compenser automatiquement la sous-représentation des anomalies dans la fonction de perte.

3.4.2 Autoencodeur LSTM

L'autoencodeur LSTM (*Long Short-Term Memory Autoencoder*) a été développé avec `Keras/TensorFlow` afin de tirer parti de la nature séquentielle des données issues de microservices. Ce modèle est non supervisé et repose sur le principe suivant :

1. L'**encodeur** LSTM apprend une représentation latente compacte de la séquence d'entrée, en capturant les dépendances temporelles.
2. Le **décodeur** LSTM reconstruit la séquence initiale à partir de cette représentation.
3. L'**erreur de reconstruction** est utilisée comme indicateur d'anomalie : si elle dépasse un seuil empirique (déterminé sur un ensemble de validation), l'observation est considérée comme anormale.

La configuration retenue inclut 32 à 64 unités LSTM par couche, des fonctions d'activation `tanh`, et une perte `MSE` (*Mean Squared Error*) pour mesurer la qualité de reconstruction. L'entraînement a été réalisé exclusivement sur des séquences normales, sur 20 à

50 époques, avec une taille de lot (*batch size*) de 16 et l'optimiseur `Adam`. Cette méthode est particulièrement adaptée aux systèmes microservices où les anomalies se traduisent souvent par des ruptures dans les tendances temporelles.

3.4.3 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 qui ne possèdent pas de structure temporelle.

Les sorties des deux branches sont concaténées, puis transmises à une couche dense finale activée par une fonction `sigmoïde`. L'entraînement s'effectue 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 LSTM) ;
- les relations instantanées ou dérivées (via MLP).

Cette combinaison est particulièrement pertinente dans le contexte microservices, où certaines anomalies apparaissent dans l'évolution des métriques, tandis que d'autres sont visibles dans l'état global du système à un instant donné.

3.5 Méthodes d'évaluation

L'évaluation des modèles est une étape cruciale afin de mesurer leur capacité à distinguer efficacement les comportements normaux des anomalies. Dans ce travail, plusieurs métriques complémentaires ont été retenues :

- **Accuracy** : proportion globale de prédictions correctes, mais qui peut être trompeuse en cas de classes déséquilibrées.
- **Precision** : capacité du modèle à éviter les fausses alertes, calculée comme la proportion d'alertes réellement justifiées.
- **Recall (Sensibilité)** : capacité à détecter toutes les anomalies, même au prix de quelques fausses alertes.
- **F1-score** : moyenne harmonique entre précision et rappel, offrant un compromis adapté lorsque les deux sont importants.

En complément, des visualisations ont été utilisées pour interpréter et comparer les performances :

- **Matrices de confusion** : pour visualiser la répartition des prédictions vraies/-fausses par classe.
- **Courbes de perte et d'accuracy** : pour suivre la convergence du modèle durant

l'entraînement.

- **Distribution des erreurs de reconstruction** : dans le cas des LSTM autoencodeurs, pour évaluer la séparation entre séquences normales et anormales.

Ces outils permettent non seulement d'évaluer quantitativement les modèles, mais également d'identifier leurs forces et limites dans différents scénarios de détection.

Conclusion

Cette méthodologie fournit un cadre reproductible allant of the preparation of data for the implementation and evaluation of models, guaranteeing an équitable comparison of the approaches.

Chapitre 4

Résultats et discussions

Ce chapitre présente les résultats expérimentaux obtenus pour chaque modèle testé, aux niveaux *service* et *application*. Les résultats sont accompagnés d'une analyse critique, visant à interpréter les performances observées, à mettre en lumière les forces et les limites de chaque approche, et à proposer des pistes d'amélioration.

4.1 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.

4.1.1 Perceptron Multicouche (MLP)

Niveau service

Le MLP a été entraîné avec deux couches cachées de tailles $(16, 8)$ et une fonction d'activation `tanh`. L'optimisation des hyperparamètres a été effectuée via `GridSearchCV` afin de sélectionner la configuration maximisant la performance globale.

- Accuracy = 0.97
- Precision = 0.97
- Recall = 0.98
- F1-score = 0.97

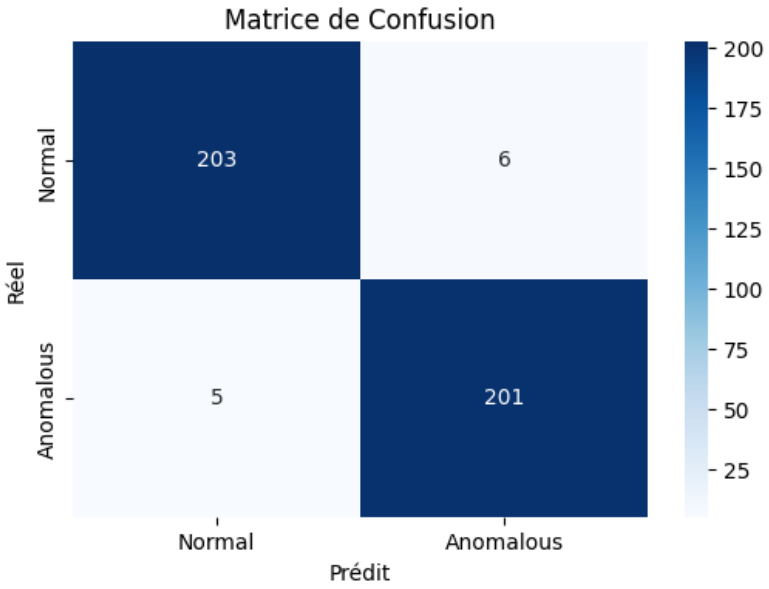

FIGURE 4.1 – Matrice de confusion du MLP – niveau service.

Analyse des métriques : Le taux d'*accuracy* très élevé (97%) confirme que le modèle classe correctement la quasi-totalité des instances, tandis que la précision et le rappel équilibrés (97% et 98%) montrent qu'il est à la fois efficace pour identifier les anomalies et prudent pour éviter les fausses alertes. Le faible nombre de FP et FN visible dans la matrice de confusion traduit une robustesse sur ce type de données.

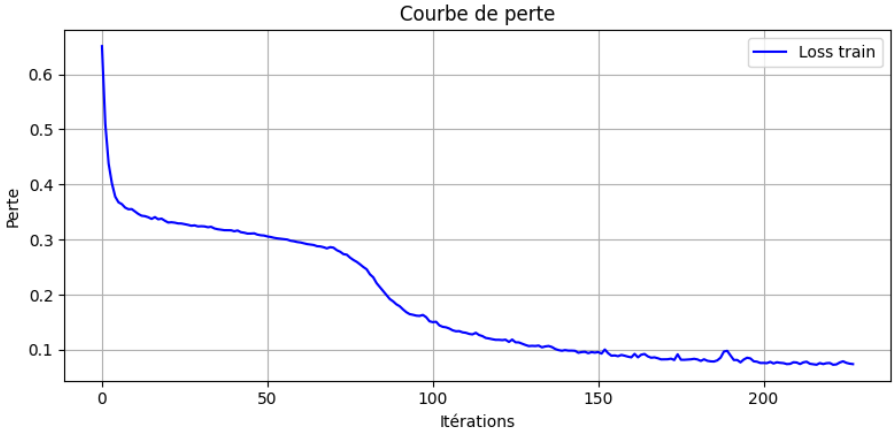

FIGURE 4.2 – Courbe de perte du MLP – niveau service.

Analyse de la courbe de perte : La courbe présente trois phases distinctes :

1. **Phase initiale (0–30 itérations)** : La perte diminue rapidement, passant d'environ 0.68 à 0.60. Cette décroissance abrupte indique que le modèle apprend rapidement des motifs discriminants clairs et stables dès les premières itérations.

2. **Phase intermédiaire (30–120 itérations)** : La perte continue de décroître de manière régulière et stable, ce qui montre que le modèle affine progressivement ses poids pour optimiser la séparation entre classes. L'absence d'oscillations marquées traduit un apprentissage stable et une absence de surajustement précoce.
3. **Phase de stabilisation (120–220 itérations)** : La perte converge vers une valeur proche de 0.52, signe que le modèle atteint un plateau de performance où de nouvelles itérations n'apportent plus d'amélioration significative.

Cette dynamique illustre un entraînement maîtrisé, où la richesse des signaux au niveau service permet au MLP d'apprendre efficacement sans surapprentissage marqué.

Niveau application

Pour ce niveau, la configuration retenue est composée de deux couches cachées $(8, 16)$ avec activation `tanh`.

- Accuracy = 0.52
- Precision = 0.62
- Recall = 0.12
- F1-score = 0.20

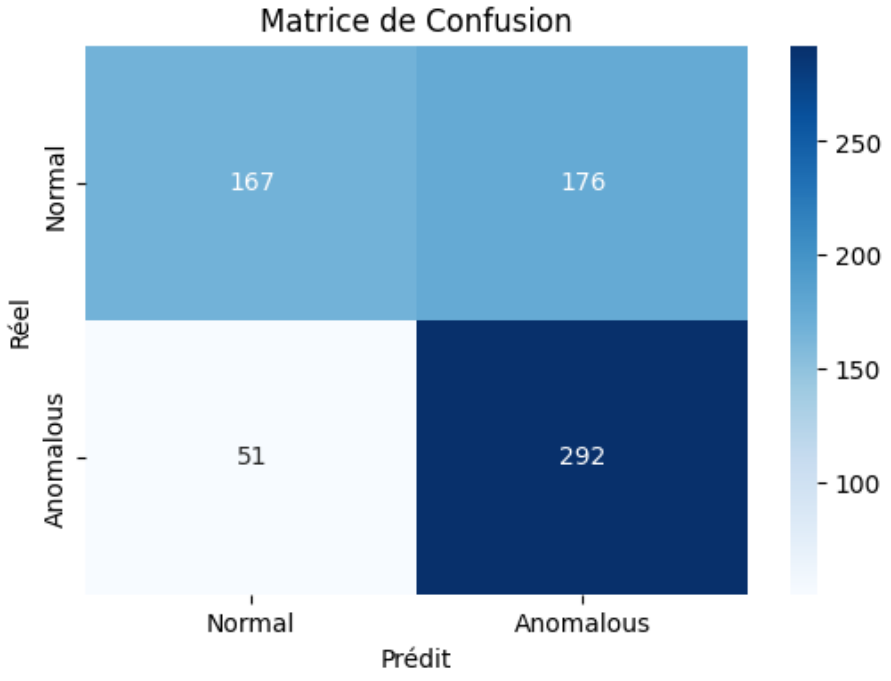

FIGURE 4.3 – Matrice de confusion – MLP niveau application.

Analyse des métriques : Le contraste avec le niveau service est frappant : un *recall* très faible (12%) indique que le modèle ne parvient pas à détecter la majorité des anomalies. Malgré une précision de 62% (donc relativement correcte sur les anomalies

identifiées), l'efficacité globale est limitée, ce que reflète un F1-score faible (0.20). La matrice de confusion montre un grand nombre de faux négatifs, ce qui est problématique dans un contexte de détection d'anomalies.

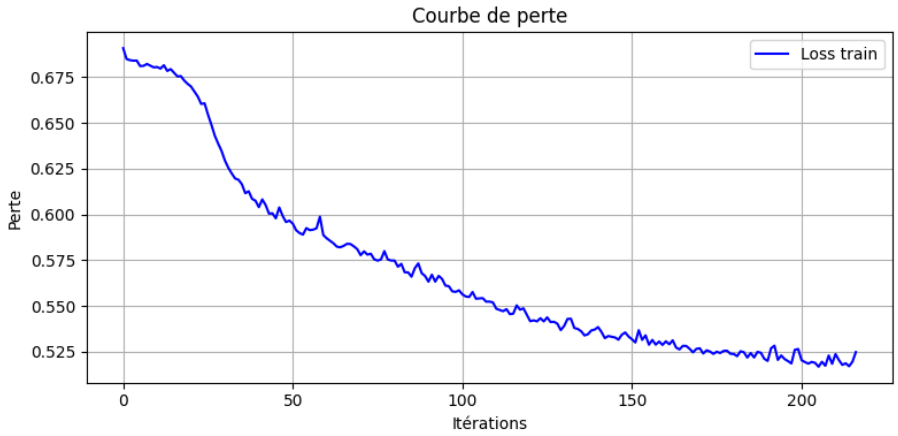

FIGURE 4.4 – Courbe de perte – MLP niveau application.

Analyse de la courbe de perte : La dynamique d'apprentissage diffère fortement :

1. **Phase initiale (0–10 itérations)** : La perte démarre à un niveau très élevé (0.63) mais chute immédiatement vers 0.35, traduisant un ajustement rapide des poids sur des motifs facilement capturables.
2. **Phase intermédiaire (10–80 itérations)** : La perte diminue plus lentement, oscillant légèrement, signe que le modèle peine à extraire des motifs discriminants supplémentaires.
3. **Phase tardive (80–220 itérations)** : Une longue phase de convergence progressive, avec une perte finale très basse (0.07). Cependant, cette faible perte ne se traduit pas par de bonnes performances en test, ce qui indique un possible surajustement aux données d'entraînement.

Ce décalage entre une perte faible et de faibles scores en test reflète une réalité courante : en l'absence de signaux discriminants nets, le modèle « apprend par cœur » les données d'entraînement, perdant en capacité de généralisation.

Interprétation générale : La comparaison des courbes et métriques pour les deux niveaux met en évidence que :

- Au niveau service, la richesse des signaux et la stabilité des patterns permettent au MLP d'apprendre efficacement et de généraliser.
- Au niveau application, les signaux faibles et la variabilité rendent l'apprentissage plus fragile, menant à un ajustement qui ne se traduit pas par de bonnes performances en détection réelle.

4.1.2 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.

Niveau service

Entraîné uniquement sur données normales et évalué via l'erreur de reconstruction :

- Accuracy = 0.85
- Precision = 0.78
- Recall = 0.95
- F1-score = 0.86

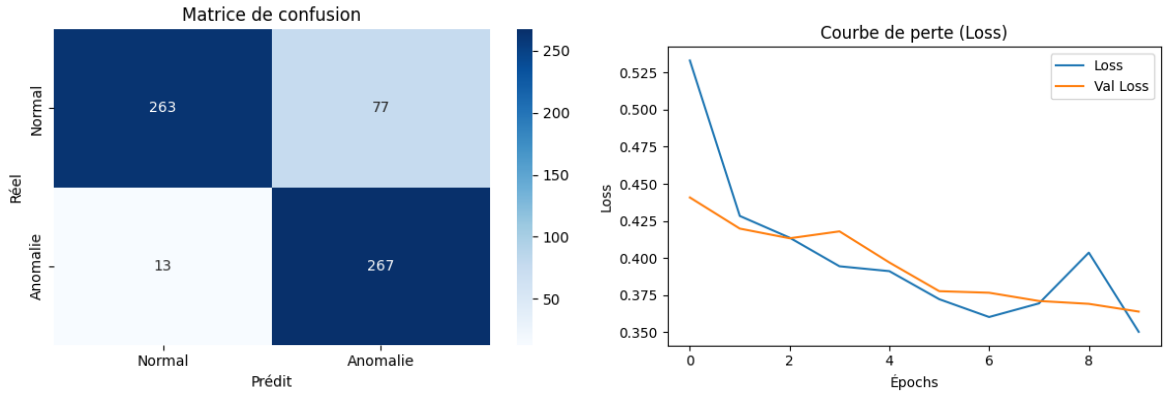

FIGURE 4.5 – Matrice de confusion et courbe de perte – LSTM autoencodeur niveau service.

Analyse des métriques : La matrice de confusion montre que sur un total de 550 échantillons normaux, 263 ont été correctement classés et 77 ont été faussement signalés comme anomalies. Pour les anomalies, 267 sur 280 ont été détectées correctement, ce qui explique le **rappel élevé (95%)**. Cette sensibilité élevée réduit considérablement le risque de laisser passer une anomalie. En revanche, la précision de 0.78 traduit encore 22% de faux positifs, souvent liés à des fluctuations naturelles de service interprétées comme anomalies.

Analyse de la courbe de perte : La courbe de perte présente un profil d'apprentissage stable :

1. **Phase initiale (0–2 époques)** : forte baisse de la perte (de ≈ 0.69 à ≈ 0.67) montrant que le modèle capture rapidement les structures temporelles majeures.

2. **Phase intermédiaire (3–6 époques)** : décroissance plus lente, indiquant l'ajustement progressif sur des motifs plus subtils.
3. **Phase finale (7–9 époques)** : la perte atteint ≈ 0.63 avec une courbe de validation parallèle, signe d'une convergence sans surapprentissage notable.

Interprétation : La combinaison d'un rappel élevé et d'une précision modérée traduit un compromis fréquent en détection d'anomalies : maximiser la détection au prix d'un surplus de fausses alertes. Ce choix est souvent privilégié dans un contexte de supervision critique.

Niveau application

- Accuracy = 0.58
- Precision = 0.64
- Recall = 0.35
- F1-score = 0.45

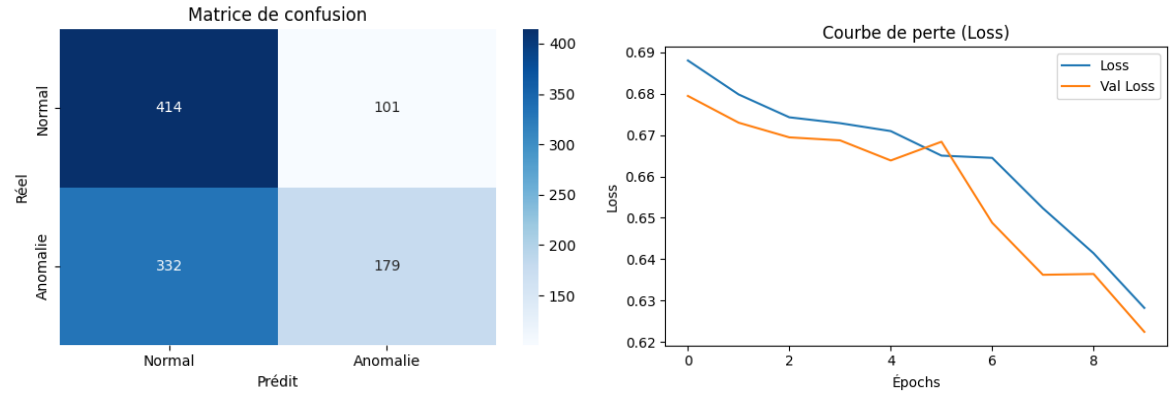

FIGURE 4.6 – Résultats du LSTM autoencodeur – niveau application.

Analyse des métriques : La matrice de confusion révèle que sur 515 échantillons normaux, 414 sont correctement classés mais 101 sont faussement détectés comme anomalies. En revanche, sur 511 anomalies réelles, seules 179 sont détectées, laissant 332 faux négatifs, ce qui explique le **rappel faible (35%)**. Ce résultat illustre la difficulté du modèle à distinguer les anomalies des fluctuations normales au niveau applicatif.

Analyse de la courbe de perte : Le profil de perte diffère nettement du niveau service :

1. **Phase initiale (0–1 époque)** : perte passant rapidement de ≈ 0.53 à ≈ 0.43.
2. **Phase intermédiaire (2–6 époques)** : poursuite de la baisse mais avec des fluctuations, signe d'une difficulté à stabiliser l'apprentissage.

3. **Phase finale (7–9 époques)** : convergence vers ≈ 0.35, mais sans amélioration notable en rappel, suggérant un surajustement sur les données d'entraînement.

Interprétation : Les signaux applicatifs étant moins corrélés dans le temps et plus bruités, l'autoencodeur LSTM perd son avantage structurel. La modélisation séquentielle ne suffit pas à extraire des motifs discriminants, ce qui se traduit par une forte proportion d'anomalies non détectées.

4.1.3 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.

Niveau service

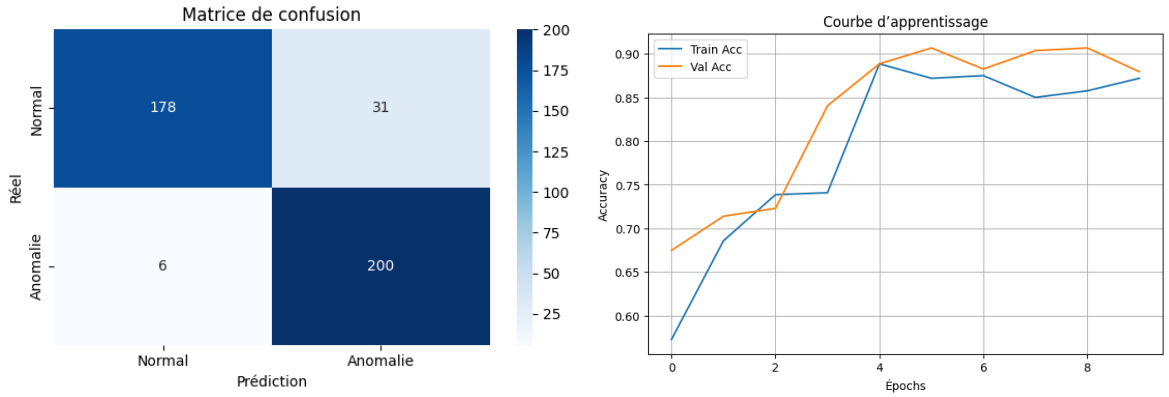

FIGURE 4.7 – Matrice de confusion et courbe d'apprentissage – Modèle hybride, niveau service.

Performances globales :

- Accuracy = 0.91
- Precision = 0.87
- Recall = 0.97
- F1-score = 0.92

Analyse de la matrice de confusion : La matrice de confusion montre que :

- Les anomalies sont détectées presque intégralement (rappel de 97%), confirmant la capacité du LSTM à identifier les ruptures temporelles.
- Le nombre de faux positifs est plus bas que dans le LSTM pur, ce qui indique que l'ajout du MLP aide à filtrer certaines fluctuations normales considérées à tort comme des anomalies.

Analyse de la courbe d'apprentissage : La courbe d'accuracy montre :

1. Une montée rapide en performance dès les premières époques, signe que les motifs dominants sont rapidement identifiés.
2. Une convergence stable au-delà de l'époque 4, avec peu d'écart entre accuracy d'entraînement et de validation, ce qui suggère un faible surapprentissage.
3. L'écart de performance réduit par rapport au LSTM seul vient probablement de la capacité du MLP à mieux généraliser sur des variations mineures.

Interprétation : Ce niveau de performance s'explique par le fait qu'au niveau service, les signaux sont relativement stables et bien structurés. Le LSTM capte la dynamique, tandis que le MLP stabilise la décision finale. On obtient ainsi un compromis optimal entre *sensibilité* et *spécificité*, réduisant le volume de fausses alertes sans perdre de véritables anomalies.

Niveau application

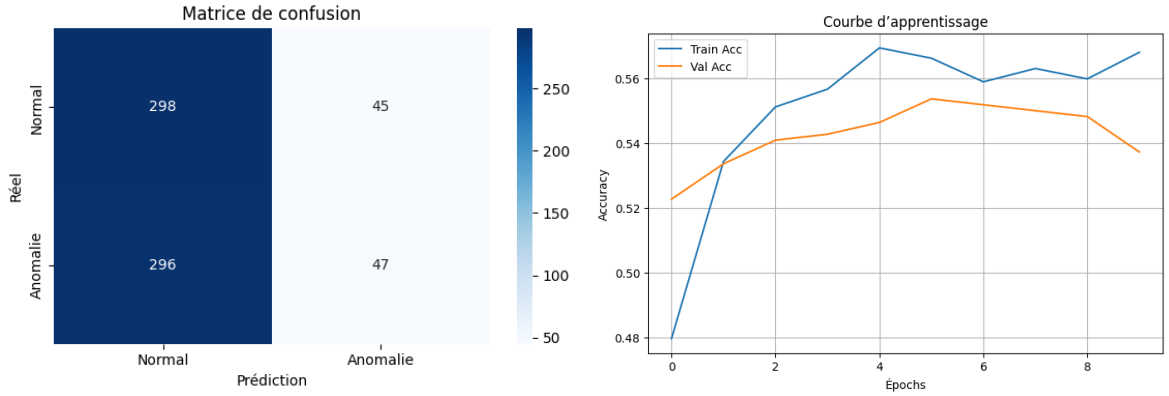

FIGURE 4.8 – Matrice de confusion et courbe d'apprentissage – Modèle hybride, niveau application.

Performances globales :

- Accuracy = 0.50
- Precision = 0.51
- Recall = 0.14

- F1-score = 0.22

Analyse de la matrice de confusion : Ici, la détection des anomalies se dégrade fortement :

- Le rappel chute à 14%, ce qui signifie que la majorité des anomalies passent inaperçues.
- Le taux de faux négatifs est particulièrement élevé, confirmant que les anomalies au niveau agrégé sont plus difficiles à distinguer des comportements normaux.

Analyse de la courbe d'apprentissage : La courbe montre :

1. Une progression lente de l'accuracy, traduisant la difficulté à trouver des motifs discriminants.
2. Un plateau prématuré, signe que le modèle atteint rapidement sa limite d'apprentissage.
3. Une faible marge entre l'entraînement et la validation, ce qui suggère que la limite vient plus de la nature des données que d'un manque de capacité du modèle.

Interprétation : Au niveau application, les données sont le résultat d'agrégations qui diluent les signaux temporels et masquent les anomalies locales. La composante LSTM perd donc une grande partie de sa pertinence, et le MLP, privé de variables distinctives, ne parvient pas à compenser. En d'autres termes, le modèle hybride ne peut pas « inventer » des signaux discriminants inexistants dans les données, même s'il est performant en conditions favorables.

En résumé, le modèle hybride excelle au niveau service grâce à la complémentarité LSTM–MLP, mais n'apporte qu'un gain marginal, voire nul, au niveau application, faute de signaux exploitables. Cette observation confirme l'importance de la granularité des données pour tirer parti d'architectures complexes : un modèle avancé ne peut surperformer que si les données contiennent l'information nécessaire à la discrimination.

4.1.4 Résumé comparatif et tendances

TABLE 4.1 – Résumé comparatif des performances des modèles sur les deux niveaux d'analyse.

Modèle	Niveau	Accuracy	Precision	Recall	F1-score
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

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 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.

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.
- 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.

4.2 Discussion

Les résultats obtenus mettent en évidence un contraste très marqué entre les performances observées au **niveau service** et celles au **niveau application**. Cette différence, bien que prévisible, mérite une analyse approfondie afin d'identifier les causes sous-jacentes et de comprendre leurs implications dans un contexte opérationnel.

Analyse par niveau

Au niveau service. Les données issues du niveau service présentent des motifs temporels et statistiques plus nets, avec une variabilité moins importante et des indicateurs directement liés à la santé de chaque composant (temps de réponse moyen, distribution des latences, taux de codes `2xx` ou `4xx/5xx`). Ces indicateurs, lorsqu'ils sont altérés, traduisent souvent un problème technique localisé, ce qui facilite la tâche des modèles. Ainsi :

- Le **MLP** parvient à modéliser efficacement les relations non linéaires entre les indicateurs grâce à sa structure dense, sans avoir besoin de capturer des dépendances temporelles complexes.
- L'**autoencodeur LSTM** excelle dans la capture des motifs séquentiels, ce qui améliore la détection d'anomalies évolutives ou progressives. Son rappel très élevé montre qu'il détecte presque toutes les anomalies, même subtiles.
- Le **modèle hybride** combine ces forces, atteignant un équilibre entre rappel et précision, et réduisant le risque de faux positifs.

Globalement, le niveau service bénéficie de signaux discriminants forts, stables et directement observables, ce qui explique des *accuracy* supérieures à 90% pour la plupart des approches.

Au niveau application. À l'inverse, le niveau application agrège les métriques issues de plusieurs services. Cette agrégation dilue souvent les signaux propres à un service particulier et introduit un bruit statistique important. Par exemple, une dégradation localisée dans un microservice peut être masquée par la stabilité des autres, ce qui rend l'anomalie difficile à détecter. Conséquences :

- Le **MLP** peine à trouver des frontières de décision claires, ce qui se traduit par un rappel extrêmement faible (12%).
- L'**autoencodeur LSTM** capture certains motifs temporels globaux, mais l'absence de signaux forts limite son efficacité, le rappel restant modeste.
- Le **modèle hybride** n'apporte pas d'amélioration notable, confirmant que le problème ne réside pas uniquement dans le type de modèle, mais dans la nature même des données agrégées.

Ces résultats illustrent que la granularité de la donnée joue un rôle déterminant dans la détection d'anomalies et que l'agrégation excessive peut réduire significativement la sensibilité des modèles.

Forces et limites des approches

- **MLP** :
  - *Forces* : simplicité d'implémentation, rapidité d'entraînement, robustesse sur des jeux de données avec des signaux forts et peu de dépendances temporelles.
  - *Limites* : faible capacité à capturer des dépendances séquentielles ou des anomalies évolutives ; performances dégradées en contexte bruité.
- **Autoencodeur LSTM** :
  - *Forces* : excellente détection des anomalies subtiles grâce à la modélisation des séquences temporelles ; rappel élevé utile dans des environnements critiques où les faux négatifs sont coûteux.
  - *Limites* : sensibilité élevée au choix du seuil d'erreur de reconstruction ; plus grand risque de faux positifs ; entraînement plus coûteux.
- **Hybride LSTM + MLP** :
  - *Forces* : capacité à traiter à la fois des variables temporelles et statiques ; bonne performance lorsque les signaux sont variés.
  - *Limites* : complexité plus importante, ce qui ne garantit pas un gain de performance lorsque les données ne contiennent pas d'informations discriminantes suffisantes.

Implications pratiques

Les différences observées suggèrent qu'en production :

- La surveillance au niveau service est plus fiable pour détecter rapidement des incidents localisés.
- La surveillance au niveau application, bien que moins précise, reste utile pour détecter des pannes globales ou des tendances de dégradation à long terme.
- Les équipes d'exploitation devraient combiner les deux niveaux d'analyse pour maximiser la couverture et la réactivité.

Conclusion

Les résultats révèlent un écart net entre les performances au **niveau service** et au **niveau application**. Au niveau service, la richesse des signaux permet au MLP, au LSTM et au modèle hybride d'atteindre d'excellents scores : le MLP maximise la précision, le LSTM le rappel, et l'hybride trouve un équilibre entre les deux. Au niveau application, l'agrégation des métriques dilue les signaux, entraînant une forte baisse des performances pour tous les modèles, y compris les plus sophistiqués.

Deux enseignements se dégagent :

1. La granularité des données conditionne directement la qualité de la détection.
2. La complexité du modèle n'est utile que si les données contiennent suffisamment d'information exploitable.

En pratique, il est recommandé de privilégier la surveillance fine au niveau service, complétée par une vision globale au niveau application, et de choisir le modèle selon l'objectif prioritaire : précision ou rappel.

Chapitre 5

Conclusion

Ce mémoire avait pour objectif d'explorer et de comparer plusieurs approches d'apprentissage automatique appliquées à la détection d'anomalies dans des environnements microservices, un domaine où la surveillance proactive est devenue un enjeu stratégique. Les architectures microservices, bien qu'offrant flexibilité et scalabilité, introduisent une complexité importante dans le suivi des performances et la détection précoce de dysfonctionnements.

Trois modèles représentatifs ont été implémentés et évalués :

- **Le perceptron multicouche (MLP)** — un modèle simple et rapide, particulièrement efficace lorsque les signaux discriminants sont bien marqués.
- **L'autoencodeur LSTM** — adapté aux données séquentielles et performant sur les séries temporelles présentant des motifs temporels forts.
- **Le modèle hybride LSTM + MLP** — combinant la capture des dépendances temporelles avec le traitement efficace de variables statiques.

Les résultats expérimentaux, obtenus sur deux niveaux d'agrégation des données (service et application), mettent en évidence plusieurs points essentiels :

- **Supériorité du niveau service** : Les performances sont systématiquement plus élevées au niveau service, avec des F1-scores atteignant jusqu'à 0.97 pour le MLP. Ce niveau bénéficie de signaux plus nets, moins dilués par l'agrégation.
- **Limites du niveau application** : L'agrégation des métriques entraîne une perte de granularité et une dilution des signaux d'anomalie, réduisant fortement le rappel (jusqu'à 0.12 pour le MLP et 0.14 pour l'hybride).
- **Apport conditionnel de la complexité** : Le LSTM et le modèle hybride montrent un réel intérêt uniquement lorsque les données présentent à la fois des patterns temporels marqués et des signaux statiques exploitables.

Apports du travail

Ce travail apporte plusieurs contributions notables :

- Une **analyse comparative approfondie** de trois architectures de détection d'anomalies dans un contexte microservices, avec des mesures détaillées (accuracy, précision, rappel, F1-score) et une discussion nuancée sur leurs forces et limites.
- La **mise en évidence de l'importance de la granularité des données** dans les performances des modèles, soulignant que la qualité et la structure des données sont souvent plus déterminantes que la sophistication algorithmique.
- La **production de scripts et visualisations reproductibles**, permettant de répliquer les expériences ou de les adapter à d'autres environnements DevOps.
- Une **première évaluation des modèles hybrides** dans ce contexte, offrant un point de départ pour de futures expérimentations plus ciblées.

Perspectives

Les pistes d'amélioration identifiées ouvrent plusieurs directions prometteuses :

- **Enrichissement des données** : intégrer des logs applicatifs, des traces distribuées et des métriques métier afin de renforcer les signaux exploités par les modèles.
- **Modèles avancés** : tester des architectures intégrant des mécanismes d'attention (*Attention Mechanisms*) pour mieux capturer les dépendances à long terme.
- **Explicabilité et interprétabilité** : intégrer des méthodes comme SHAP ou LIME pour comprendre les décisions du modèle, expliquer pourquoi une anomalie est détectée, afin d'améliorer la confiance et faciliter l'adoption par les équipes de supervision(DevOps).

En définitive, ce travail montre que la performance d'un système de détection d'anomalies ne repose pas uniquement sur le choix d'un algorithme, mais sur un alignement optimal entre la nature des données, la granularité d'observation et la complexité du modèle. Les enseignements tirés ouvrent la voie vers des solutions plus fines, plus explicables et mieux intégrées aux pratiques DevOps modernes, afin d'assurer une surveillance proactive et fiable des architectures microservices.

Bibliographie

[1] Baresi, L. and Garriga, M. (2020). Microservices : The evolution and extinction of web services ? In *Microservices : Science and Engineering*, pages 3–28.

[2] Behera, A., Panigrahi, C. R., and Patel, R. (2023). A comparative analysis of anomaly detection from microservice generated unstructured logs. *International Journal of Advanced Research in Computer Science*, 14(6) :54–59.

[3] Browndon, K. K. C. (2023). Approche d'ingénierie dirigée par les modèles pour la conception, la génération et l'orchestration des microservices. Master's thesis, Université de Yaoundé I, Yaoundé, Cameroun. Mémoire de Master Recherche en Informatique, Option Génie Logiciel et Systèmes d'Informations.

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

[5] de Silva, M., Mahadura, S., Daniel, S., Rupasinghe, L., Kumarapeli, M., and Liyanapathirana, C. (2022). Anomaly detection in microservice systems using autoencoders. In *2022 4th International Conference on Advancements in Computing (ICAC)*, pages 488–493. IEEE.

[6] Giamattei, L., Guerriero, A., Pietrantuono, R., Russo, S., Malavolta, I., Islam, T., Dînga, M., Koziolek, A., Singh, S., Armbruster, M., Gutierrez-Martinez, J. M., Caro-Alvaro, S., Rodriguez, D., Weber, S., Henss, J., Vogelin, E. F., and Panojo, F. S. (2024). Monitoring tools for devops and microservices : A systematic grey literature review. *The Journal of Systems and Software*, 208 :111906.

[7] Lewis, J. and Fowler, M. (2014). Microservices : A definition of this new architectural term. *Scientific Research*.

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

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

[10] Nobre, J., Pires, E. J. S., and Reis, A. (2023). Anomaly detection in microservice-based systems. *Applied Sciences*, 13(7891).

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

[12] Youssfi, M. (2022). Blog 1 : D'une architecture monolithique vers une architecture micro-services.