



Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

SYSTÈME DE BACKTESTING HÉBERGÉ SUR KUBERNETES POUR LA DÉTECTION D'ANOMALIES

RAPPORT

Adam Grossenbacher

Travail de Bachelor

Filière ISC, orientation Ingénierie des Données
Haute école d'ingénierie et d'architecture de Fribourg
ID : B24ISC30

Superviseurs

Jean Hennebert
Jonathan Rial
Jonathan Donzallaz
Beat Wolf

Mandants

iCoSys
LYSR sàrl

Experts

Gérôme Bovet
Geoffrey Papaux

.....

Hes-SO // FRIBOURG
FREIBURG

Fribourg, juillet 2024

Remerciements

Je souhaite tout d'abord présenter mes sincères remerciements à mes experts, M. Gérôme Bovet et M. Geoffrey Papaux, pour avoir choisi ce sujet de bachelor et pour le temps et l'énergie qu'ils y ont consacré.

J'exprime toute ma gratitude à mes superviseurs, M. Jean Hennebert et M. Beat Wolf, ainsi que M. Jonathan Donzallaz et M. Jonathan Rial, pour leur implication, leur soutien tout au long du projet, leurs conseils avisés et leur réactivité face à mes nombreuses questions.

Fribourg, 12 juillet 2024

Adam Grossenbacher

Résumé

Ce travail de bachelor, réalisé à la Haute école d'ingénierie et d'architecture de Fribourg (HEIA-FR) en collaboration avec l'institut iCoSys et la startup LYSR, a pour objectif de développer un système de backtesting pour évaluer des modèles de détection d'anomalies basés sur l'IA à partir de données historiques. Le projet comprend le développement d'un prototype de détection d'anomalies, la mise en place d'un système de backtesting qui utilise des technologies telles que Ray, Kubernetes et MLflow, ainsi que son intégration au backend et au frontend de la plateforme LYSR.

Les résultats démontrent que le système de backtesting fonctionne efficacement et s'intègre bien à la plateforme LYSR, répondant ainsi aux objectifs fixés. Des améliorations futures sont proposées pour optimiser l'efficacité et enrichir les fonctionnalités.

Mots clés : Backtesting, Ray, Détection d'anomalies, Kubernetes, Machine learning, MLflow

Table des matières

| | |
|---------------------------------------|-----------|
| Remerciements | 1 |
| Résumé | 3 |
| Table des matières | 5 |
| Liste des figures | 7 |
| 1 Introduction | 9 |
| 1.1 Contexte et motivations | 9 |
| 1.2 Objectifs principaux | 10 |
| 1.3 Objectifs secondaires | 11 |
| 1.4 Méthodologie | 11 |
| 1.5 Structure | 12 |
| 2 Analyse | 13 |
| 2.1 Anomalie | 13 |
| 2.2 Détection d'anomalies | 18 |
| 2.3 Backtesting | 27 |
| 2.4 Rapport de backtesting | 29 |
| 2.5 Kubernetes | 29 |
| 2.6 Ray | 30 |
| 2.7 MLflow | 33 |
| 2.8 LYSR | 33 |
| 2.9 Conclusion | 35 |

| | |
|---|-----------|
| 3 Conception | 36 |
| 3.1 Détection d'anomalies | 36 |
| 3.2 Backtesting avec Ray | 38 |
| 3.3 Rapport de backtesting | 39 |
| 3.4 Intégration du backtesting au backend de LYSR | 40 |
| 3.5 Intégration du backtesting au frontend de LYSR | 44 |
| 3.6 Conclusion | 45 |
| 4 Implémentation | 46 |
| 4.1 Détection des anomalies : premier prototype | 46 |
| 4.2 Backtesting local avec Ray et MLflow | 51 |
| 4.3 Rapport de backtesting | 54 |
| 4.4 Intégration du backtesting au backend de LYSR | 56 |
| 4.5 Intégration du backtesting au frontend de LYSR | 58 |
| 4.6 Conclusion | 59 |
| 5 Évaluation | 60 |
| 6 Améliorations | 64 |
| 7 Résultats | 67 |
| 8 Durabilité | 80 |
| 9 Conclusion | 83 |
| 9.1 Conclusion personnelle | 84 |
| 9.2 Utilisation des outils d'intelligence artificielle (IA) | 84 |
| Déclaration d'honneur | 85 |
| Glossaire | 86 |
| Bibliographie | 87 |
| Annexes | 89 |

Liste des figures

| | |
|---|----|
| Fig. 2.1: Anomalie ponctuelle | 14 |
| Fig. 2.2: Anomalie collective | 14 |
| Fig. 2.3: Anomalies contextuelles | 15 |
| Fig. 2.4: Anomalie de changement de niveau | 15 |
| Fig. 2.5: Anomalie de tendance | 16 |
| Fig. 2.6: Anomalie de variance | 16 |
| Fig. 2.7: Courbes ROC | 22 |
| Fig. 2.8: Système ML distribué sans Ray | 30 |
| Fig. 2.9: Système ML distribué avec Ray | 30 |
| Fig. 2.10: Composants de Ray | 31 |
| Fig. 3.11: Processus de préparation et d'entraînement des modèles | 37 |
| Fig. 3.12: Processus d'inférence pour la détection des anomalies | 37 |
| Fig. 3.13: Backtesting avec Ray | 38 |
| Fig. 3.14: Diagramme de séquence - Lancement du backtesting | 41 |
| Fig. 3.15: Diagramme de séquence - Statut et logs du backtesting | 43 |
| Fig. 3.16: Frontend du backtesting | 44 |
| Fig. 4.23: Rapport de classification | 50 |
| Fig. 7.24: Requête HTTP : Vérification du fonctionnement de l'API | 68 |
| Fig. 7.25: Requête HTTP : Lancement d'un job de backtesting | 69 |
| Fig. 7.26: Requête HTTP : Récupération du statut d'un job | 70 |
| Fig. 7.27: Requête HTTP : Récupération des logs d'un job | 71 |
| Fig. 7.28: Requête HTTP : Récupération des résultats d'un job | 72 |
| Fig. 7.29: Requête HTTP : Récupération du rapport d'un job | 73 |
| Fig. 7.30: Interface web : Accueil | 74 |
| Fig. 7.31: Interface web : Saisie des informations | 75 |

| | |
|---|----|
| Fig. 7.35: Interface web : Visualisation du rapport | 77 |
| Fig. 7.36: Interface web : Visualisation des logs | 77 |
| Fig. 7.37: Pods Kubernetes | 78 |
| Fig. 7.38: Secrets Kubernetes | 79 |
| Fig. 8.39: Objectifs de développement durable [14] | 80 |

1 Introduction

Ce travail de bachelor est réalisé dans le cadre de la formation « Bachelor of Science HES-SO en Informatique et systèmes de communication », à la Haute école d'ingénierie et d'architecture de Fribourg (*HEIA-FR*). Il est mandaté par l'institut iCoSys, spécialisé dans l'intelligence artificielle et les systèmes complexes, ainsi que par la startup LYSR, qui propose un système continu et performant de surveillance des processus pour les entreprises, basé sur l'intelligence artificielle (*IA*).

1.1 Contexte et motivations

La startup LYSR est spécialisée dans la détection d'anomalies et la maintenance prédictive des données de séries temporelles pour l'industrie 4.0. LYSR propose une plate-forme fiable, facile à utiliser et évolutive pour stocker, visualiser et analyser ces données. La plateforme utilise des technologies telles que Kubernetes, Kafka et Ray, intégrées dans une architecture de microservices évolutive, capable de fonctionner sur des clouds publics ou privés.

Actuellement, la plateforme permet l'intégration de modèles d'intelligence artificielle (*IA*) prêts à l'emploi ainsi que de modèles personnalisés. Ces modèles analysent des flux de données continus (*streams*) et détectent des anomalies en fonction d'un ensemble de règles. Cependant, les nouveaux modèles sont souvent testés avec des données historiques dans des Jupyter Notebooks (souvent en local) ou ne sont pas testés du tout. L'objectif de ce travail de bachelor est de résoudre ce problème en développant un système de backtesting pour évaluer les modèles et les ensembles de règles sur des données historiques, garantissant ainsi leur efficacité dans la détection d'anomalies. Ce système

sera déployé sur Kubernetes et pourra être intégré en tant que microservice dans la plateforme LYSR.

1.2 Objectifs principaux

Les objectifs principaux de ce projet sont les suivants :

1. Détection des anomalies : premier prototype

Le premier objectif de ce projet est de comprendre ce qu'est une anomalie et d'explorer les différentes méthodes de détection, qu'elles utilisent ou non l'intelligence artificielle (IA). Il est également essentiel de se familiariser avec les métriques employées pour évaluer la performance de ces techniques.

Ensuite, l'objectif est de concevoir et d'implémenter un prototype en utilisant un dataset de séries temporelles contenant des anomalies pour tester diverses méthodes de détection. Les étapes clés du prototype incluent l'entraînement d'un modèle sur ces séries temporelles pour générer un score. Des règles sont ensuite appliquées à ce score pour détecter les anomalies. Enfin, les résultats sont comparés à la vérité terrain pour évaluer l'efficacité du modèle à l'aide des métriques appropriées.

2. Backtesting local avec Ray et MLflow

Le deuxième objectif est de comprendre les technologies Ray et MLflow, utilisées par LYSR, ainsi que les concepts de backtesting. Il s'agit ensuite de concevoir et d'implémenter un système de backtesting local dans un Jupyter Notebook en utilisant ces technologies.

Ce système permettra de tester l'efficacité des modèles et des règles en les appliquant à des données. L'objectif est d'identifier la combinaison modèle + règles la plus performante. Les résultats du backtesting, présentés sous forme de valeurs et de scores, seront utilisés dans un prochain objectif pour la création d'un rapport (objectif secondaire n°1).

3. Intégration du backtesting au backend de LYSR

Le troisième objectif consiste à intégrer le système de backtesting développé localement au backend de la plateforme LYSR. Cela nécessite d'abord de comprendre le fonctionnement de la plateforme LYSR, y compris son API pour récupérer les données, les modèles et les règles.

L'intégration comprendra la lecture des données, des modèles et des règles via l'API de LYSR, ainsi que l'exécution des jobs dans le cluster Ray de LYSR.

1.3 Objectifs secondaires

Les objectifs secondaires de ce projet, réalisables en fonction du temps restant après l'accomplissement des objectifs principaux, sont :

1. Génération de rapports de backtesting

Comprendre les éléments clés d'un rapport de backtesting et en générer un (en JSON ou autre format) à la fin du processus, permettant d'observer et de comparer les résultats.

2. Intégration du backtesting au frontend de LYSR

Intégrer le système de backtesting au frontend de LYSR. Ainsi, via une interface web, l'utilisateur pourra sélectionner des modèles, des règles et des données pour effectuer du backtesting.

1.4 Méthodologie

La méthodologie agile est utilisée pour ce travail de bachelor (*TB*), adoptant une approche flexible et itérative de gestion de projet, centrée sur la collaboration et l'adaptation rapide aux changements. Ce choix s'explique par les nombreuses inconnues associées à ce *TB*. En optant pour cette méthodologie, il est possible de livrer rapidement des produits fonctionnels tout en progressant par étapes.

Chaque objectif primaire et secondaire, présenté précédemment, correspond à un sprint (itération) d'une durée d'une à deux semaines. Chaque sprint inclut le développement et les tests d'une partie du travail de bachelor, assurant une progression structurée et méthodique.

Cette structure garantit que chaque itération produit des résultats concrets et évaluables, facilitant ainsi l'ajustement continu et l'amélioration du travail de bachelor. L'approche agile permet de répondre efficacement aux défis imprévus et d'intégrer les retours de manière proactive. Cette méthodologie favorise également une communication ouverte entre toutes les parties prenantes, assurant que les objectifs sont constamment alignés avec les attentes des superviseurs et les besoins évolutifs du *TB*.

1.5 Structure

Ce travail de bachelor est organisé en neuf chapitres.

Le premier 1 « Introduction » présente le contexte, les motivations et les objectifs.

Le chapitre 2 « Analyse » introduit des concepts clés tels que les anomalies, la détection d'anomalies, le backtesting, ainsi que les outils utilisés comme Kubernetes, Ray, MLflow et LYSR.

Le chapitre 3 « Conception » décrit l'architecture et le design du système de backtesting.

Le chapitre 4 « Implémentation » couvre le développement du projet, détaillant le script de backtesting, l'API de backtesting et le frontend du backtesting.

Le chapitre 5 « Évaluation » décrit les tests effectués et les résultats obtenus.

Le chapitre 6 « Améliorations » constitue une réflexion sur de potentielles futures améliorations.

Le chapitre 7 « Résultats » représente une synthèse du travail effectué.

Le chapitre 8 « Durabilité » présente la durabilité du projet vis-à-vis des dix-sept objectifs du développement durable.

Le chapitre 9 « Conclusion » résume les points-clés de ce travail de bachelor, incluant une conclusion personnelle, la déclaration d'honneur et l'utilisation des outils d'intelligence artificielle.

L'ordre des chapitres ne correspond pas nécessairement à l'ordre des tâches réalisées durant le projet.

Les termes professionnels anglo-saxons d'usage courants, en Suisse romande comme dans les pays francophones, sont retranscrits à l'identique.

Les termes en italique sont définis dans le glossaire.

Les références sont identifiées par un numéro et détaillées dans la bibliographie.

Le code source ainsi que tous les documents relatifs au travail de bachelor sont accessibles sur le dépôt GitLab à l'adresse suivante : https://gitlab.forge.hefr.ch/adam.grossenb/tb_backtesting_kubernetes_anomaly_detection^o. Plusieurs fichiers README sont présents dans ce dépôt, fournissant des explications sur le contenu, le code et le fonctionnement.

2 Analyse

Dans ce chapitre, nous explorons diverses technologies et concepts cruciaux pour l'analyse de données et la détection d'anomalies dans des séries temporelles. Cette analyse est structurée en sous-chapitres dédiés aux anomalies, à la détection d'anomalies, au backtesting, à Kubernetes, à Ray, à MLflow et à la plateforme LYSR.

2.1 Anomalie

D'après le dictionnaire Le Robert, une anomalie est un écart par rapport à la normale ou à la valeur théorique [11]. Dans ce travail de bachelor, nous nous intéressons aux anomalies présentes dans des séquences de données mesurées à intervalles de temps réguliers (séries temporelles). Ces données peuvent représenter, par exemple, la température d'un moteur, les ventes d'une entreprise ou le nombre de visites sur un site web. Il existe de nombreux types d'anomalies tels que :

Anomalies ponctuelles (Point Anomaly) :

- **Définition** : Une seule observation qui dévie de manière significative des autres observations.
- **Exemple** : Dans une série de températures journalières, un jour avec une température extrêmement haute par rapport aux autres jours.

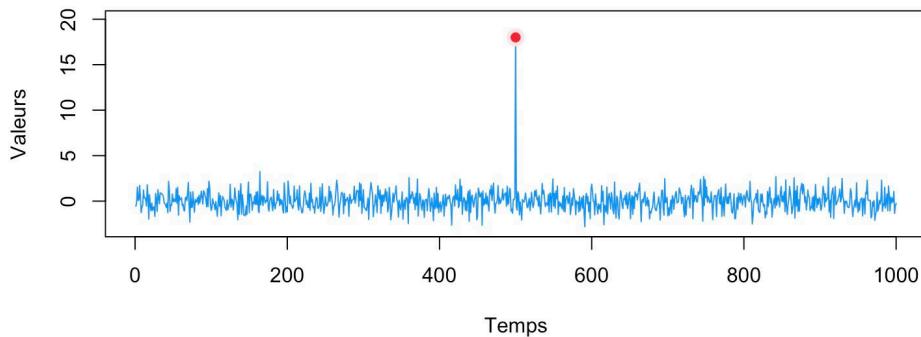


Fig. 2.1 – Anomalie ponctuelle

Anomalie collective (Collective Anomaly) :

- **Définition** : Un groupe d’observations qui, ensemble, dévient du comportement attendu, même si les observations individuelles peuvent ne pas être des anomalies.
- **Exemple** : Un groupe de transactions bancaires consécutives de petits montants transférés en peu de temps, ce qui pourrait indiquer une activité frauduleuse.

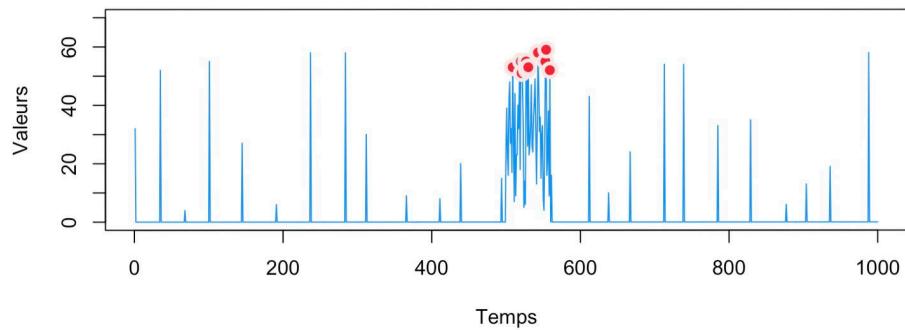


Fig. 2.2 – Anomalie collective

Anomalie contextuelle (Contextual Anomaly) :

- **Définition** : Une observation qui est considérée comme une anomalie dans un certain contexte mais qui peut être normale dans un autre.
- **Exemple** : Une température de 30°C pourrait être normale en été mais serait une anomalie en hiver.

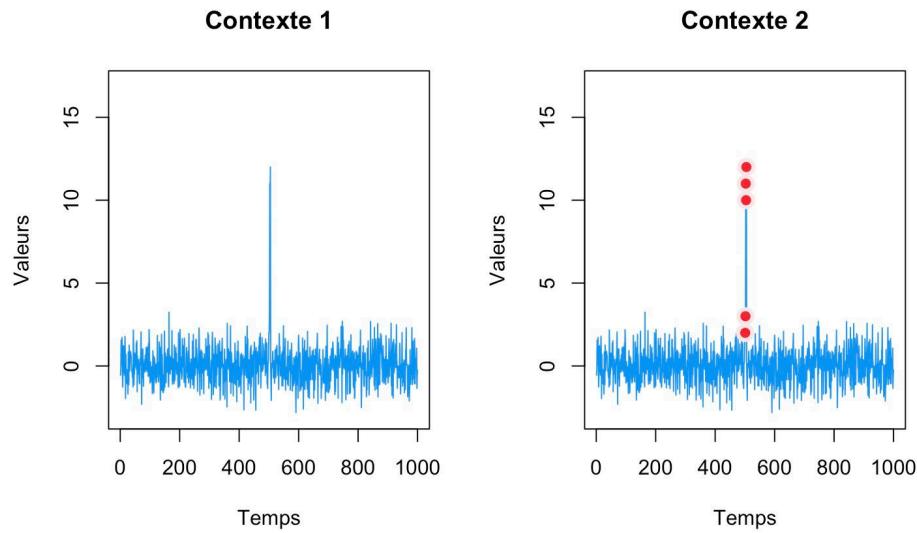


Fig. 2.3 – Anomalies contextuelles

Anomalie de changement de niveau (Level Shift Anomaly) :

- **Définition :** Un changement brusque et significatif dans le niveau moyen des observations.
- **Exemple :** Une série chronologique de ventes mensuelles où il y a soudainement une augmentation ou une diminution permanente des ventes à partir d'un certain point.

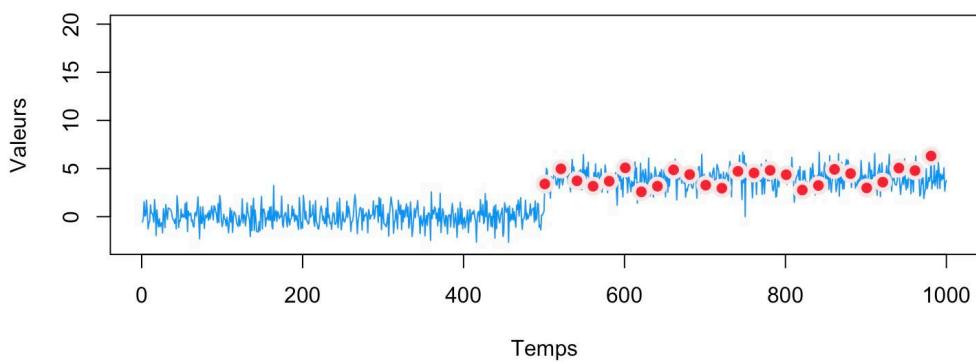


Fig. 2.4 – Anomalie de changement de niveau

Anomalie de tendance (Trend Anomaly) :

- **Définition** : Un changement dans la tendance sous-jacente des données.
- **Exemple** : Une série chronologique des prix d'actions côté en bourse qui stagne pendant plusieurs mois puis commence soudainement à augmenter de manière continue.

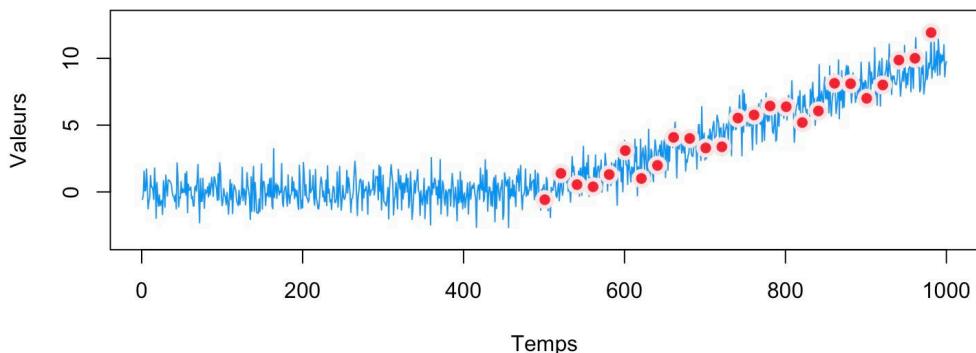


Fig. 2.5 – Anomalie de tendance

Anomalie de variance (Variance Anomaly) :

- **Définition** : Un changement significatif dans la variabilité des données.
- **Exemple** : Une série de données de capteurs où la variance des mesures passe soudainement d'une valeur basse à une valeur élevée, indiquant une possible défaillance du capteur.

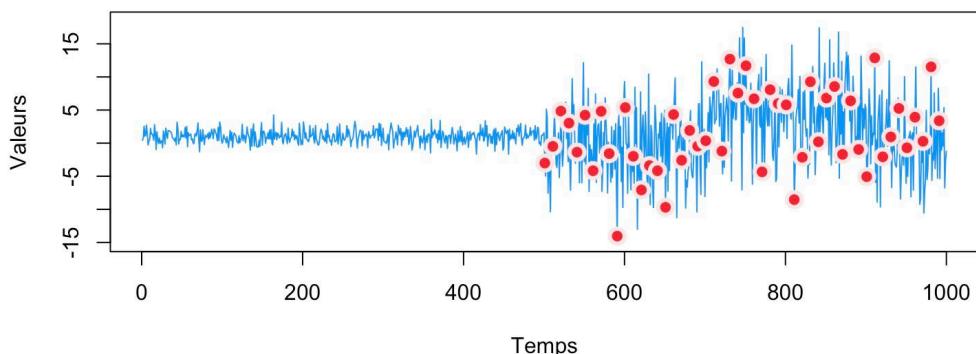


Fig. 2.6 – Anomalie de variance

En conclusion, il existe de nombreux types d'anomalies dans les séries temporelles, chacune étant identifiée de manière différente. Leur détection est souvent un travail minutieux dont la complexité peut varier considérablement en fonction des données. Le chapitre suivant explique comment ces anomalies peuvent être détectées.

2.2 Détection d'anomalies

D'après le dictionnaire Le Robert, la détection est définie comme l'action de détecter, c'est-à-dire de découvrir l'existence de quelque chose [11]. La détection d'anomalies consiste donc à identifier des événements qui deviennent des normes établies. Elle est employée dans des secteurs très divers comme, par exemple, en finance pour détecter des fraudes, en fabrication pour repérer des défauts ou des pannes d'équipement, en cybersécurité pour surveiller des activités réseau inhabituelles ou encore, en santé pour identifier des conditions anormales chez les patients.

LYSR, le mandataire de ce travail de bachelor, se spécialise dans la détection d'anomalies pour l'industrie 4.0. Le processus de détection d'anomalies mis en place par LYSR se déroule en trois étapes :

1. **Collecte des données** : Les données sont collectées en temps réel à partir des machines via des capteurs connectés ou non (Edge Computing).
2. **Traitements et analyse** : Les données sont traitées et analysées en temps réel à l'aide de l'intelligence artificielle (IA), avec des résultats variables selon les modèles utilisés.
3. **Identification des anomalies** : Des anomalies sont détectées en appliquant des règles prédefinies aux résultats obtenus. Un exemple de règle serait un seuil qui ne doit pas être dépassé pendant trois secondes.

Ce processus permet de réduire les coûts et les temps d'arrêt en prédisant ou en signalant lorsque qu'une machine a besoin de maintenance ou de remplacement.

2.2.1 Méthodes de détection sans IA

Comment analyser les flux de données continus pour détecter des anomalies ? Cela peut nécessiter l'utilisation de modèles d'intelligence artificielle (IA) ou d'autres méthodes non basées sur l'IA. La détection des anomalies sans recours à l'IA inclut des méthodes telles que la détection manuelle, l'analyse de données, l'examen de graphiques et l'application de tests statistiques. Ces approches sont généralement efficaces pour des volumes de données limités, mais elles nécessitent souvent l'intervention de spécialistes, tels que des analystes de données ou des statisticiens. Cependant, ces méthodes ne sont pas utilisées par LYSR, car LYSR traite de grandes quantités de données en continu et en temps réel, rendant ces méthodes inefficaces comparées à celles basées sur l'IA.

2.2.2 Méthodes de détection par IA

La détection par intelligence artificielle (IA) est plus rapide que le travail manuel et plus efficace pour traiter de grandes quantités de données. Cependant, elle nécessite l'intervention d'ingénieurs ou de data scientists pour entraîner les modèles. Cette méthode est employée par LYSR et constitue l'un des principaux thèmes de ce travail de bachelor.

Un bon modèle d'IA nécessite des données pour l'entraîner. Ces données se divisent en trois types : les données supervisées, les données non supervisées et les données semi-supervisées. Chaque type de données joue un rôle crucial dans l'approche utilisée pour entraîner le modèle, comme indiqué ci-dessous.

- **Données supervisées** : Les techniques de détection d'anomalies supervisées reposent sur des modèles entraînés avec des données étiquetées, incluant des cas normaux et anormaux. En raison de la rareté des données étiquetées et du déséquilibre des classes, ces techniques sont rarement utilisées. Les Support Vector Machines (SVM) et les k-NN sont deux exemples de ces approches.
 - **Approches basées sur la distance (K-NN)** : Les points normaux sont proches de leurs voisins, tandis que les anomalies sont éloignées. L'algorithme k-NN utilise la distance pour classer les données en fonction des k voisins les plus proches.
 - **Approches basées sur les SVM** : Les SVM (Support Vector Machines) tracent une frontière séparant les données normales des anomalies, avec les anomalies situées en dehors de cette frontière.
- **Données non supervisées** : Les techniques de détection d'anomalies non supervisées construisent un modèle à partir de données non étiquetées pour identifier de manière autonome des motifs ou des anomalies. Elles sont largement utilisées en raison de leur applicabilité étendue, bien qu'elles nécessitent souvent de grands ensembles de données et une puissance de calcul significative. Les méthodes courantes incluent le clustering, les arbres de décision et les réseaux de neurones.
 - **Approches basées sur le clustering (K-means)** : Le clustering regroupe les données similaires en clusters, les anomalies étant les points en dehors de ces clusters.
 - **Approches basées sur les arbres de décision (ex. Isolation Forest)** : L'Isolation Forest divise les observations en sous-ensembles, les anomalies étant plus facilement isolées que les points normaux.
 - **Approches basées sur les réseaux de neurones (ex. auto-encodeurs)** : Les auto-encodeurs apprennent à reconstruire les données normales, détectant les anomalies par une erreur de reconstruction élevée.

- **Données semi-supervisées** : Les techniques de détection d'anomalies semi-supervisées combinent les avantages des approches supervisées et non supervisées, améliorant l'efficacité et la précision des modèles d'apprentissage automatique avec une quantité limitée de données étiquetées. La propagation des labels est un exemple de cette approche.
 - **Propagation des labels** : La propagation des labels utilise des données étiquetées pour attribuer des étiquettes aux données non étiquetées en fonction de leur similarité, puis affine l'algorithme avec ces nouvelles étiquettes.

2.2.3 Métriques d'évaluation

Une fois le modèle développé à partir des données d'entraînement, il est essentiel d'évaluer ses performances et son efficacité pour la détection des anomalies. Pour cela, plusieurs métriques clés sont utilisées. Les principales sont décrites ci-dessous :

Taux de faux positifs : La proportion d'instances normales incorrectement classées comme anomalies. Réduire ce taux est crucial pour éviter les fausses alertes.

$$\text{Taux de faux positifs} = \frac{\text{Faux positifs}}{\text{Faux positifs} + \text{Vrais négatifs}} \quad (2.1)$$

Taux de faux négatifs : La proportion d'anomalies non détectées. Réduire ce taux est important pour s'assurer que les véritables anomalies ne passent pas inaperçues.

$$\text{Taux de faux négatifs} = \frac{\text{Faux négatifs}}{\text{Faux négatifs} + \text{Vrais positifs}} \quad (2.2)$$

Précision globale (Accuracy) : La proportion de prédictions correctes parmi l'ensemble des prédictions. Cette métrique peut être trompeuse dans le contexte de la détection des anomalies en raison du déséquilibre des classes (les anomalies sont rares).

$$\text{Précision globale} = \frac{\text{Vrais positifs} + \text{Vrais négatifs}}{\text{Total des prédictions}} \quad (2.3)$$

Précision (Precision) : La proportion de véritables anomalies parmi les instances identifiées comme anomalies. Cette métrique est cruciale lorsque le coût des fausses alertes est élevé.

$$\text{Précision} = \frac{\text{Vrais positifs}}{\text{Vrais positifs} + \text{Faux positifs}} \quad (2.4)$$

Une précision optimale de 1.0 signifie que toutes les instances détectées comme anomalies sont effectivement des anomalies, sans aucun faux positif.

Rappel (Recall) : La proportion de véritables anomalies détectées parmi toutes les anomalies présentes. Un rappel élevé est important pour minimiser les cas non détectées.

$$\text{Rappel} = \frac{\text{Vrais positifs}}{\text{Vrais positifs} + \text{Faux négatifs}} \quad (2.5)$$

Un rappel optimal de 1.0 signifie que toutes les anomalies présentes dans le jeu de données (global) ont été détectées, sans aucun faux négatif.

Score F1 : La moyenne harmonique de la précision et du rappel, offrant un équilibre entre les deux. Cette métrique est particulièrement utile lorsque les classes sont déséquilibrées.

$$\text{Score F1} = 2 \cdot \frac{\text{Précision} \cdot \text{Rappel}}{\text{Précision} + \text{Rappel}} \quad (2.6)$$

Un F1-score optimal de 1.0 indique un équilibre parfait entre précision et rappel, tandis qu'un F1-score de 0.5 signifie que pour chaque prédiction correcte, le modèle commet deux erreurs (faux négatifs ou faux positifs).

ROC AUC : La courbe ROC (Receiver Operating Characteristic) illustre le compromis entre le taux de vrais positifs et le taux de faux positifs. L'AUC (Area Under the Curve) mesure l'aire sous cette courbe et fournit une indication générale de la performance du modèle.

$$\text{Aire sous ROC} = \int_0^1 \text{ROC}(x) \cdot dx \quad (2.7)$$

Une AUC optimale de 1.0 signifie que le modèle distingue parfaitement les anomalies des instances normales, tandis qu'une AUC de 0.5 indique une performance équivalente au hasard.

Courbes ROC

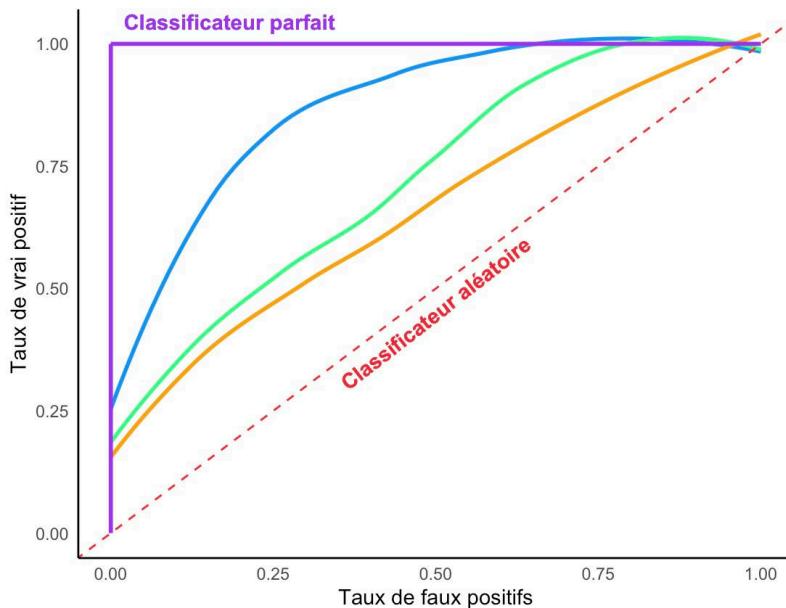


Fig. 2.7 – Courbes ROC

Interprétations des métriques

L'interprétation des métriques est une étape cruciale dans le développement d'un modèle. Il est essentiel de les comprendre pour évaluer et améliorer le modèle. Ci-dessous, un exemple fictif de tableau présente différentes métriques pour divers modèles hypothétiques.

| | Précision | Rappel | F1 | AUC |
|------------|-----------|--------|-------|-------|
| modèle n°1 | 0.640 | 1 | 0.780 | 0.0 |
| modèle n°2 | 0.107 | 0.5 | 0.177 | 0.106 |
| modèle n°3 | 0.470 | 0.462 | 0.462 | 0.205 |

L'interprétation des métriques du tableau est la suivante :

Modèle n°1 : Il est le plus performant en termes de rappel et de F1 score, mais l'AUC de 0.0 est suspecte et nécessite une vérification.

Modèle n°2 : Il présente la plus faible performance globale, avec des scores de précision, rappel, F1 et AUC très bas.

Modèle n°3 : Il offre des performances intermédiaires sur toutes les métriques.

2.2.4 Défis

La détection des valeurs aberrantes dans les séries temporelles est complexe en raison de plusieurs facteurs :

- **Définition des anomalies** : La définition des anomalies varie selon le contexte, nécessitant une contextualisation approfondie.
- **Déséquilibre des classes** : Les anomalies sont rares par rapport aux instances normales, biaisant les modèles vers les instances majoritaires.
- **Variabilité des anomalies** : Les anomalies se manifestent de diverses manières, rendant difficile la création d'un modèle universel.
- **Évolution des données** : Les données changent avec le temps, nécessitant des mises à jour constantes des modèles.
- **Qualité des données** : Le bruit, les données incorrectes ou manquantes peuvent masquer les anomalies, compliquant le prétraitement.
- **Élaboration du modèle IA** : Features engineering, éviter le surapprentissage (overfitting) et garantir la robustesse.

Ces défis nécessitent des approches sophistiquées et adaptatives pour être surmontés, exigeant vigilance et adaptation continue des méthodes.

2.2.5 Feature Engineering

Le Feature Engineering est crucial pour transformer les données brutes en features plus représentatives et exploitables par les modèles d'IA, améliorant ainsi la détection des anomalies et la performance des modèles prédictifs. Cela inclut la sélection de features pertinentes, la transformation des features, la création de nouvelles features dérivées et l'ajout d'interactions entre variables.

• Sélection des features

La sélection des features identifie et conserve les variables les plus pertinentes pour le modèle.

- **Pourquoi sélectionner les features ?**

Réduire le nombre de variables simplifie les modèles et améliore la performance.

- **Exemples de méthodes de sélection des features :**

- Méthodes statistiques : Utilisation de statistiques pour évaluer l'importance des features.
- Algorithmes de sélection : Utilisation d'algorithmes de ML pour sélectionner les features (par ex., recursive feature elimination).

• Extraction des features

L'extraction des features crée de nouvelles variables à partir des données existantes, permettant de capturer des informations importantes.

- **Pourquoi extraire des features ?**

Cela réduit la complexité des données, révèle des structures sous-jacentes, et améliore la performance des modèles.

- **Exemples de techniques d'extraction des features :**

- Analyse en composantes principales (PCA) : Réduction de la dimensionnalité.
- Analyse discriminante linéaire (LDA) : Axes maximisant la séparation entre les classes.

• Transformation des features

La transformation des features modifie les variables existantes pour les rendre plus adaptées aux modèles prédictifs.

- **Pourquoi transformer les features ?**

Les transformations stabilisent la variance, rendent les données plus normales ou améliorent la convergence des algorithmes.

- **Exemple de transformations de features :**

- Normalisation : Redimensionnement des valeurs de sorte qu'elles tombent dans une gamme spécifiée, généralement $[0, 1]$ ou $[-1, 1]$.
- Logarithmique : Réduction de la skewness d'une distribution.

• **Création de features dérivées**

Les features dérivées sont de nouvelles variables créées à partir des variables existantes pour capturer des informations supplémentaires ou des patterns non apparents dans les données d'origine.

▸ **Pourquoi ajouter des features dérivées ?**

Elles peuvent améliorer la performance des modèles de prédiction en capturant des tendances ou des changements subtils que les variables d'origine ne révèlent pas. Par exemple, la variation de la température (première dérivée) ou l'accélération de cette variation (deuxième dérivée) peuvent être plus indicatives qu'une simple mesure de température.

▸ **Exemples de features dérivées :**

- Première dérivée : Calcul du taux de changement d'une variable, comme la vitesse de changement de la température (dx/dt)
- Deuxième dérivée : Calcul de l'accélération du changement d'une variable (d^2x/dt^2).
- Transformations statistiques : Moyennes, écarts-types, quantiles et autres statistiques pour résumer et décrire la distribution des données.
- Transformations catégoriques : Fréquence d'apparition.

• **Interactions entre features**

Les interactions entre features capturent l'effet combiné de deux ou plusieurs variables, révélant des relations complexes non visibles dans les variables individuelles.

▸ **Pourquoi ajouter des interactions entre features ?**

Elles permettent aux modèles d'IA de capturer des relations non linéaires et des effets combinés entre variables. Par exemple, la combinaison de la température et de l'humidité peut avoir un impact différent par rapport à ces variables prises séparément.

▸ **Exemples d'interactions entre features :**

- Produit de variables : Interaction représentée par le produit de deux features ($x_1 * x_2$).
- Ratios : Capturer les effets relatifs entre variables.
- Fonctions non linéaires : Appliquer des fonctions combinées, comme $\sin(x_1) * \cos(x_2)$.

En résumé, la détection d'anomalies est essentielle dans des secteurs comme la finance, la fabrication, la cybersécurité et la santé. LYSR se spécialise dans ce domaine pour l'industrie 4.0 en utilisant l'IA pour analyser des flux de données en temps réel.

Les méthodes de détection peuvent être manuelles ou automatisées. Les techniques manuelles conviennent aux petites quantités de données, tandis que l'IA est plus efficace pour les grands volumes. Les données d'entraînement peuvent être supervisées, non supervisées ou semi-supervisées, influençant ainsi l'algorithme utilisé.

Les métriques telles que la précision, le rappel, le score F1 et l'AUC sont essentielles pour évaluer les modèles mais doivent être interprétées avec soin. Les défis principaux incluent la définition des anomalies, le déséquilibre des classes, la variabilité des données et leur qualité. Le Feature Engineering est crucial pour maximiser la performance des modèles et améliorer la détection des anomalies.

En conclusion, bien que complexe, la détection d'anomalies est indispensable. Elle améliore notamment les performances des services, la qualité des produits et l'expérience utilisateur. Les avancées en intelligence artificielle et en traitement des données permettent d'affiner les méthodes de détection, les rendant toujours plus précises.

2.3 Backtesting

Le backtesting vise à simuler les performances d'un modèle en l'appliquant à des données historiques pour évaluer sa précision et sa fiabilité.

Actuellement, LYSR ne dispose pas de système de backtesting. Lorsqu'un nouveau modèle d'IA est développé, il est généralement testé localement avec des données historiques dans un Jupyter Notebook, et parfois même pas du tout. L'objectif de ce travail de bachelor est de remédier à ce problème en développant un système de backtesting distribué, directement intégré à la plateforme LYSR. Ce système permettra d'évaluer les modèles ainsi que les règles associées pour identifier la meilleure combinaison afin de détecter les anomalies avec la plus grande précision.

2.3.1 Étapes

Les étapes du backtesting sont les suivantes :

1. **Sélection des données historiques** : Choisir des données représentatives et s'assurer que la période sélectionnée couvre différents scénarios de marché pour obtenir une évaluation exhaustive.
2. **Application du modèle aux données** : Fournir les données historiques au modèle, qui génère des sorties via l'inférence, pouvant être des scores ou des classifications.
3. **Application des règles aux outputs du modèle** : Appliquer les règles aux sorties du modèle pour déterminer si elles constituent une anomalie.
4. **Analyse des résultats obtenus** : Mesurer les performances du modèle et des règles en utilisant des métriques appropriées, telles que celles présentées au chapitre précédent (2.2.3 métriques d'évaluation).
5. **Rapport** : Générer un rapport de backtesting et enregistrer les résultats.

2.3.2 Avantages

Le backtesting offre de nombreux avantages, notamment :

- **Validation du modèle** : Permet de vérifier l'efficacité d'un modèle avant de l'appliquer dans des conditions réelles.
- **Validation des règles** : Permet de s'assurer que les règles sont correctement définies et adaptées au modèle.

- **Identification des points faibles** : Aide à repérer les faiblesses et les risques potentiels associés au modèle.
- **Optimisation** : Facilite l'ajustement et l'amélioration du modèle pour maximiser ses performances.

2.3.3 Limites

Cependant, le backtesting présente plusieurs limites :

- **Disponibilité des données** : Les données disponibles peuvent être insuffisantes pour certaines applications.
- **Dépendance aux données historiques** : Les modèles peuvent être trop optimisés pour les données passées, ce qui ne garantit pas leur performance future.
- **Sur-optimisation (overfitting)** : L'ajustement excessif des modèles aux données historiques peut conduire à des performances trompeuses sur de nouvelles données.
- **Faux positifs** : Il est crucial d'éviter les faux positifs, qui peuvent donner une fausse impression de la performance du modèle, ainsi que les faux négatifs.
- **Hypothèses simplifiées** : Le backtesting peut simplifier certaines hypothèses, bien qu'elles soient souvent beaucoup plus complexes en réalité.

2.3.4 Conclusion

Un backtesting rigoureux permet d'identifier les forces et les faiblesses d'un modèle ainsi que les meilleures règles à lui appliquer, facilitant les ajustements avant son déploiement en production. Cependant, les performances passées ne garantissent pas les performances futures. Ce pourquoi les modèles doivent être mis à jour régulièrement pour rester efficaces. En résumé, le backtesting est essentiel dans le développement et l'évaluation des modèles.

2.4 Rapport de backtesting

L'analyse de la littérature pour trouver des exemples de rapports de backtesting dans le cadre de la détection d'anomalies s'est avérée difficile. En effet, la majorité des résultats concernent le backtesting dans les domaines de la finance et des marchés boursiers qui utilisent des rapports très différents, avec des métriques spécifiques telles que le rendement et le bénéfice. Ces éléments sont peu pertinents pour la détection d'anomalies. Les seules informations pertinentes trouvées sont des conseils [15] pour évaluer la détection d'anomalies. Parmi ces conseils, on retrouve les métriques présentées au chapitre 2.2.3 pour évaluer les résultats, telles que la précision, le rappel et le f1-score. Il y est également recommandé de visualiser les données afin de communiquer clairement les résultats. Enfin, il est rappelé d'interpréter les résultats avec prudence, car les performances passées ne garantissent pas les performances futures.

2.5 Kubernetes

Kubernetes (*K8s*) est un système open-source qui automatise le déploiement, la mise à l'échelle et la gestion des applications conteneurisées. Il est essentiel à ce projet car le système de backtesting développé repose sur Ray, qui lui-même utilise Kubernetes. De plus, Kubernetes offre des fonctionnalités avancées telles que la découverte de services, la gestion des configurations et des secrets, ainsi que la mise à jour continue des applications, ce qui améliore la fiabilité et la maintenabilité du système de backtesting.

2.6 Ray

Ray est un framework de calcul unifié open-source qui facilite la mise à l'échelle des charges de travail en IA et en Python [1]. Il permet ainsi de passer de plusieurs systèmes distribués distincts (Figure 2.8), chacun correspondant à une partie spécifique du ML, à un seul système distribué qui englobe toutes les parties du ML (Figure 2.9).

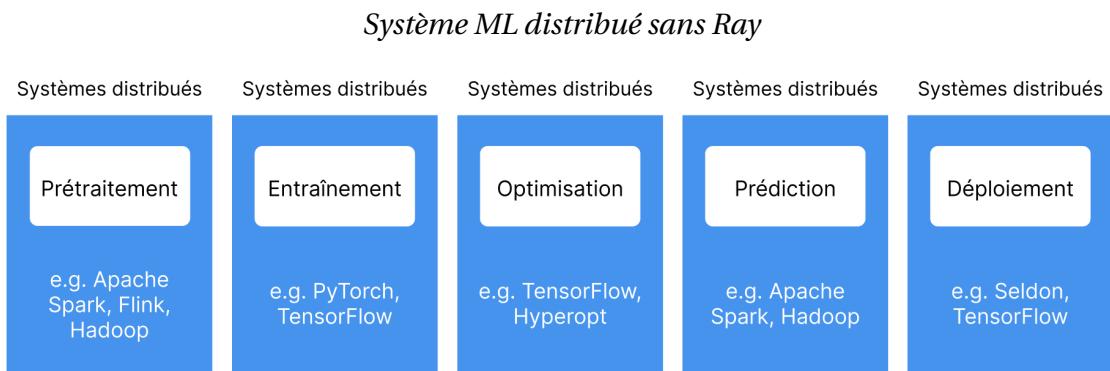


Fig. 2.8 – Système ML distribué sans Ray

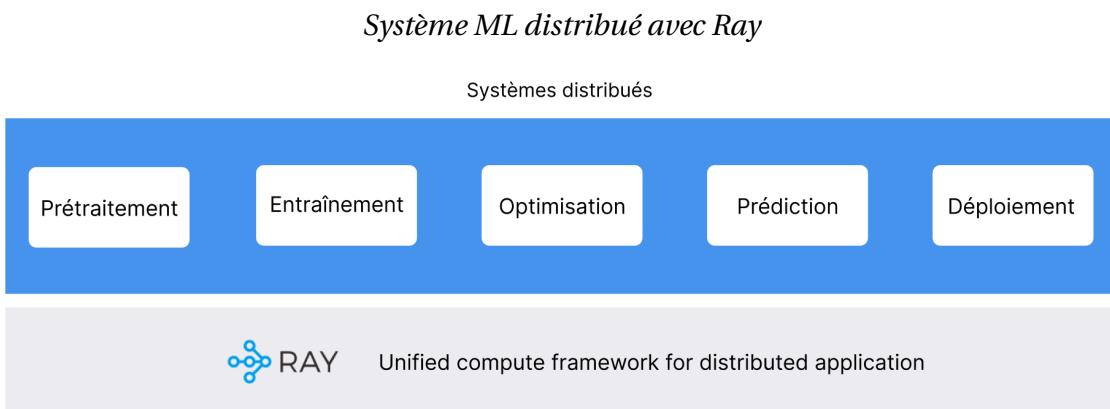


Fig. 2.9 – Système ML distribué avec Ray

Ray simplifie la création de plateformes de machine learning évolutives et robustes en offrant des outils de calcul, une API ML unifiée, et en facilitant la transition du développement à la production. Ray automatise également la gestion des composants, la planification des tâches et la mise à l'échelle automatique. Ainsi, Ray simplifie le développement, le déploiement et la gestion des systèmes de ML distribués. Ce framework est crucial pour ce travail, car le système de backtesting doit fonctionner sur le cluster Ray de LYSR. La Figure 2.10 présente un schéma des composants essentiels de Ray

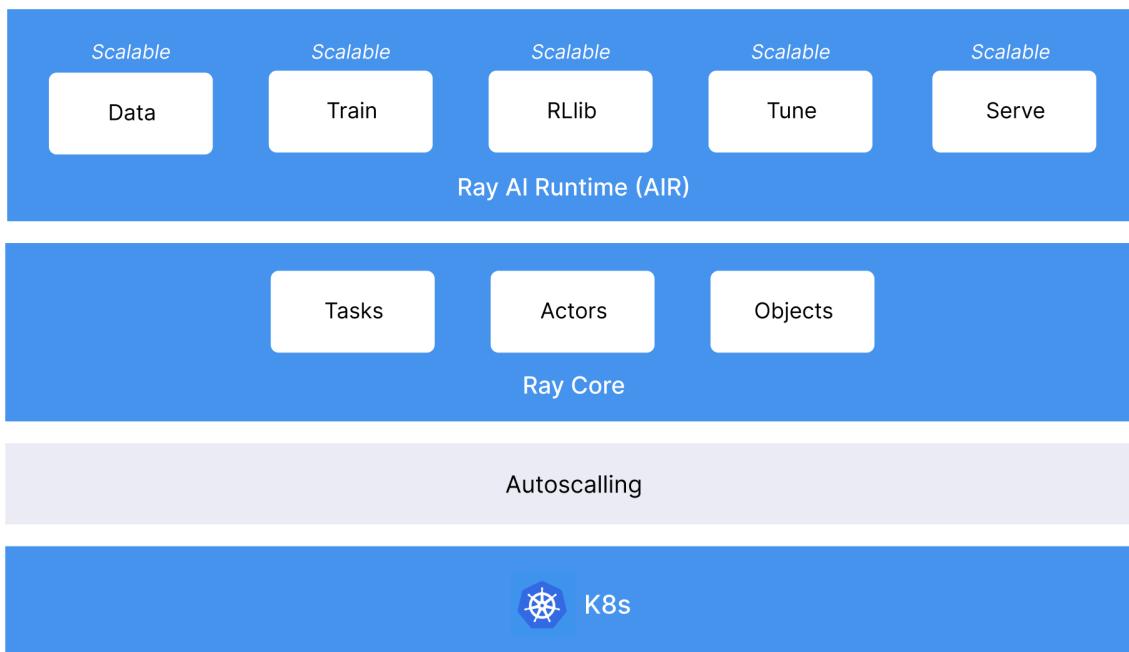


Fig. 2.10 – Composants de Ray

L'image ci-dessus montre que Ray se compose de trois couches : Ray AI Runtime, Ray Core et K8s (Ray Cluster). Chacune de ces couches contient des composants distincts. Voici une brève description de ces couches et de leurs composants.

Ray AI Runtime : Bibliothèques Python open-source pour des outils ML.

- Data : Chargement et prétraitement de données.
- Train : Entraînement de modèles distribué.
- RLLib : Apprentissage par renforcement distribué.
- Tune : Ajustement des hyperparamètres.
- Serve : Mise en service pour déployer des modèles en ligne.

Ray Core : Bibliothèque Python open-source pour le calcul distribué et le scaling des applications ML.

- Tasks : Ray exécute de manière asynchrone des fonctions sur des workers Python séparés, appelés tasks. Les tasks peuvent spécifier leurs besoins en termes de CPU, GPU et ressources personnalisées, ce qui permet au planificateur de cluster d'optimiser l'exécution parallélisée.
- Actors : Les actors sont des workers avec état (stateful workers). Lorsqu'un actor est instancié, un nouveau woker est créé. Les méthodes de cet actor peuvent accéder et modifier l'état du worker. Les actors peuvent également spécifier leurs besoins en CPU, GPU et ressources personnalisées.
- Objects : Les tasks et les actors créent des objects distants, stockés n'importe où dans le cluster. Ces objects sont mis en cache dans un object store en mémoire partagée. Il peut y avoir un seul store dédié par nœud, mais un object distant peut résider sur plusieurs nœuds. Les objects distants peuvent être partagés et récupérés à travers le cluster sans qu'une task ou un acteur spécifique maintienne une référence continue à ces object.

K8s (Ray Cluster) : Ensemble de nœuds worker. K8s ajuste automatiquement le nombre de nœuds du cluster Ray en fonction de la demande.

Pour plus d'informations, la documentation de Ray [16] explique en détail tous ses composants et bien plus encore.

2.7 MLflow

MLflow est une plateforme conçue pour simplifier le développement de projets en machine learning. Elle facilite le suivi des expériences, l'emballage du code dans des environnements reproductibles, ainsi que le partage et le déploiement des modèles. LYSR utilise MLflow pour héberger ses modèles dans le registre de modèles MLflow.

Un des principaux avantages de MLflow est ses modèles standardisés, qui s'intègrent avec une variété d'outils, permettant ainsi une meilleure reproductibilité et un déploiement facile. Par exemple, un modèle peut être développé avec l'une des nombreuses bibliothèques compatibles avec MLflow, telles que PyTorch, Scikit-learn, Hugging Face, Keras, et TensorFlow, pour n'en citer que quelques-unes. Ensuite, ce modèle peut être enregistré au format MLflow standard, le rendant facilement utilisable tous de la même manière.

2.8 LYSR

LYSR est une plateforme d'Intelligence Artificielle (IA) dédiée à la surveillance des processus industriels. Ses principales fonctionnalités sont les suivantes :

1. **Traitements évolutifs** : Permet aux entreprises de surveiller simultanément des milliers de flux de données.
2. **Algorithmes avancés et IA plug-and-play** : Envoie des alertes en temps réel pour la maintenance prédictive.

Ces caractéristiques font de LYSR un outil prometteur pour optimiser la gestion et la maintenance des infrastructures industrielles.

2.8.1 Stack technologique

La plateforme LYSR utilise principalement Python et Java pour exécuter ses modèles et microservices, en s'appuyant sur MLflow et Ray. Elle fonctionne sur un cluster Kubernetes, qui gère également un cluster Ray pour l'orchestration des tâches de machine learning. MLflow est utilisé pour héberger les modèles de machine learning. Kafka gère les flux de données en temps réel, Minio assure le stockage permanent des données compatible avec S3, et Grafana Mimir sert de base de données temporelle pour le stockage et l'indexation des données chronologiques. Bien qu'il existe d'autres tech-

nologies utilisées, elles sont de moindre importance pour ce travail. Ces technologies constituent donc les contraintes de ce travail de bachelor, étant donné que le système de backtesting devra pouvoir s'intégrer à la plateforme LYSR.

2.8.2 Workflow

Le workflow de la plateforme LYSR est le suivant (adapté de la documentation LYSR [2]).

1. **Workspace definition** : Un workspace est un environnement dédié où l'utilisateur organise les ressources liées à un processus de surveillance. Il comprend des endpoints, des streams, des processeurs, des modèles, des règles et des dashboards.
2. **Endpoint definition** : Un endpoint est un point d'accès lié à un workspace, utilisé pour insérer de nouvelles données dans la plateforme. LYSR génère automatiquement une API REST permettant aux gateway ou systèmes embarqués d'envoyer leurs données.
3. **Sending data streams** : Un stream est une série chronologique de valeurs stockées par LYSR, créée lorsque des systèmes embarqués ou des gateway transmettent des données collectées via des capteurs.
4. **Exploring values** : L'outil Explorer de LYSR permet de visualiser et d'inspecter les valeurs des streams. Des graphiques et tableaux peuvent être générés pour faciliter la navigation dans les données.
5. **Defining dashboards** : Un dashboard est composé de graphiques basés sur les données des streams. Sa configuration est flexible et peut être sauvegardée. Les valeurs sont mises à jour approximativement chaque seconde pour une surveillance en temps réel.
6. **Defining models** : Un modèle peut être un réseau neuronal profond (deep neural network), un arbre de décision (decision tree) ou des algorithmes plus simples. Une fois définis, les modèles peuvent être instanciés et connectés pour traiter et analyser les streams.
7. **Instanciating processor** : Un processeur est une instance de modèle appliquée à un stream, générant un output, souvent un score d'anomalie. Les streams sont associés aux inputs du processeur, qui se déclenche selon différentes stratégies : à l'arrivée de nouvelles données ou lorsque les fenêtres de traitement sont remplies. L'output du processeur crée un nouveau stream dans LYSR.

8. **Defining rules :** Une règle permet d'envoyer des alertes lorsque certaines conditions sont remplies. LYSR permet de définir des règles flexibles basées sur les valeurs et timestamps des streams. Lorsqu'une condition est remplie, une alerte est déclenchée et envoyée à l'utilisateur. Le système d'alerte inclut différents paramètres pour une flexibilité maximale (patience, auto-close, remind).

2.9 Conclusion

Ce chapitre a présenté une analyse succincte des concepts et technologies essentiels à la détection d'anomalies dans les séries temporelles. Nous avons défini les anomalies, exploré les principes méthodes de détection manuelles et automatisées, et présenté des métriques d'évaluation pour mesurer la performance des modèles d'IA.

Le backtesting a été abordé comme un outil crucial pour valider les modèles sur des données historiques, garantissant leur fiabilité avant leur déploiement. Kubernetes et Ray ont été introduits comme des solutions clés pour la gestion des applications distribuées, permettant une mise à l'échelle efficace et un déploiement robuste dans les applications de machine learning.

MLflow a été mis en avant pour sa capacité à suivre et à déployer des modèles de machine learning de manière reproductible. Enfin, la plateforme LYSR a été brièvement décrite, illustrant son efficacité dans la surveillance et l'analyse des flux de données en temps réel.

En résumé, ce chapitre démontre l'importance d'intégrer des technologies avancées à des méthodologies rigoureuses pour améliorer la détection d'anomalies et optimiser les processus dans tous les secteurs concernés.

3 Conception

Dans ce chapitre, nous détaillons la conception des différentes composantes nécessaires à l'élaboration du système de backtesting. Cette conception est structurée en sous-chapitres qui abordent le fonctionnement de la détection d'anomalies, le backtesting avec Ray, la structure du rapport de backtesting, ainsi que l'intégration du backtesting au backend et au frontend de LYSR.

3.1 Détection d'anomalies

La détection d'anomalies en temps réel via IA se fait en deux étapes distinctes. La première est l'entraînement du modèle et la seconde, la détection d'anomalies à l'aide du modèle. Ces deux étapes sont détaillées dans les sections suivantes.

Entraînement du modèle

La première étape de la détection d'anomalies par IA dans les séries temporelles consiste à entraîner un modèle avec un jeu de données historiques ou fictif si aucune donnée n'est disponible. Les deux parties principales, comme le montre la figure 3.11, sont le traitement des données et l'entraînement du modèle. Une fois validé, le modèle peut être déployé dans un registre de modèles (p. ex. MLflow Model Registry).

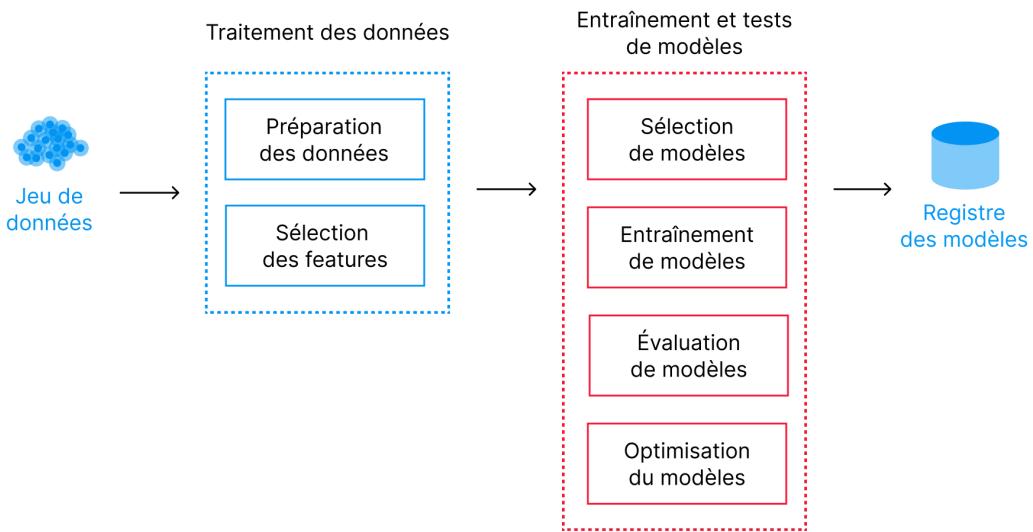


Fig. 3.11 – Processus de préparation et d'entraînement des modèles

Le traitement des données comprend la collecte, la préparation (par exemple, le nettoyage, la normalisation) et la sélection des features pertinentes pour le modèle. L'entraînement du modèle inclut le choix de l'algorithme (par exemple, isolation forest, autoencodeurs), l'entraînement avec les données et l'évaluation à l'aide de métriques telles que la précision, le rappel et le F1-score.

Détection des anomalies

La deuxième étape, après le développement du modèle, consiste à détecter les anomalies potentielles. Comme illustré à la figure 3.12, cette étape inclut la génération de séries temporelles à partir des capteurs, l'analyse des données par inférence, et l'application de règles pour identifier les anomalies.

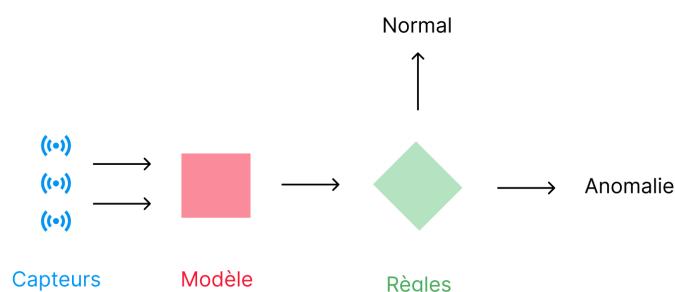


Fig. 3.12 – Processus d'inférence pour la détection des anomalies

3.2 Backtesting avec Ray

Le système de backtesting à implémenter dans ce travail doit fonctionner avec Ray. Cela permet de distribuer les processus de backtesting sur plusieurs nœuds, améliorant ainsi les performances et optimisant l'utilisation des ressources. Le schéma ci-dessous (Figure 3.13) illustre les étapes du système de backtesting fonctionnant avec Ray.

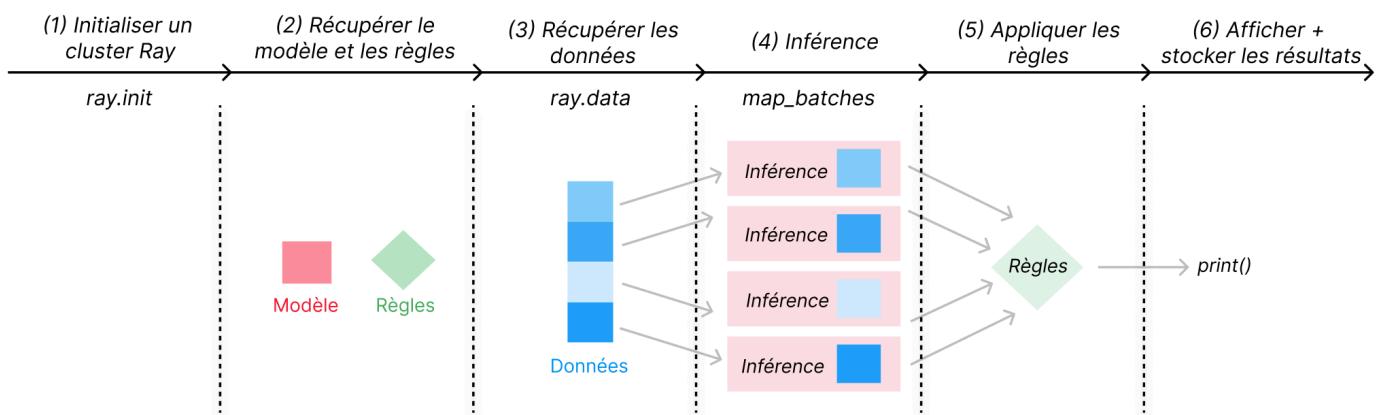


Fig. 3.13 – Backtesting avec Ray

Les étapes sont les suivantes :

- (1) Initialiser du cluster Ray** : Démarrer un cluster Ray ou se connecter à un cluster existant pour permettre la parallélisation des tâches avec `ray.init`.
- (2) Récupérer le modèle et les règles** : Importer le modèle et les règles via l'API de LYSR. Les modèles sont au format MLflow. Les règles sont des chaînes de caractères qui, une fois appliquées, retournent un booléen.
- (3) Récupérer les données** : Importer les données au format CSV ou Parquet via l'API de LYSR. Ces données sont ensuite converties en Ray Dataset avec `ray.data`. Cela permet un traitement parallèle. `Ray.data` prend en charge la planification des ressources CPU et GPU.
- (4) Inférence** : Réaliser l'inférence en parallèle si le modèle est stateless, sinon en séquence si stateful, sur les partitions des données avec `ds.map_batches()`. Cette méthode permet de configurer le traitement en parallèle via le paramètre `concurrency` (le nombre de workers Ray à utiliser simultanément). Elle offre également d'autres paramètres de

configuration tels que batch_size (le nombre de lignes par batch) et gpus (le nombre de GPUs à réserver pour chaque worker parallèle).

(5) **Appliquer les règles** : Appliquer les règles sur les résultats d'inférence collectés. Cette étape peut également être effectuée en parallèle de manière similaire à l'inférence, en utilisant `ds.map_batches()`.

(6) **Afficher + stocker les résultats** : Présenter et enregistrer les résultats collectés. L'enregistrement se fait sous forme d'objet storage au format Parquet.

Comme mentionné précédemment, le backtesting ne réalise pas l'inférence de la même manière pour les modèles stateless et stateful, car les modèles stateful maintiennent un état interne dépendant des entrées précédentes. Pour garantir la cohérence et l'actualité de cet état, il est essentiel de ne pas effectuer les inférences en parallèle, ce qui pourrait sinon entraîner des résultats incohérents ou non pertinents. Dans ce travail, il est souhaité que les deux cas soient pris en compte. Pour ce faire, l'information indiquant si un modèle est stateful ou stateless se trouve dans les métadonnées du modèle. Les étapes présentées précédemment seront implémentées en Python pour assurer la compatibilité avec Ray.

3.3 Rapport de backtesting

Le backtesting, présenté dans le chapitre précédent, génère des résultats bruts tels que les prédictions et les résultats des règles appliquées. Ces données sont utilisées pour créer un rapport facilitant la compréhension et l'interprétation des résultats du backtesting. Le contenu du rapport est le suivant :

- **Matrice de confusion** : 0 à N matrices de confusion proportionnelles au nombre de règles (uniquement si les données sont étiquetées).
- **Métriques** : 0 à N métriques (précision, rappel, f1-score) proportionnelles au nombre de règles (uniquement si les données sont étiquetées).
- **Graphiques des séries temporelles**, comprenant différentes sous-parties :
 - Les données initiales (input)
 - Les anomalies réelles (uniquement si les données sont étiquetées)
 - Les résultats des règles (uniquement si des règles sont définies)
 - Les résultats des règles combinées avec l'opérateur OR (uniquement s'il y a au moins deux règles)

Ce rapport permet de visualiser les données temporelles initiales ainsi que les emplacements des anomalies réelles et potentielles. Il évalue également la performance des modèles et des règles combinées lorsque les données sont étiquetées. Le rapport sera au format HTML pour permettre des graphiques dynamiques et interactifs, ce qui est utile, par exemple, pour sélectionner uniquement les valeurs intéressantes, zoomer sur une période spécifique et exporter ces vues sous forme d'image. De plus, le format HTML du rapport facilitera son intégration au frontend de la plateforme LYSR. La bibliothèque Plotly sera utilisée car elle répond parfaitement à ces besoins. Le rapport sera généré par un script Python.

3.4 Intégration du backtesting au backend de LYSR

LYSR propose de nombreux services, notamment pour leur plateforme (alerte, ingestor, etc.), le stockage (MLflow, MinIO, etc.), et le cluster Ray. Pour intégrer le script de backtesting à LYSR et ses services, de nouveaux services spécifiques au backtesting doivent être implémentés.

Les schémas ci-dessous (Figures 3.14 et 3.15) illustrent les nouveaux services (en bleu) à implémenter et leur interaction avec les services existants de LYSR (en gris). Le premier diagramme de séquence (Figure 3.14) montre le processus de lancement d'un job de backtesting.

sd [Run a backtesting job]

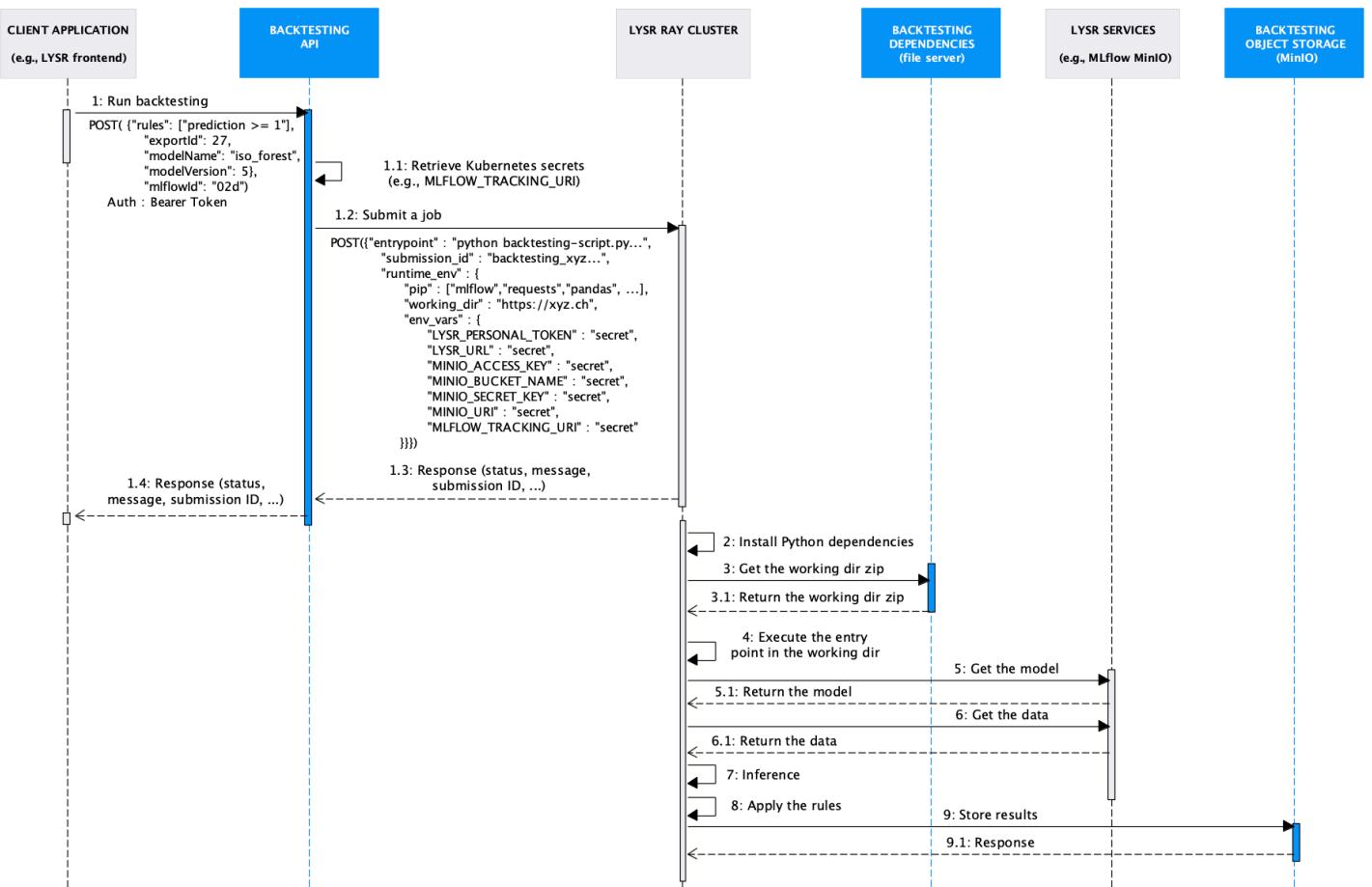


Fig. 3.14 – Diagramme de séquence - Lancement du backtesting

Description des interactions du diagramme de séquence précédent :

1: Run backtesting - Un utilisateur envoie une requête POST à l'API REST du backtesting avec les informations nécessaires : identifiant du dataset (exportId), nom et version du modèle (ModelName & ModelID), liste de règles (rules) et identifiant MLflow de l'utilisateur (mlflowId). L'accès token de LYSR doit être renseigné dans le header. L'API, développée en Java avec Spring Boot, assure la compatibilité avec les autres services de LYSR.

1.1: Retrieve Kubernetes secrets - L'API backtesting récupère les secrets Kubernetes pour accéder aux services LYSR, MLflow et MinIO.

1.2: Submit a job - L'API backtesting envoie une requête POST au cluster Ray pour lancer le job avec les détails nécessaires : commande à exécuter (entrypoint), dépendances (pip), répertoire de travail (working_dir) et variables d'environnement (env_vars). Le cluster Ray est accessible uniquement au sein du cluster Kubernetes.

1.3: Response - Le cluster Ray retourne une réponse HTTP à l'API backtesting avec un status code et un message JSON.

1.4: Response - L'API backtesting renvoie une réponse HTTP, au format JSON, à l'utilisateur incluant le submission ID nécessaire au suivi des logs et du statut du job.

2: Install Python dependencies - Le cluster Ray installe les dépendances nécessaires sur les workers utilisés pour le backtesting.

3: Get the working dir zip - Le script de backtesting est téléchargé depuis le serveur de fichiers, servant de répertoire de travail.

4: Execute the entry point in the working dir - La commande de lancement du script de backtesting est exécutée dans le répertoire de travail.

5: Get the model - Le modèle sélectionné est téléchargé depuis le registre de modèles de MLflow.

6: Get the data - Les données de l'utilisateur, issues d'un export(=dataset) depuis la plateforme LYSR, sont téléchargées.

7: Inference - Le modèle réalise l'inférence sur les données, retournant une prédiction.

8. Apply the rules - Les règles sont appliquées aux résultats.

9. Store results - Les données (input et résultats) sont enregistrées au format Parquet et uploadées sur MinIO.

Le diagramme de séquence (Figure 3.15) montre le processus de récupération du statut et des logs d'un job de backtesting en cours ou terminé sur le cluster Ray.

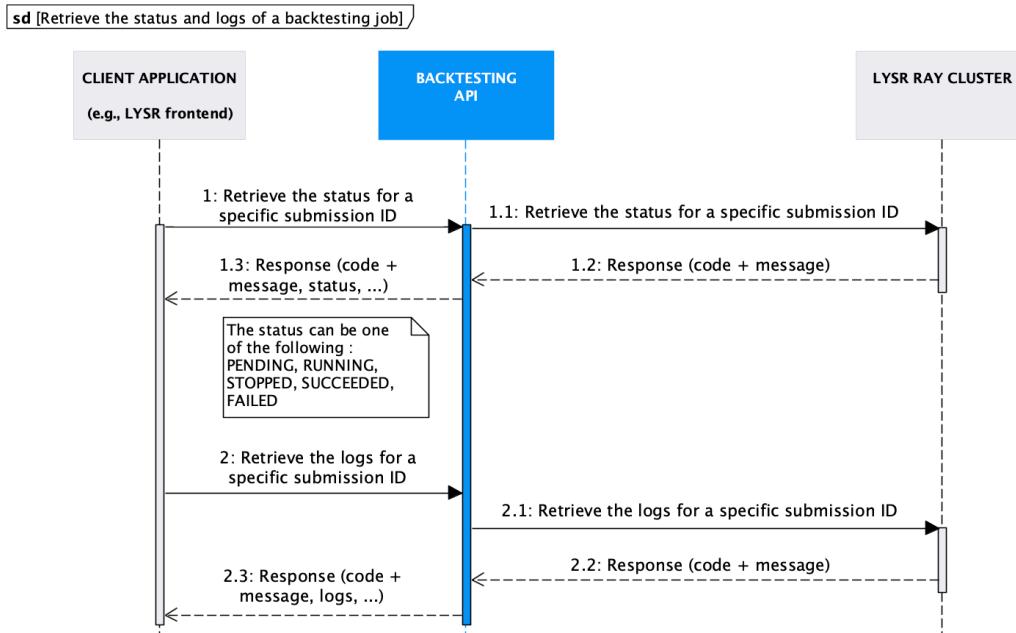


Fig. 3.15 – Diagramme de séquence - Statut et logs du backtesting

Le processus de récupération du statut ou des logs est presque identique, comme suit :

n: Retrieve the status or logs for a specific submission ID - L'utilisateur envoie une requête GET à l'API backtesting avec le submission ID du job de backtesting.

n.1: Retrieve the status or logs for a specific submission ID - L'API backtesting interroge le cluster Ray.

n.2 Response - Le cluster Ray retourne une réponse HTTP à l'API, au format JSON.

n.3: Response - L'API renvoie une réponse HTTP à l'utilisateur avec le statut ou les logs actuels du job, au format JSON.

Note : L'API de backtesting permet également de récupérer les résultats au format Parquet ainsi que le rapport HTML. Leur fonctionnement est identique à celui des logs et du statut, à l'exception du format de réponse qui diffère.

3.5 Intégration du backtesting au frontend de LYSR

Le backend présenté dans le chapitre précédent permet de lancer entre autres le backtesting et de récupérer le rapport correspondant. Une interface utilisateur intuitive est préférable à des requêtes manuelles. Le frontend doit donc répondre aux besoins des utilisateurs tout en s'intégrant harmonieusement à la plateforme LYSR.

Pour rappel, sur la plateforme LYSR, un utilisateur peut disposer d'un ou plusieurs espaces de travail (workspaces) contenant toutes les ressources liées au processus de surveillance, y compris les exports, les modèles et les règles. Les éléments de l'interface sont les suivants :

- Une liste pour sélectionner les exports (=datasets) du workspace de l'utilisateur.
- Une liste pour sélectionner les modèles du workspace de l'utilisateur.
- Une liste pour sélectionner les règles du workspace de l'utilisateur.
- Un ou plusieurs champs pour ajouter de nouvelles règles.
- Un bouton pour lancer le backtesting.
- Une section pour afficher le statut du backtesting en temps réel.
- Un bouton pour afficher les logs une fois le backtesting terminé.
- Une section pour afficher le rapport une fois le backtesting terminé.

La figure 3.16 présente une esquisse minimalistre du contenu du frontend et de ses interactions avec les autres services de LYSR.

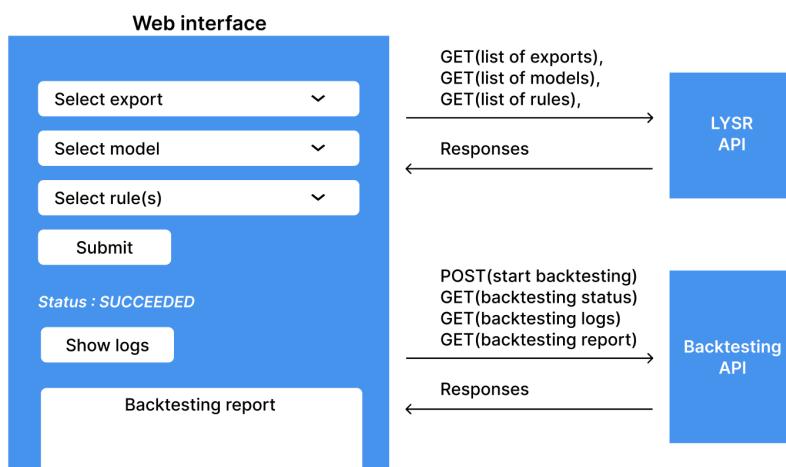


Fig. 3.16 – Frontend du backtesting

Le frontend interagit avec l'API de LYSR pour récupérer les données de l'utilisateur et avec l'API de backtesting pour effectuer les actions relatives à ce dernier. Elle devra également respecter la charte graphique de la plateforme LYSR.

3.6 Conclusion

Ce chapitre a présenté la conception des diverses composantes nécessaires à l'élaboration du système de backtesting. Il a abordé le fonctionnement de la détection d'anomalies via l'intelligence artificielle, l'implémentation du backtesting avec Ray, la structure du rapport de backtesting, ainsi que l'intégration de ces fonctionnalités aux backends et frontends de la plateforme LYSR. L'approche modulaire micro-service permet d'offrir une bonne flexibilité et l'interface devrait répondre aux besoins des utilisateurs qui utilisent le backtesting.

4 Implémentation

Dans ce chapitre, nous détaillons les différentes étapes d'implémentation du système de backtesting. Cette conception est structurée en sous-chapitres qui abordent la détection des anomalies et le développement du premier prototype, l'utilisation de Ray et MLflow pour le backtesting local, la génération de rapports de backtesting, l'intégration du backtesting au backend de LYSR, et enfin, la création d'une interface web pour le système de backtesting.

4.1 Détection des anomalies : premier prototype

L'analyse et la conception ont permis de comprendre la détection des anomalies ainsi que les métriques associées, ce qui a permis d'implémenter un premier prototype de détection d'anomalies.

Fonctionnement du prototype :

1. **Sélection et division des données** : Un jeu de données de séries temporelles contenant des anomalies est sélectionné et divisé en deux parties : 80 % pour l'entraînement et 20 % pour les tests.
2. **Préparation des données** : Le jeu de données est préparé (normalisation, sélection des caractéristiques, etc.).
3. **Entraînement du modèle** : Un modèle est entraîné à l'aide du jeu de données d'entraînement, générant en sortie un score, une probabilité ou une autre valeur en fonction du modèle.

4. **Inférence sur les données de test** : Le modèle est utilisé pour inférer sur le jeu de données de test.
5. **Détection des anomalies** : Des règles, telles que des seuils, sont appliquées aux résultats de l'inférence pour détecter les anomalies.
6. **Comparaison avec la vérité terrain** : Les résultats du modèle et des règles sont comparés à la vérité terrain.
7. **Évaluation et génération de rapport** : Des métriques (précision, rappel, F1-score, etc.) sont utilisées pour évaluer le modèle et les règles puis un rapport est généré.

L'efficacité du modèle n'est pas l'objectif principal de ce prototype. L'accent est mis sur le fonctionnement de la détection des anomalies et leur évaluation.

Un Jupyter Notebook, disponible sur GitLab dans le dossier /code/1-Prototype-Process-Miner, a permis d'implémenter les étapes décrites ci-dessus. Quatre modèles ont été entraînés, testés et évalués sur le même jeu de données « Paper Mill » provenant de l'industrie de la pâte et du papier, où les anomalies correspondent à des ruptures de papier. Voici le rapport Markdown généré à la fin du processus :

CLASSIFICATION REPORT

This document presents a comparative analysis of different models applied to the Paper Mill dataset from the pulp and paper industry. In this dataset, anomalies correspond to paper breaks.

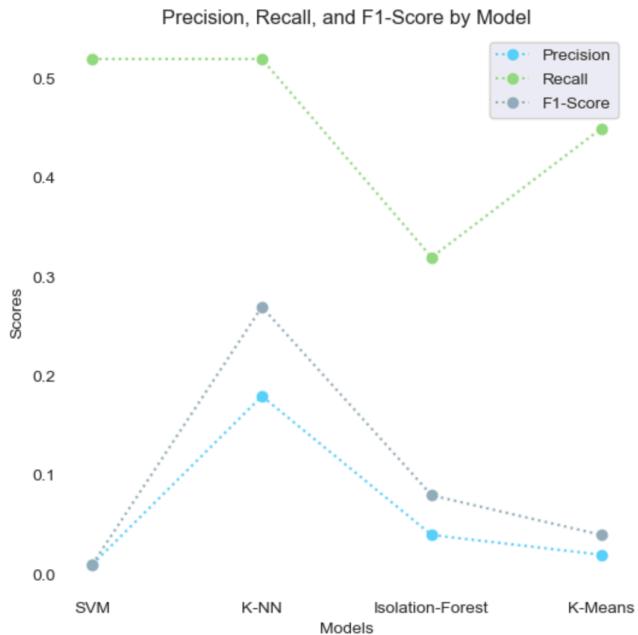
CLASSIFICATION METRICS

Here is a summary of the metrics for anomaly detection across different models. The anomalies are represented by paper breaks, while the rest of the time is considered normal.

| Model | Dataset : Paper Mill | | | | |
|------------------|----------------------|------|------|------|------|
| | P | R | F1 | FPR | FNR |
| SVM | 0.01 | 0.52 | 0.01 | 0.95 | 0.48 |
| K-NN | 0.18 | 0.52 | 0.27 | 0.03 | 0.48 |
| Isolation-Forest | 0.04 | 0.32 | 0.08 | 0.08 | 0.68 |
| K-Means | 0.02 | 0.45 | 0.04 | 0.25 | 0.55 |

Here is a graphical representation of the metrics (precision, recall, and F1-score) from the above table.

Here is a graphical representation of the metrics (precision, recall, and F1-score) from the above table.



PRECISION: It indicates the proportion of true breaks among the instances identified as breaks. A precision of 1.0 means that all instances detected as anomalies are indeed anomalies (no false positives).

RECALL: It indicates the proportion of true anomalies detected among all the anomalies present. A recall of 1.0 means that all anomalies present in the dataset have been detected (no false negatives).

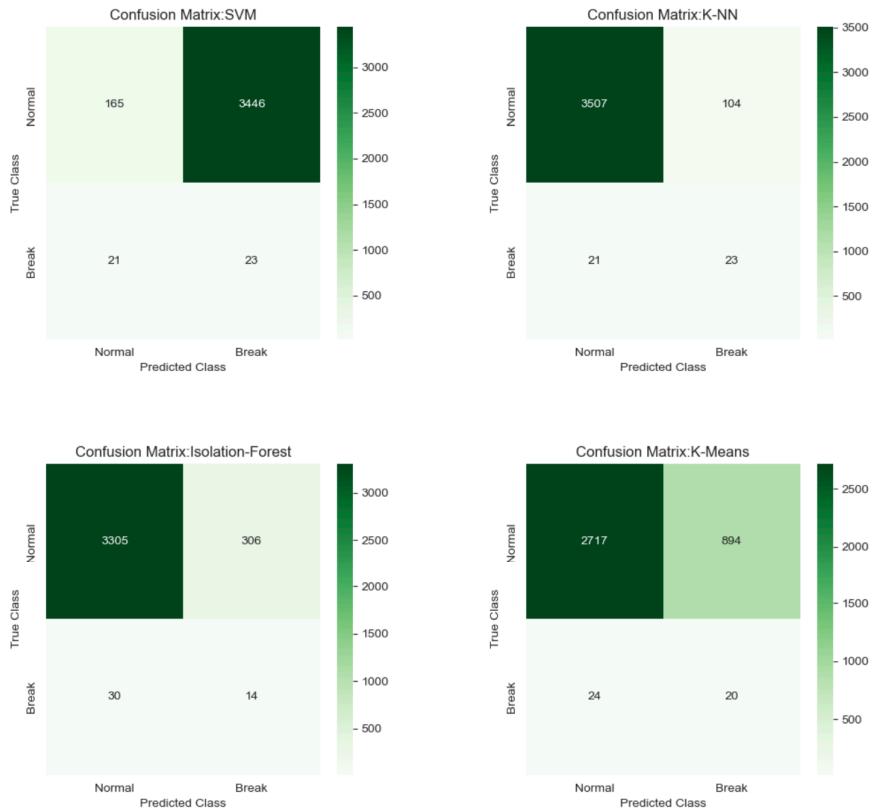
F1-SCORE: The harmonic mean of precision and recall, indicating a balance between the two. An F1-score of 1.0 indicates a perfect balance between precision and recall. An F1-score of 0.5 means that for every correct prediction, the model makes two errors (either false negatives or false positives).

FALSE POSITIVE RATE: It shows how often the model incorrectly predicts a positive class (break). A high score indicates a large number of false alarms (or false breaks).

FALSE NEGATIVE RATE: It shows how often the model incorrectly predicts a negative class (normal). A high score indicates many missed break cases.

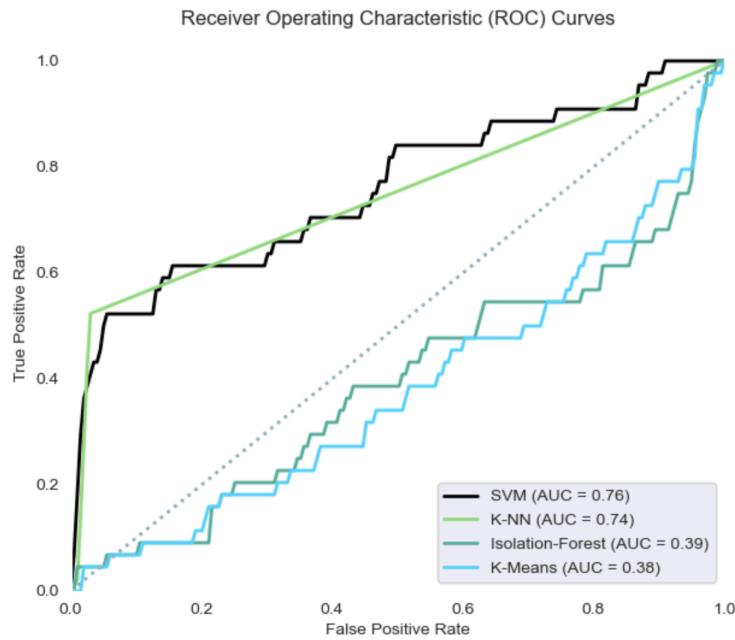
CONFUSION MATRICES

Here is the confusion matrix representing false positives, false negatives, true positives, and true negatives.



ROC CURVES

Here are the ROC curves for each of the models with the area under the curve (AUC) indicated in the legend.



ROC: The ROC (Receiver Operating Characteristic) curve illustrates the trade-off between the true positive rate and the false positive rate.

AUC: The AUC (Area Under the Curve) measures the area under this curve and provides a general indication of the model's performance. An AUC of 1.0 means that the model perfectly distinguishes anomalies from normal instances. An AUC of 0.5 indicates performance equivalent to random chance.

Fig. 4.23 – Rapport de classification

Le rapport ci-dessus compare quatre modèles de machine learning : deux modèles supervisés (SVM et K-NN) et deux modèles non supervisés (Isolation Forest et K-means), en utilisant plusieurs métriques telles que le F1-score, le rappel, la précision et l'AUC.

4.2 Backtesting local avec Ray et MLflow

L'analyse et la conception ont permis de comprendre le backtesting, Ray et MLflow, conduisant à l'implémentation d'un script Python pour exécuter des tâches de backtesting sur un cluster Ray en local. Le code est disponible dans un Jupyter Notebook sur GitLab dans le dossier /code/2-Backtesting-Local.

Fonctionnement et pseudo-code du script :

```
*****#
# Initialize the Ray cluster #
*****#
- Initialize Ray cluster

*****#
#      Retrieve the model      #
*****#
- Set MLFlow tracking URI
- Define model name and version
- Load the model from MLFlow
- Retrieve and install model dependencies
- Check if the model is stateful from its metadata

*****#
#      Retrieve the rules      #
*****#
- Load a predefined list of rules

*****#
#      Retrieve the data      #
*****#
- Define export ID, workspace ID, and bearer token
- Make a GET request to download data
- Load data using Ray (Ray Dataset)
- Check for existence of "anomaly" column (labels), store values in a Dataframe,
drop the column from Ray Dataset
- Check for existence of "timestamp" column, store values in a Dataframe, drop
the column from Ray Dataset

*****#
#          Inference          #
*****#
- Define Inference class with a model and a callable method for inference
- Set concurrency level based on whether the model is stateful or if data is
```

```
labeled
- Apply inference to the data using Ray, with specific batch size and concurrency
settings

*****#
#      Apply the rules      #
*****#
- Define function to dynamically apply rules to a batch of data
- Apply rules to data using Ray, with specific batch size and concurrency
settings

*****#
#      Store results      #
*****#
- Initialize Minio client
- Write Ray Dataset to Parquet file
- Upload Parquet file to Minio bucket
```

Description détaillée des étapes :

1. **Initialisation du cluster Ray** : Un cluster Ray est initialisé pour le traitement distribué des données.
2. **Récupération du modèle** : Le modèle ML est récupéré à partir du serveur de suivi MLflow de LYSR. Le processus vérifie également si le modèle est stateful (conserve l'état entre les prédictions) et installe les dépendances nécessaires.
3. **Récupération des règles** : Des règles simples sont chargées pour évaluer les prédictions du modèle (de 0 à N règles). Chaque règle est une chaîne de caractères, par exemple, « prediction >= 1 ».
4. **Récupération des données** : Les données (=export) sont téléchargées via l'API de LYSR et converties en Ray Dataset pour un traitement ultérieur. Une vérification est également effectuée pour déterminer si les données sont étiquetées, c'est-à-dire si elles contiennent une colonne « [...].anomaly » ainsi qu'une colonne de timestamps.
5. **Inférence** : Une classe d'inférence est définie pour effectuer des prédictions sur les données. Ray applique cette classe sur des batchs, ajustant ainsi la parallélisation en fonction de l'état du modèle et des données. L'inférence est réalisée en parallèle si le modèle est stateless et que les données ne sont pas étiquetées. En revanche, si le modèle est stateful ou que les données sont étiquetées, l'inférence se fait séquentiellement pour maintenir la cohérence et l'intégrité des résultats.

6. **Application des règles** : Une fonction est définie pour appliquer les règles aux inputs ou outputs (prédictions) du modèle afin de détecter des anomalies. Cette exécution est également gérée par Ray. Les règles actuellement évaluées sont simples (par exemple, « prediction ≥ 1 »). Si aucune règle n'est définie, cette étape est omise lors du backtesting.
8. **Stockage des résultats** : Les résultats des prédictions et des règles sont stockés dans un fichier Parquet, puis uploadés sur un serveur MinIO pour un stockage à long terme. Ces données sont utilisées ultérieurement pour générer un rapport de backtesting.

Cette partie de l'implémentation a rencontré trois problèmes distincts :

1. **Installation de Ray** : Je recommande d'installer Ray directement avec Pip plutôt qu'avec Conda. L'installation via Conda générait de nombreuses erreurs peu explicites. De plus les versions n'étaient pas les mêmes.
2. **Version de Pandas** : J'ai dû rétrograder la version de Pandas à 2.1.4, car les versions plus récentes ne possèdent plus une fonction utilisée par Ray 2.23.0. Plus de détails sont disponibles sur l'issue suivante : <https://github.com/ray-project/ray/issues/42842>.
3. **Problème avec tqdm** : Je recommande de ne pas installer le package tqdm, une barre de progression. Bien que parfois suggéré par des messages d'information, il s'avère plus gênant qu'utile avec Ray, car il masque les informations, avertissements et erreurs de Ray ainsi que certains print.

4.3 Rapport de backtesting

L'analyse et la conception ont conduit à la création d'un rapport de backtesting, avec l'implémentation d'un script permettant de le générer à partir des résultats du script de backtesting décrit précédemment. Le code est disponible dans un Jupyter Notebook sur GitLab, dans le dossier /code/4-Backtesting-Report.

Fonctionnement et pseudo-code du script :

```
#*****#
#   Generate and store      #
#   the backtesting report  #
#*****#

Function plot_timeseries:
    Create subplots based on number of rules and truths
    Plot input data on the first subplot
    If there are truths, plot them on the second subplot
    Plot each rule on subsequent subplots
    If multiple rules, plot their combined values
    Return the figure

Function plot_confusion_matrix:
    Calculate and plot confusion matrix
    Return the figure

Function plot_classification_report:
    Calculate and plot classification report
    Return the figure

Function save_combined_html:
    Open output HTML file
    Write all figures to the file
    Close the file

Main:
    Filter DataFrame to get inputs, rules, truths, and timestamps
    Initialize list for figures

    If truths exist:
        For each rule:
            Generate and store confusion matrix and classification report

    Generate and store timeseries plot
```

Save all plots into a single HTML file

Upload the HTML file to Minio bucket

Description des étapes clés :

1. **Génération d'un graphique avec les séries temporelles** : Inclure les données initiales, les anomalies réelles, les résultats des règles et la combinaison des règles si ces données existent.
2. **Matrice de confusion par règle** : Création de matrices de confusion si des règles et des données étiquetées existent.
3. **Calcul des métriques par règle** : Génération des métriques si des règles et des données étiquetées existent.
4. **Enregistrement des résultats** : Sauvegarde les graphiques, matrices et métriques dans un fichier HTML.
5. **Upload des résultats sur MinIO** : Upload le fichier HTML dans un bucket MinIO.

Les graphiques et les matrices de confusion sont générés à l'aide de la bibliothèque Plotly. Ce script a été intégré à la fin du script de backtesting afin que celui-ci génère désormais un rapport à la fin du processus.

4.4 Intégration du backtesting au backend de LYSR

L'analyse et la conception ont permis de comprendre le fonctionnement de LYSR et de concevoir de nouveaux services pour intégrer le script de backtesting présenté dans les chapitres précédents. Le code correspondant à cette partie du travail est disponible dans le dossier /code/3-Backtesting-LYSR.

Étapes de l'intégration :

1. **Adaptation du script de backtesting** : Adapter le code de backtesting local pour qu'il soit exécuté comme un job sur le cluster Ray de LYSR.
2. **Développement d'une API REST** : Développer une API REST pour initier le processus de backtesting et la déployer sur Kubernetes.
3. **Déploiement d'un object storage (MinIO)** : Déployer MinIO sur Kubernetes pour stocker les résultats du backtesting, simulant ainsi celui de LYSR.
4. **Déploiement d'un serveur de fichiers** : Déployer un serveur de fichiers sur Kubernetes pour que tout worker Ray puisse accéder au script de backtesting.
5. **CI/CD** : Automatiser l'intégration et le déploiement des nouveaux services décrits ci-dessus (2 à 4).

Ces cinq points sont détaillés dans les sections suivantes.

4.4.1 Adaptation du script de backtesting

Le backtesting, initialement développé localement sous forme de Jupyter Notebook, a été converti en script Python pour être exécuté comme un job sur le cluster Ray. Ce script accepte plusieurs arguments pour identifier un modèle, des règles et des données se trouvant sur différents services de LYSR. La commande pour lancer le script est la suivante :

```
python backtesting-script.py
    --rule "prediction >= 1" "prediction < 1"
    --export_id 46
    --model_name "175-iso_forest_model"
    --model_version 23
```

Deux problèmes ont été rencontrés lorsque le script a été exécuté sur le cluster Ray de LYSR, et non plus en local :

1. **Accessibilité des données** : Les données enregistrées dans un fichier CSV n'étaient pas accessibles par les workers Ray, car le fichier n'était pas correctement enregistré dans l'object store de Ray. La solution a été de stocker directement les données dans la mémoire partagée des workers, évitant ainsi une opération de lecture/écriture supplémentaire.
2. **Incompatibilité des versions de Python** : La version de Python utilisée pour sauvegarder le modèle (Python 3.9.19) différait de celle utilisée par le cluster Ray (Python 3.11.9), empêchant ainsi le chargement correct du modèle. Ce problème, initialement masqué par un avertissement plutôt que par une erreur, a pris du temps à être identifié mais a été résolu en harmonisant les versions de Python. Il est crucial d'utiliser la même version de Python pour enregistrer les modèles que celle utilisée par le cluster Ray.

4.4.2 Développement d'une API REST

Une API a été développée pour initier les jobs de backtesting sur le cluster Ray qui accessible uniquement depuis l'intérieur du cluster Kubernetes. Cette API, développée en Java avec Spring Boot, offre six endpoints :

- **GET /{workspaceId}/backtesting** - Vérifie le fonctionnement du service.
- **POST /{workspaceId}/backtesting** - Lance un job de backtesting.
- **GET /{workspaceId}/backtesting/{submissionId}/status** - Récupère le statut d'un job de backtesting.
- **GET /{workspaceId}/backtesting/{submissionId}/logs** - Récupère les logs d'un job de backtesting.
- **GET /{workspaceId}/backtesting/{submissionId}/results** - Récupère les résultats d'un job de backtesting.
- **GET /{workspaceId}/backtesting/{submissionId}/report** - Récupère le rapport d'un job de backtesting

L'API a été conteneurisée et déployée sur Kubernetes.

Quatre classes Java ont été implémentées :

1. **BacktestingServiceApplication.java** : Classe principale de l'application Spring Boot, exposant les endpoints pour gérer les jobs de backtesting.
2. **BacktestingRequest.java** : Modèle de requête utilisée pour initier le backtesting avec des attributs comme rules, exportId, modelName et modelVersion.
3. **KubernetesSecretsReader.java** : Lit des secrets Kubernetes.
4. **UniqueIdentifierGenerator.java** : Génère des identifiants uniques pour les jobs de backtesting.

Un problème a été rencontré concernant l'accès aux secrets Kubernetes depuis les pods. Localement, l'accès via le fichier `/.kube/config` fonctionnait bien. Cependant, dans un pod Kubernetes, des rôles spécifiques doivent être configurés, ce qui nécessitait des autorisations particulières. Finalement, il a été recommandé de passer les secrets par volume aux pods.

4.4.3 Déploiement d'un object storage (MinIO)

Un système de stockage MinIO a été déployé sur Kubernetes pour stocker les résultats et les rapports du backtesting, évitant ainsi d'utiliser celui de LYSR tout en le simulant.

4.4.4 Déploiement d'un serveur de fichiers

Un serveur de fichiers simple avec Nginx a été déployé sur Kubernetes pour récupérer le script de backtesting par tous les workers Ray. Ray nécessite l'utilisation de HTTPS pour accéder au script utilisé comme working dir, ce qui a exigé la création d'un ingress avec un certificat SSL.

4.5 Intégration du backtesting au frontend de LYSR

La conception succincte a permis de développer une interface web pour le système de backtesting. Le code correspondant à cette partie du projet est disponible dans le dossier `/code/5-Backtesting-Frontend`.

L'interface implémente tous les endpoints de l'API, à l'exception de celui permettant de récupérer les résultats. Elle permet d'initier un job de backtesting, de suivre son état

en temps réel, puis d'afficher son rapport et ses logs. L'interface est conteneurisée et déployée sur Kubernetes.

Pour des raisons de rapidité et étant donné que c'était un objectif secondaire, l'interface a été développée en VueJS, le framework que je maîtrise le mieux, contrairement à Angular utilisé par LYSR pour son frontend. Il me semblait néanmoins pertinent de créer cette interface pour regrouper et visualiser les résultats obtenus, ne serait-ce que pour une démonstration. Cela permet également de montrer comment implémenter l'API de backtesting et de proposer à LYSR une conception d'interface respectant leur charte graphique.

4.6 Conclusion

Ce chapitre a présenté l'implémentation du système de backtesting en détaillant chaque étape clé. La détection des anomalies et le développement du premier prototype ont été abordés en premier, puis le backtesting local avec Ray et MLflow a été implémenté, suivi de la génération de rapports de backtesting et de l'intégration du système de backtesting au backend de LYSR. Enfin, la création d'une interface web a été réalisée. Chaque section a mis en avant les défis rencontrés et les solutions apportées.

5 Évaluation

Dans ce chapitre, nous évaluons les différentes composantes du projet, à savoir le script de backtesting, l'API de backtesting et l'interface de backtesting. Chaque section aborde les tests effectués et les résultats obtenus. L'objectif est de s'assurer de leur bon fonctionnement et de leur intégration harmonieuse dans l'ensemble du système.

5.1 Backtesting script

Le tableau ci-dessous présente les tests effectués et leurs résultats pour le script de backtesting. Ce script, développé en Python, permet d'évaluer un ensemble de règles et de modèles sur des données temporelles, le tout étant parallélisé avec Ray. Un rapport est généré à la fin du processus.

| ID | Description | Résultats |
|----|---|-----------|
| 1 | Backtesting avec données étiquetées + 0 à N règle sur la prédiction (résultat de l'inférence) | Réussie |
| 2 | Backtesting avec données étiquetées + 0 à N règle sur une colonne de l'export (données initiales) | Échoué |
| 3 | Backtesting avec données non étiquetées + 0 à N règles sur la prédiction (résultat de l'inférence) | Réussie |
| 4 | Backtesting avec données non étiquetées + 0 à N règle sur une colonne de l'export (données initiales) | Échoué |

Les tests ont été réalisés uniquement avec un modèle stateless. Cependant, les modèles stateful devraient également fonctionner, car ils sont pris en compte dans le script de

backtesting. Les données étiquetées contiennent une colonne indiquant s'il s'agit d'une anomalie ou non.

Les tests 2 et 4 échouent parce que les noms des colonnes des exports contiennent des points (par exemple, engine.temperature). Seuls les noms de colonnes simples sont correctement interprétés par le script de backtesting. Néanmoins, l'application de règles sur les résultats de l'inférence, qui était l'objectif principal, fonctionne correctement.

5.2 Backtesting API

Le tableau suivant présente les tests effectués et leurs résultats pour l'API de backtesting. Cette API, développée en Java (Spring Boot), permet d'initier le backtesting et d'autres actions telles que la récupération des résultats ou du rapport.

| ID | Description | Résultats |
|-----|---|----------------------------|
| 1 | Accéder à un endpoint inexistant | 404 Not Found |
| 2 | Récupérer le statut de l'API | 200 OK {status: RUNNING} |
| 3 | Démarrer un job de backtesting avec un body valide* | 200 OK |
| 3.1 | Récupérer le statut du job après 0 seconde | 200 OK {status: PENDING} |
| 3.2 | Récupérer le statut du job après 10 secondes | 200 OK {status: RUNNING} |
| 3.3 | Récupérer le statut du job après 30 secondes | 200 OK {status: SUCCEEDED} |
| 3.4 | Récupérer les logs du job | 200 OK |
| 3.5 | Récupérer les résultats du job | 200 OK |
| 3.6 | Récupérer le rapport du job | 200 OK |
| 4 | Démarrer un job de backtesting avec body invalide** | 200 OK |
| 4.1 | Récupérer le statut du job après 0 seconde | 200 OK {status: PENDING} |
| 4.2 | Récupérer le statut du job après 10 secondes | 200 OK {status: FAILED} |
| 4.3 | Récupérer les logs du job | 200 Ok |
| 4.4 | Récupérer les résultats du job | 404 Not Found |
| 4.5 | Récupérer le rapport du job | 404 Not Found |

* Un body valide comprend une structure correcte et un contenu approprié, c'est-à-dire un exportId, un modèle et des règles existants.

** Un body invalide comprend une structure incorrecte ou un contenu incorrect, comme un exportId inexistant, un modèle inexistant ou des règles inexistantes.

Les tests de l'API sont positifs, les résultats obtenus sont conformes aux attentes.

5.3 Backtesting interface

Le tableau suivant présente les tests effectués et leurs résultats pour l'interface du backtesting. Ce frontend, développé avec Vue.js, permet d'initier le backtesting via une interface graphique, ainsi que de visualiser les résultats et les logs.

| ID | Description | Résultats |
|----|--|-----------|
| 1 | Afficher la liste des exports d'un workspace | Réussi |
| 2 | Afficher la liste des modèles d'un workspace | Réussi |
| 3 | Afficher la liste des règles d'un workspace | Réussi |
| 4 | Ajouter manuellement 1 à N règles | Réussi |
| 5 | Lancer le backtesting avec les options sélectionnées | Réussi |
| 6 | Afficher le statut du backtesting en direct | Réussi |
| 7 | Afficher les logs du backtesting | Réussi |
| 8 | Afficher le rapport du backtesting | Réussi |
| 9 | Relancer un nouveau backtesting | Réussi |

Les tests de l'interface sont positifs, sans problèmes à signaler.

5.4 Conclusion

L'évaluation des différentes composantes du projet révèle que le script de backtesting, l'API de backtesting et l'interface de backtesting fonctionnent globalement bien. Les tests effectués ont montré que, malgré quelques échecs liés à des détails techniques comme le format des noms de colonnes, les objectifs ont été atteints.

Tous ces tests ont été effectués manuellement. Pour respecter la philosophie DevOps, et plus précisément DevSecOps, il serait nécessaire d'automatiser ces tests. Bien que cela n'ait pas été possible durant la durée du projet, il est important de noter que des tests unitaires devraient être réalisés pour s'assurer du bon fonctionnement du code en isolant et en examinant des sections spécifiques. De plus, des tests d'intégration pourraient vérifier l'interaction entre les différents services, tels que l'API, le frontend et le cluster Ray. Enfin, des tests de sécurité sont essentiels, incluant la surveillance et la mise à jour des dépendances.

6 Améliorations

Dans ce chapitre, nous décrivons les améliorations suggérées pour le système de backtesting implémenté. Bien que de nombreuses fonctionnalités aient déjà été développées, de nouvelles fonctionnalités et améliorations pourraient être ajoutées si le temps le permettait. Nous détaillons ces suggestions pour chaque composant : le script de backtesting, l'API de backtesting et l'interface utilisateur, avec une estimation du temps et du niveau de difficulté.

6.1 Backtesting script

Le système de backtesting fonctionne actuellement avec des règles simples, telles que « prediction ≥ 1 ». Bien que cela soit suffisant pour ce travail de bachelor, il serait intéressant, dans une future itération, d'améliorer le script pour évaluer des règles plus complexes, telles que « engine.temperature < 80 » ou « prediction $\geq 1 \& lux > 10000$ ». La difficulté de cette tâche peut augmenter en fonction de la complexité des règles à évaluer. Il serait utile d'analyser si des outils existent pour faciliter ce processus.

Une autre amélioration consisterait à exécuter l'inférence et l'application des règles en parallèle pour les données étiquetées (+modèle stateless), ce qui n'est pas le cas actuellement. Pour ce faire, il faudrait utiliser une référence permettant de retrouver quelle donnée correspond à quelle vérité, même si l'ordre des résultats change. Cela pourrait être réalisé relativement facilement lors du chargement et traitement des données.

6.2 Backtesting API

L'API fonctionne bien, mais certains aspects négligés en raison du temps limité pourraient être améliorés. Voici les améliorations à effectuer avant la mise en production :

- **Validation des endpoints** : Les données fournies à l'API ne subissent actuellement pas de contrôle strict. Il faudrait vérifier la conformité des données en paramètres, dans l'en-tête ou le corps des requêtes (schéma, types, longueur, etc.). Cette validation garantit l'intégrité des données, prévient les erreurs et renforce la sécurité.
- **Gestion des erreurs** : Les exceptions sont actuellement attrapées (catch) de manière générale, ce qui rend les messages retournés assez génériques. Il serait nécessaire d'améliorer ce processus pour fournir des messages plus précis et utiles à l'utilisateur.
- **Gestion des secrets** : Les données confidentielles utilisées par le script de backtesting sont actuellement stockées dans des secrets Kubernetes. Il serait pertinent d'évaluer si ce stockage est la meilleure solution ou s'il existe des options plus efficaces.
- **Surveillance du statut des jobs Ray** : Actuellement, il faut effectuer du polling pour récupérer le statut d'un job Ray. Il serait utile d'explorer des alternatives comme les WebSockets, le long polling ou les Server-Sent Events (SSE) pour obtenir le statut des jobs de manière plus efficace.
- **Nouveau endpoint** : Un endpoint pourrait être implémenté pour récupérer la liste de tous les backtestings d'un utilisateur/workspace LYSR.

Ces améliorations ne sont pas très complexes, mais nécessitent du temps pour être bien réalisées, en particulier pour assurer la sécurité, ce qui est primordial pour une API en production.

6.3 Backtesting interface

Le frontend du backtesting fonctionne bien et simplifie les actions relatives au backtesting via une interface graphique, mais certaines fonctionnalités pourraient être ajoutées :

- **Téléchargement des résultats bruts** : Actuellement, les résultats bruts du backtesting sont enregistrés sur MinIO. Il serait intéressant d'ajouter une section permettant de télécharger ces fichiers, d'autant plus que l'endpoint pour ce téléchargement fonctionne déjà.

- **Liste de tous les backtestings :** Actuellement, il n'est pas possible d'accéder aux anciens backtestings lancés depuis l'interface. Une nouvelle section pourrait permettre de visualiser tous les backtesting et de consulter leurs résultats.
- **Comparaison des résultats :** Actuellement, le frontend permet de visualiser uniquement les résultats d'un seul backtesting. Il serait intéressant d'ajouter une section permettant de sélectionner plusieurs backtesting et de comparer leurs résultats à travers de nouvelles visualisations.

Ces tâches sont d'une difficulté modérée mais demandent du temps, surtout pour la conception des visualisations permettant de comparer les résultats du backtesting.

6.4 Conclusion

Ce chapitre a présenté les améliorations suggérées pour le système de backtesting, couvrant le script, l'API et l'interface utilisateur. Bien que le système soit déjà fonctionnel, ces améliorations visent à optimiser son efficacité et à enrichir ses fonctionnalités, assurant ainsi une meilleure expérience utilisateur et une intégration plus robuste.

7 Résultats

Dans ce chapitre, nous décrivons les résultats du système de backtesting, incluant le script de backtesting, l'API de backtesting, l'interface de backtesting, ainsi que leur intégration et déploiement continu.

7.1 Backtesting script

Le script fonctionne parfaitement sur un cluster Ray local ainsi que sur celui de LYSR. Il s'intègre également avec Mlflow, MinIO et les autres services de LYSR, facilitant ainsi la récupération des modèles, règles et données. Le script de backtesting prend en charge plusieurs fonctionnalités, y compris les modèles stateful et stateless, les données étiquetées et non étiquetées, ainsi que de 0 à N règles appliquées aux prédictions des modèles. À la fin de son exécution, un rapport de backtesting est généré. Ce rapport s'adapte aux données, par exemple, selon qu'elles soient étiquetées ou non, ou qu'il y ait zéro ou plusieurs règles. Le format HTML du rapport permet d'interagir avec les données pour une meilleure compréhension et facilite son intégration à la plateforme LYSR. Grâce à cette flexibilité, le système peut répondre à divers besoins de backtesting et permet d'interpréter efficacement les résultats.

7.2 Backtesting API

Cette section présente chaque endpoint de l'API avec une capture d'écran qui décrit la requête effectuée via Postman, ainsi que la réponse correspondante. La documentation Swagger est disponible à l'adresse suivante : <https://2024-tb-k8s-backtesting.kube.isc.heia-fr.ch/swagger-ui/index.html>.

Vérification du fonctionnement de l'API (Figure 7.24)

The screenshot shows a Postman interface with the following details:

- Method: GET
- URL: https://2024-tb-k8s-backtesting.kube.isc.heia-fr.ch/175/backtesting
- Headers: (6) - includes Content-Type: application/json
- Body: (Pretty) - displays a JSON response:

```
1 {  
2     "status": "RUNNING",  
3     "workspace": "175"  
4 }
```
- Response status: 200 OK
- Time: 138 ms
- Size: 323 B
- Buttons: Save, Share, Send

Fig. 7.24 – Requête HTTP : Vérification du fonctionnement de l'API

Description : La requête GET vérifie l'état de l'API de backtesting, avec une réponse JSON confirmant que l'API est en cours d'exécution (RUNNING).

Lancement d'un job de backtesting (Figure 7.25)

HTTP TB / RUN BACKTESTING

POST <https://2024-tb-k8s-backtesting.kube.isc.heia-fr.ch/175/backtesting>

Save Share

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

Body (raw JSON) Beautify

```

1 {
2   "rules": ["prediction >= 1", "prediction < 1"],
3   "exportId": 46,
4   "modelName": "175-iso_forest_model",
5   "modelVersion": 23,
6   "mlflowId": "02d35649-[REDACTED]-be26ced94c51"
7 }
```

Body Cookies Headers (8) Test Results Status: 200 OK Time: 137 ms Size: 376 B Save as example

Pretty Raw Preview Visualize JSON

```

1 {
2   "message": "Submitted the job successfully",
3   "submission_id": "backtesting_32DzWRo76-5c7ziW"
4 }
```

Fig. 7.25 – Requête HTTP : Lancement d'un job de backtesting

Description : La requête POST lance un job de backtesting, avec un body JSON incluant des règles, un ID d'export (=dataset), le nom et la version d'un modèle, ainsi qu'un ID Mlflow. Le token d'accès LYSR est inclus dans le header « Authorization ». La réponse JSON confirme la soumission réussie du job et retourne le submission ID.

Récupération du statut d'un job (Figure 7.26)

The screenshot shows a POSTMAN interface with the following details:

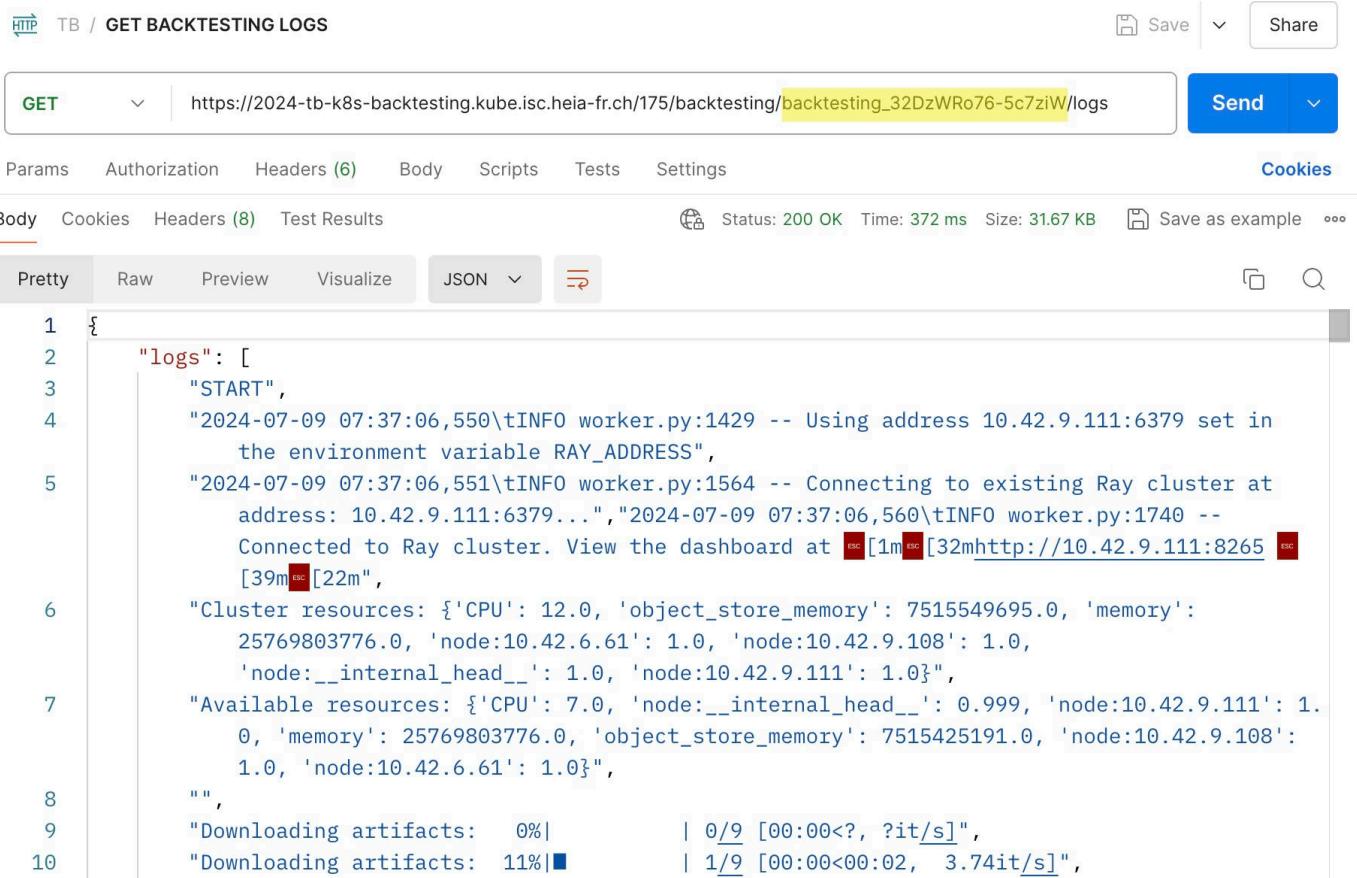
- Method: GET
- URL: https://2024-tb-k8s-backtesting.kube.isc.heia-fr.ch:175/backtesting/backtesting_32DzWRo76-5c7ziW/status
- Status: 200 OK
- Time: 43 ms
- Size: 307 B
- Headers (8) include: Content-Type: application/json; charset=UTF-8, Accept: */*, User-Agent: PostmanRuntime/7.32.1, Host: 2024-tb-k8s-backtesting.kube.isc.heia-fr.ch:175, Connection: keep-alive, Cache-Control: no-cache, Postman-Token: 16f3a2d0-1a2e-4a2b-8a2d-1a2e1a2e1a2e
- Body (Pretty):

```
1 {  
2   "status": "SUCCEEDED"  
3 }
```

Fig. 7.26 – Requête HTTP : Récupération du statut d'un job

Description : La requête GET vérifie le statut d'un job de backtesting avec une réponse JSON indiquant le statut (SUCCEEDED). Les différents statuts possibles sont « PENDING », « RUNNING », « FAILED », « STOPPED » et « SUCCEEDED ».

Récupération des logs d'un job (Figure 7.27)



The screenshot shows a POSTMAN interface with the following details:

- Method:** GET
- URL:** https://2024-tb-k8s-backtesting.kube.isc.heia-fr.ch/175/backtesting/backtesting_32DzWRo76-5c7ziW/logs
- Status:** 200 OK
- Time:** 372 ms
- Size:** 31.67 KB
- Headers (6):** Content-Type, Accept, User-Agent, Host, Connection, Content-Length
- Body:** JSON response containing logs.

```

1 {
2     "logs": [
3         "START",
4         "2024-07-09 07:37:06,550\/INFO worker.py:1429 -- Using address 10.42.9.111:6379 set in
5             the environment variable RAY_ADDRESS",
6         "2024-07-09 07:37:06,551\/INFO worker.py:1564 -- Connecting to existing Ray cluster at
7             address: 10.42.9.111:6379...","2024-07-09 07:37:06,560\/INFO worker.py:1740 --
8             Connected to Ray cluster. View the dashboard at [1m[32mhttp://10.42.9.111:8265 [39m[22m",
9         "Cluster resources: {'CPU': 12.0, 'object_store_memory': 7515549695.0, 'memory':
10             25769803776.0, 'node:10.42.6.61': 1.0, 'node:10.42.9.108': 1.0,
11             'node:__internal_head__': 1.0, 'node:10.42.9.111': 1.0}",
12         "Available resources: {'CPU': 7.0, 'node:__internal_head__': 0.999, 'node:10.42.9.111': 1.
13             0, 'memory': 25769803776.0, 'object_store_memory': 7515425191.0, 'node:10.42.9.108':
14                 1.0, 'node:10.42.6.61': 1.0}",
15         "",
16         "Downloading artifacts:  0% | 0/9 [00:00<?, ?it/s]",
17         "Downloading artifacts: 11% | 1/9 [00:00<00:02,  3.74it/s]"
18     ]
19 }

```

Fig. 7.27 – Requête HTTP : Récupération des logs d'un job

Description : La requête GET permet de récupérer les logs d'un job de backtesting, avec une réponse JSON contenant les logs des différentes étapes du processus.

Récupération des résultats d'un job (Figure 7.28)

Fig. 7.28 – Requête HTTP : Récupération des résultats d'un job

Description : La requête GET permet de récupérer les résultats d'un job de backtesting, sous forme d'un fichier Parquet. Enregistrer et ouvrir ce fichier avec un outil compatible permet d'afficher tous les résultats du backtesting.

Récupération du rapport d'un job (Figure 7.29)

The screenshot shows a browser-based interface for making HTTP requests. At the top, it says "HTTP TB / GET BACKTESTING REPORT". Below that is a search bar with "GET" selected and the URL "https://2024-tb-k8s-backtesting.kube.isc.heia-fr.ch/175/backtesting/backtesting_32DzWRo76-5c7ziW/report". To the right are "Save" and "Share" buttons. Below the search bar is a navigation bar with tabs: "Params", "Authorization", "Headers (6)", "Body", "Scripts", "Tests", "Settings", and "Cookies". The "Headers (6)" tab is selected. Under "Body", there are tabs for "Pretty", "Raw", "Preview", "Visualize", and "HTML" (which is selected). The "HTML" view displays the following code:

```
1 <html><head><meta charset="utf-8"/></head><body>
2 <div style="display: flex; flex-wrap: wrap; justify-content: space-around; width: 100%">
3 <div style="width: 50%;"><div> <script type="text/javascript">window.
4 PlotlyConfig = {MathJaxConfig: 'local'};</script>
<script charset="utf-8" src="https://cdn.plot.ly/plotly-2.32.0.min.js"></
      script> <div id="d6758104-e9a0-441d-9927-f4bd065e6ea6"
      class="plotly-graph-div" style="height:480px; width:100%;"></div> <script
      type="text/javascript"> window.PLOTLYENV=window.
```

Fig. 7.29 – Requête HTTP : Récupération du rapport d'un job

Description : La requête GET permet de récupérer le rapport HTML d'un job de back-testing.

7.3 Backtesting interface

Cette section présente l'interface web du backtesting accessible à l'adresse suivante : <https://2024-tb-k8s-backtesting-app.kube.isc.heia-fr.ch/>.

Accès à l'interface web (Figure 7.30)

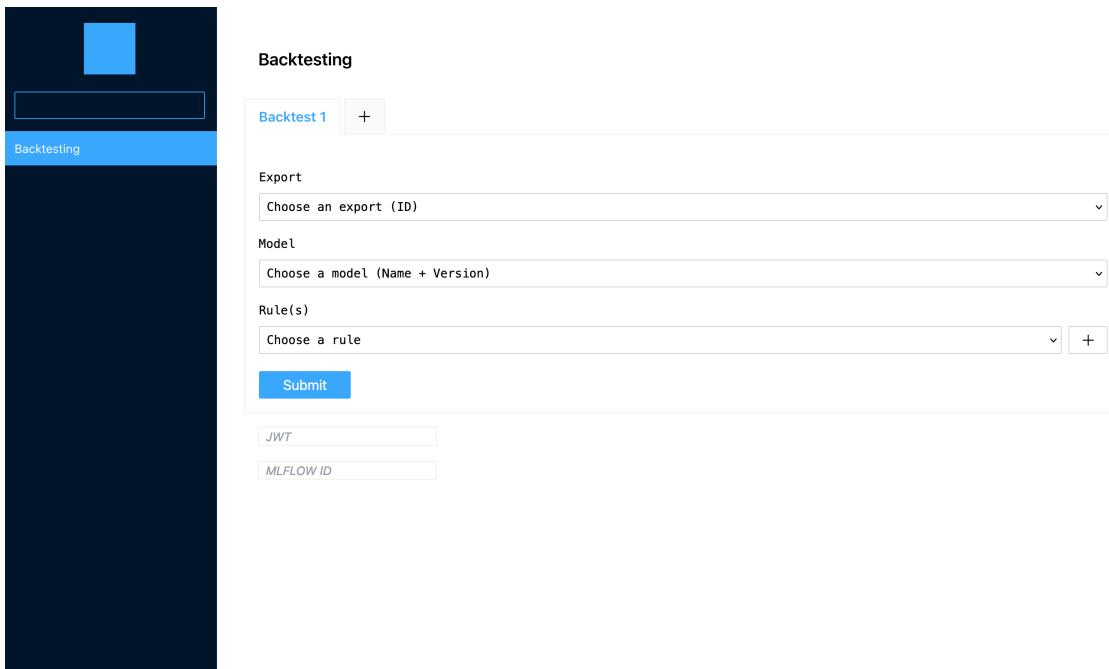


Fig. 7.30 – Interface web : Accueil

Description : L'interface permet de sélectionner les paramètres du backtesting. Cependant, avant d'accéder à la liste de ses exports, modèles et règles de son workspace LYSR, l'utilisateur doit saisir son access token et son ID MLflow dans les deux champs en bas de la page. Ces champs sont temporaires car si cette interface était intégrée à la plateforme LYSR, l'utilisateur serait déjà authentifié et ces valeurs seraient connues du frontend.

Saisie des informations de backtesting et lancement (Figure 7.31)

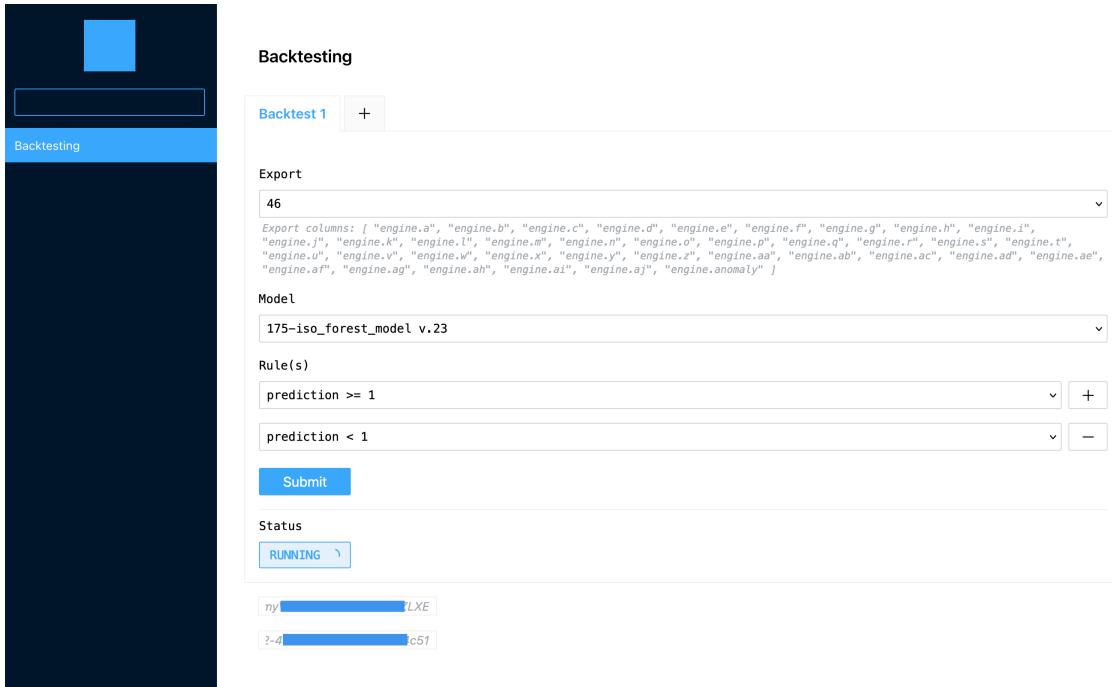
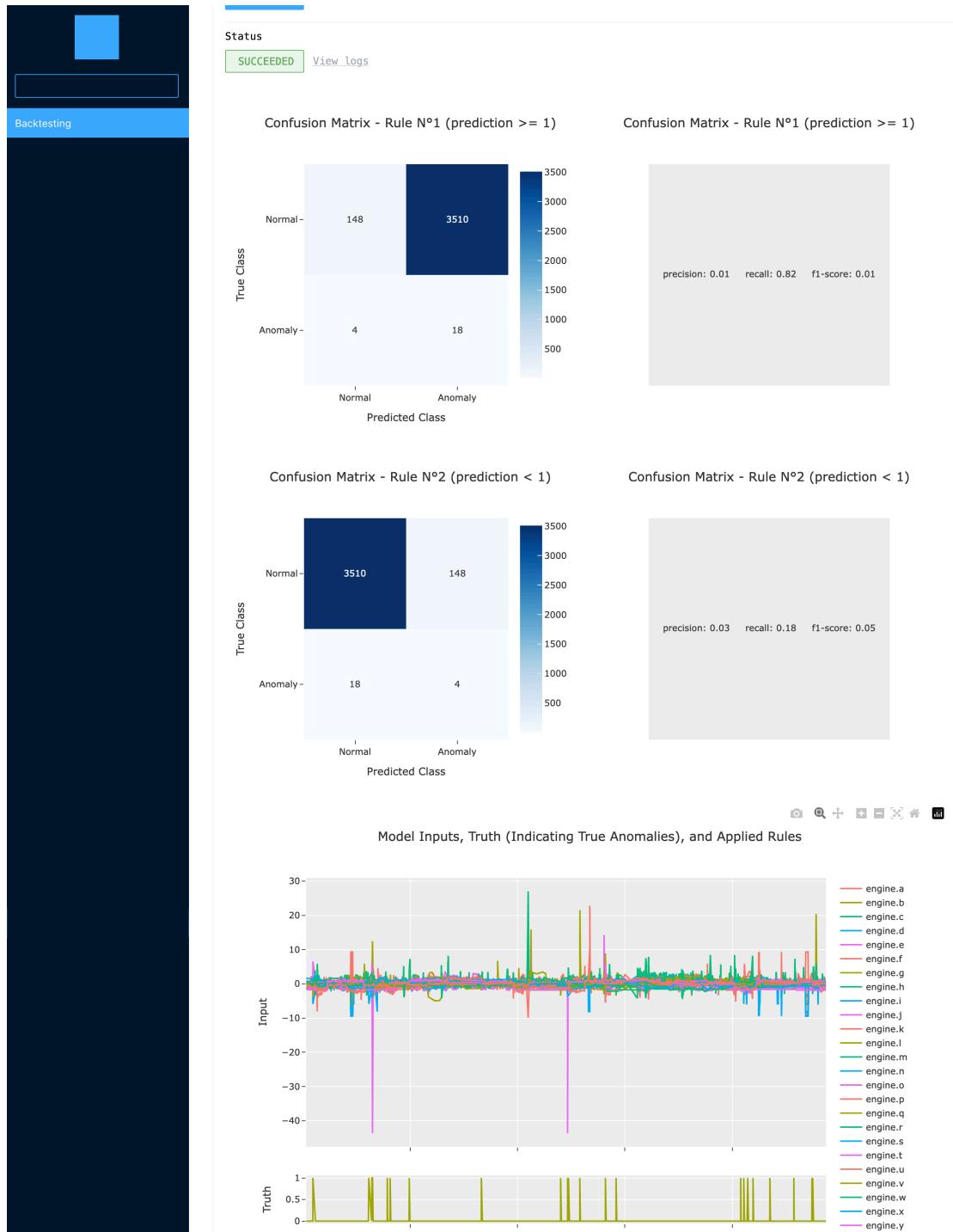


Fig. 7.31 – Interface web : Saisie des informations

Description : Une fois l'access token et l'ID MLflow saisis, l'utilisateur peut sélectionner les paramètres du backtesting, dont l'export, le modèles et 0 à N règles prédefinies ou personnalisées. Après avoir sélectionné un export, les noms de ses colonnes sont affichés sous le champ, ce qui peut être utile pour appliquer une règle à l'une d'elles. Une fois tous les paramètres définis, l'utilisateur peut lancer le backtesting et attendre la fin du job. Le statut permet de suivre la progression en direct.

Visualisation du rapport (Figure 7.35)



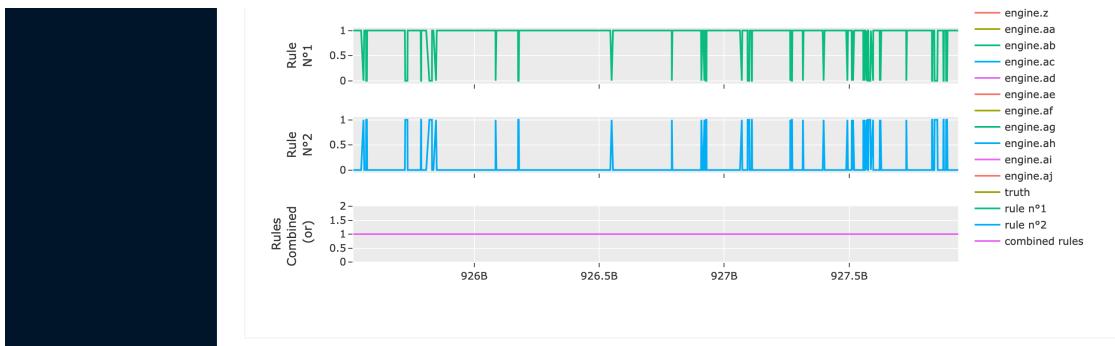


Fig. 7.35 – Interface web : Visualisation du rapport

Description : Si le job de backtesting s'est correctement déroulé, le rapport est affiché à l'utilisateur, qui peut interagir avec pour une meilleure compréhension et visualisation. Dans cet exemple, le rapport contient tous les éléments possibles, car les données sont étiquetées et il y a plus de deux règles. Dans le cas contraire, seul le graphique affichant les inputs serait présenté.

Visualisation des logs (Figure 7.36)

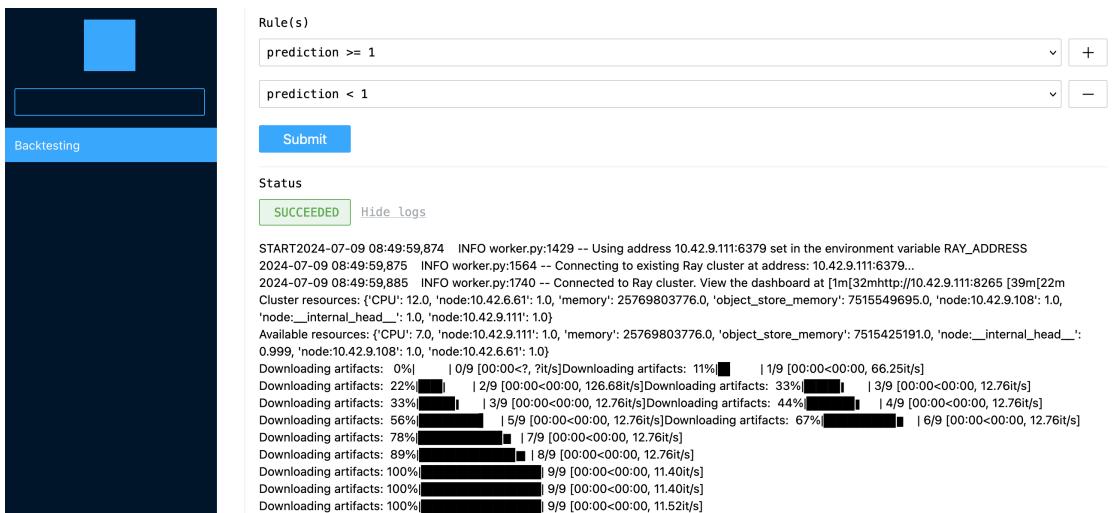


Fig. 7.36 – Interface web : Visualisation des logs

Description : Que le job de backtesting ait réussi ou non, l'utilisateur peut afficher les logs.

7.4 Déploiement de backtesting et autres services

L'API de backtesting et le frontend sont déployés sur le cluster Kubernetes de la HEIA-FR. Pour garantir leur bon fonctionnement, un système de stockage MinIO a été mis en place pour enregistrer les résultats de backtesting, simulant celui de LYSR. En outre, un serveur de fichiers a été configuré pour permettre l'accès au script de backtesting depuis n'importe quel worker de Ray.

La capture d'écran ci-dessous (Figure 7.37) présente les pods déployés sur Kubernetes.

| State | Name | Image | Ready | Restarts | IP | Node | Age |
|---------|---|---|-------|----------|-------------|-----------|-----------|
| Running | backtesting-754799dfd9-rf7kx | registry.forge.hefr.ch/adam.grossenb/tb_backtesting_kubernetes_anomaly_detection/backtesting | 1/1 | 0 | 10.42.3.224 | k8s-node1 | 32 mins |
| Running | backtesting-dependencies-78c76b5998-tqjs4 | registry.forge.hefr.ch/adam.grossenb/tb_backtesting_kubernetes_anomaly_detection/backtesting-dependencies | 1/1 | 0 | 10.42.3.221 | k8s-node1 | 57 mins |
| Running | backtesting-frontend-7d7b4c8b56-4b95z | registry.forge.hefr.ch/adam.grossenb/tb_backtesting_kubernetes_anomaly_detection/backtesting-frontend | 1/1 | 0 | 10.42.9.122 | k8s-node7 | 13 hours |
| Running | minio-6678fb4789-lhfbh | quay.io/minio/minio | 1/1 | 0 | 10.42.3.219 | k8s-node1 | 1.4 hours |

Fig. 7.37 – Pods Kubernetes

Description : Les quatre pods visibles sur l'image correspondent à l'API (backtesting), au serveur de fichiers (backtesting-dependencies), au frontend (backtesting-frontend) et au système de stockage MinIO (minio). Les images sont obtenues depuis le container registry de la HEIA-FR, à l'exception de MinIO qui utilise une image publique. En suivant les pratiques DevOps, les containers sont automatiquement build et déployés à chaque modification du code sur le dépôt GitLab, assurant ainsi une intégration continue et un déploiement continu (CI/CD).

La seule configuration manuelle requise concerne les secrets suivants (Figure 7.38) nécessaires au bon fonctionnement du backtesting.

The screenshot shows a Kubernetes Secret named "env-secret" (Active) in the namespace "2024-tb-k8s-backtesting". The secret contains the following data:

| Key | Value | Action |
|-----------------------------|---|-----------------------|
| BACKTESTING_WORKING_DIR_URI | https://2024-tb-k8s-backtesting-dependencies.kube.isc.heia-fr.ch/backtesting-script30.py.zip | <button>Copy</button> |
| LYSR_URL | https://[REDACTED] | <button>Copy</button> |
| MINIO_ACCESS_KEY | [REDACTED] | <button>Copy</button> |
| MINIO_BUCKET_NAME | backtesting-data | <button>Copy</button> |
| MINIO_SECRET_KEY | [REDACTED] | <button>Copy</button> |
| MINIO_URI | https://2024-tb-k8s-backtesting-minio-s3.kube.isc.heia-fr.ch | <button>Copy</button> |
| MLFLOW_TRACKING_URI | https://[REDACTED] | <button>Copy</button> |
| RAY_CLUSTER_URL | http://[REDACTED] | <button>Copy</button> |

Fig. 7.38 – Secrets Kubernetes

Description : Les identifiants permettant d'accéder au script de backtesting, à l'API de LYSR, à MinIO, à MLflow et au cluster Ray sont définis dans un secret de type Opaque. Ces valeurs sont ensuite lues par l'API qui les transmet comme variables d'environnement au job Ray.

7.5 Conclusion

Ce chapitre a décrit les résultats du système de backtesting, en couvrant le script, l'API, l'interface utilisateur, ainsi que leur intégration et déploiement continu. Ces résultats démontrent le bon fonctionnement du système et sa capacité à s'intégrer harmonieusement dans l'écosystème de LYSR.

8 Durabilité

« L'impact des TIC est particulièrement important en termes d'émissions globales de gaz à effet de serre (GES), de consommation d'électricité et d'épuisement de ressources et métaux rares. Au niveau mondial, le numérique représente 3 à 4% des émissions de gaz à effet de serre, lesquelles pourraient doubler voire tripler d'ici 2030, entrant ainsi en contradiction avec les objectifs climatiques de réduction des émissions de gaz à effet de serre. » [3]

Une constatation de l'OFS (mai 2024) qui va à l'encontre des dix-sept Objectifs de Développement Durable (ODD) que les États membres de l'ONU doivent atteindre d'ici 2030. [14].



Fig. 8.39 – Objectifs de développement durable [14]

8.1 Impact des TIC sur le développement durable

La fabrication des infrastructures de détection d'anomalies nécessite des matériaux rares et polluants causant des dommages environnementaux et des violations des droits humains lors de leur extraction [3], comme l'or souvent exploité dans des conditions sociales et écologiques déplorables selon WWF [4]. De plus, la gestion et le recyclage des déchets électroniques ne sont pas toujours adéquats. Les systèmes de détection d'anomalies, fonctionnant avec des modèles d'IA, consomment beaucoup d'énergie, augmentant ainsi l'empreinte carbone.

8.2 Durabilité dans le contexte de la détection d'anomalies

La détection d'anomalies peut contribuer aux industries, à la santé, aux villes et à l'environnement en se rapportant aux objectifs de développement durable suivants :

Consommation et production responsables (ODD 12)

La détection d'anomalies dans les données industrielles, comme le fait LYSR, permet une maintenance prédictive efficace, réduisant ainsi le gaspillage de ressources et améliorant l'efficacité des processus industriels. Réduire le gaspillage permet de diminuer l'extraction de matériaux potentiellement rares et nuisibles ainsi que la production et le transport nécessitant beaucoup d'énergie et de matières premières.

Bonne santé et bien-être (ODD 3)

La détection d'anomalies dans les données médicales permet l'identification précoce de maladies grâce à l'analyse de signaux vitaux, de tests de laboratoire et d'imageries médicales. En améliorant les capacités de diagnostic et de traitement rapide, cette technologie contribue significativement à la santé et au bien-être des patients.

Villes et communautés durables (ODD 11)

La surveillance des infrastructures urbaines par détection d'anomalies peut prévenir les pannes dans les systèmes de transport, l'approvisionnement en eau et les réseaux électriques. En optimisant l'utilisation des ressources et en assurant la continuité des services essentiels, cette technologie soutient le développement durable des communautés urbaines.

Lutte contre le changement climatique (ODD 13)

En analysant les données climatiques et environnementales, la détection d'anomalies aide à repérer les signes avant-coureurs de phénomènes météorologiques extrêmes ou de catastrophes naturelles. Cette capacité renforce les systèmes d'alerte précoce, permettant des réponses rapides et efficaces pour contrer la nature.

En résumé, bien que la détection d'anomalies ait de nombreuses indications, elle ne constitue pas, à ce jour, une solution pour résoudre tous les objectifs de développement durable. Il est cependant certain que, dans un proche avenir, ses domaines d'application ne vont cesser de croître.

8.3 Durabilité du système de backtesting développé

Comme mentionné précédemment, l'infrastructure informatique pose des défis en matière de développement durable, bien qu'il existe des solutions pour rendre les TIC plus durables. Par exemple, la nouvelle puce M3 d'Apple offre des performances comparables à la puce M1 tout en réduisant la consommation d'énergie de moitié [5].

Dans ce travail de bachelor, le système de backtesting développé minimise son impact grâce aux technologies utilisées, telles que Kubernetes, qui contribue à une meilleure gestion de la consommation d'énergie des serveurs grâce à ses fonctionnalités d'équilibrage de charge, de gestion des ressources et d'évolutivité automatique [6]. Le script de backtesting a été conçu pour optimiser les calculs et éviter les opérations inutiles, réduisant ainsi la consommation d'énergie. Cependant, l'infrastructure utilisée est celle de la HEIA-FR, sur laquelle je n'ai pas d'impact direct en termes de matériel et de fonctionnement. Il serait intéressant, dans une future itération, d'évaluer la consommation du système ainsi que son empreinte carbone.

9 Conclusion

En conclusion, ce travail de bachelor a permis de développer un système de backtesting pour évaluer l'efficacité des modèles d'IA dans la détection d'anomalies sur des séries temporelles. Le projet, mandaté par l'institut iCoSys et la startup LYSR, s'est articulé autour de plusieurs objectifs principaux et secondaires, allant de la conception à l'implémentation et à l'évaluation du système de backtesting.

Tous les objectifs principaux et secondaires ont été atteints. Un prototype de détection d'anomalies a été développé, un système de backtesting local utilisant Ray et MLflow a été implémenté, et l'intégration au backend de LYSR a été réalisée. De plus, les rapports de backtesting ont été générés et intégrés au frontend.

La réalisation de ce projet a nécessité une compréhension approfondie des concepts de détection d'anomalies, de backtesting ainsi que des technologies utilisées, notamment Kubernetes, Ray et MLflow.

Les résultats montrent que le système de backtesting fonctionne correctement, bien que quelques ajustements techniques soient encore nécessaires. L'intégration au backend de la plateforme LYSR et le développement d'une interface assurent une utilisation optimale par les utilisateurs. Cependant, le système, dans son état actuel, nécessite encore des modifications mineures avant une intégration à 100% avec LYSR (par exemple, la gestion des tokens et du stockage).

Des améliorations futures ont été suggérées pour optimiser l'efficacité et enrichir les fonctionnalités du système, incluant l'automatisation des tests et l'amélioration de l'interface utilisateur.

9.1 Conclusion personnelle

Ce travail de bachelor a été une expérience enrichissante et stimulante, d'autant plus qu'il s'agissait de mon premier choix de sujet. J'ai apprécié la diversité des tâches - le développement en Python, en Java et en langages web (HTML, CSS, JavaScript) - qui m'a permis de maintenir un intérêt constant et d'éviter toute impression de redondance.

Ce projet m'a permis d'acquérir de nouvelles connaissances, notamment dans le domaine de la détection d'anomalies, et d'apprendre à utiliser les technologies Ray, MLflow, Spring Boot et Typst (utilisé pour la rédaction de ce rapport). De plus, j'ai pu approfondir mes compétences en Kubernetes, Java et Python.

Je suis satisfait des résultats obtenus, mais un peu frustré de ne pas avoir eu plus de temps à disposition pour parvenir à livrer une version parfaitement aboutie.

9.2 Utilisation des outils d'intelligence artificielle (IA)

J'ai utilisé, de façon ponctuelle, des outils d'intelligence artificielle pour m'aider à reformuler ou simplifier mes textes. Bien entendu, j'ai retiré les éventuelles parties confidentielles avant de les soumettre.

Voici le prompt que j'ai utilisé avec ChatGPT :

Je travaille sur un rapport technique. J'ai besoin d'améliorer la clarté, la précision et la fluidité de mon texte sans le rendre trop complexe ou verbeux. Garde les mots en anglais tels quels, sans les traduire. Voici un extrait de mon rapport :

[Insérer le texte]

Révise cet extrait pour le rendre plus clair, concis et professionnel, tout en maintenant un ton académique approprié.

Déclaration d'honneur

Je, soussigné, Adam Grossenbacher, déclare sur l'honneur que le travail rendu est le fruit d'un travail personnel. Je certifie ne pas avoir eu recours au plagiat ou à toute autre forme de fraude. Toutes les sources d'information utilisées et les citations d'auteur ont été clairement mentionnées.

A handwritten signature in black ink, appearing to read "Adam Grossenbacher".

Glossaire

HEIA-FR : Haute école d'ingénierie et d'architecture de Fribourg

IA : Intelligence artificielle

IT : Information technology

K8s : Kubernetes

ML : Machine learning

ODD : Objectifs de développement durable

OFS : Office fédéral de la statistique

TB : Travail de bachelor

TIC : Technologies de l'information et de la communication

Bibliographie

- [1] Ray. Consulté à l'adresse <https://www.ray.io/> °
- [2] LYSR. Consulté à l'adresse <https://lysr.ch/> °
- [3] Équipements TIC et durabilité. Consulté à l'adresse <https://www.bfs.admin.ch/bfs/fr/home/statistiques/culture-medias-societe-information-sport/societe-information/indicateurs-generaux/ressources-environnement/equipements-tic-durabilite.html> °
- [4] Or Le revers de la médaille. Consulté à l'adresse <https://www.wwf.ch/fr/nos-objectifs/or-le-revers-de-la-medaille> °
- [5] Apple puces M3. Consulté à l'adresse <https://www.apple.com/chfr/newsroom/2023/10/apple-unveils-m3-m3-pro-and-m3-max-the-most-advanced-chips-for-a-personal-computer/> °
- [6] Reducing the Carbon Impact of Kubernetes with Optimization. Consulté à l'adresse <https://granulate.io/blog/carbon-impact-kubernetes-optimization/> °
- [7] Kubernetes. Consulté à l'adresse <https://kubernetes.io/> °
- [8] MLflow. Consulté à l'adresse <https://mlflow.org/> °
- [9] iCoSys. Consulté à l'adresse <https://icosys.ch/> °
- [10] Équipements TIC et durabilité. Consulté à l'adresse <https://www.swissstats.bfs.admin.ch/collection/ch.admin.bfs.swissstat.fr.issue24168242300/article/issue24168242300-01> °
- [11] 2010. *Dictionnaire Le Robert.*

- [12] 2022. Force hydraulique en Suisse. Consulté à l'adresse <https://www.admin.ch/gov/fr/accueil/documentation/communiques.msg-id-94702.html>◦
- [13] 2023. Détection des anomalies. Consulté à l'adresse <https://www.ibm.com/fr-fr/topics/anomaly-detection>◦
- [14] 2024. Dix-sept objectifs de développement durable. Consulté à l'adresse <https://www.eda.admin.ch/agenda2030/fr/home/agenda-2030/die-17-ziele-fuer-eine-nachhaltige-entwicklung.html>◦
- [15] 2024. Comment rendre compte efficacement des résultats des backtests ?. Consulté à l'adresse <https://www.linkedin.com/advice/1/how-can-you-report-backtesting-results-effectively>◦
- [16] 2024. Documentation Ray. Consulté à l'adresse <https://docs.ray.io/>◦
- [17] 2024. Documentation MinIO. Consulté à l'adresse <https://min.io/docs>◦
- [18] 2024. Prédiction des anomalies de séries chronologiques. Consulté à l'adresse <https://www.ibm.com/fr-fr/topics/anomaly-detection>◦
- [19] 2024. Détection des anomalies. Consulté à l'adresse <https://www.servicenow.com/products/it-operations-management/what-is-anomaly-detection.html>◦

Annexes

L'annexe « annexes-TB-GROSSENBACHER-Adam.pdf » est structurée de la manière suivante :

- Flyer (Résumé)
- Cahier des charges
- /code/3-Backtesting-LYSR/README.md (API Quick start)

Ces documents, ainsi que tous les autres relatifs au projet, se trouvent sur GitLab à l'adresse suivante : https://gitlab.forge.hefr.ch/adam.grossenb/tb_backtesting_kubernetes_anomaly_detection. Cela inclut le code source de chaque partie implémentée ainsi que les documents relatifs au projet (directives, PVs, figures, etc.).