École Centrale Casablanca

Conteneurisation et Déploiement
Rapport

Docker et Deep Learning

Élèves :
ARABA Joé Nestor Mintonwanou
MENDY Vincent

Encadrants :
Banouar OUMAYMA

15 février 2026

Table des matières

1 Introduction

Avec l'essor de l'Intelligence Artificielle (IA) et du Deep Learning, les projets data-driven nécessitent aujourd'hui des environnements reproductibles et portables pour garantir une mise en œuvre efficace et fiable. Docker, en tant que technologie de conteneurisation, permet de standardiser l'environnement de développement, de faciliter l'entraînement de modèles sur CPU ou GPU, et de simplifier le déploiement des applications. Dans ce contexte, ce mini-projet vise à combiner les concepts de Deep Learning et de conteneurisation pour développer un modèle d'IA complet, allant de la compréhension du problème à son déploiement via une API.

Les objectifs de ce mini-projet sont les suivants :

- Collecter et prétraiter les données nécessaires pour l'entraînement du modèle.
- Implémenter et entraîner un modèle de Deep Learning adapté à la thématique choisie (CNN, GRU, LSTM, Transformer, etc.).
- Dockeriser l'ensemble du projet pour assurer la reproductibilité et la portabilité.
- Déployer le modèle sous forme d'un service API (FastAPI) entièrement conteneurisé.
- Mettre en avant les bonnes pratiques MLOps, notamment la structuration du projet et la reproductibilité des expériences.

2 Présentation du projet

La thématique choisie pour ce mini-projet est la **détection d'anomalies** dans le cadre de la **maintenance prédictive**. Les anomalies correspondent à des comportements ou à des mesures inhabituels des équipements, pouvant indiquer un risque de panne ou un dysfonctionnement imminent. L'objectif est donc de construire et de déployer un modèle capable d'apprendre le fonctionnement normal des machines et de détecter automatiquement les situations anormales avant qu'une panne ne survienne.

Nous avons choisi cette thématique en raison de sa forte pertinence industrielle : la maintenance prédictive vise à réduire les coûts liés aux pannes tout en optimisant la durée de vie des équipements. Dans ce contexte, nous avons adopté une approche *data-driven* reposant sur des autoencodeurs, notamment basés sur des architectures LSTM et GRU. Ces modèles, grâce à leurs couches récurrentes, sont capables de capturer les dépendances temporelles présentes dans les données issues de capteurs et d'apprendre le comportement normal des machines sans supervision explicite, ce qui les rend particulièrement adaptés à la détection d'anomalies.

Par ailleurs, ce projet présente une complexité technique globale en couvrant plusieurs étapes clés du cycle MLOps : le prétraitement et la préparation des données, la conception et l'entraînement d'un modèle de Deep Learning, ainsi que son déploiement via Docker dans un environnement reproductible.

3 Méthodologie

3.1 Outils et environnement

Pour la réalisation de cette étude, plusieurs bibliothèques et outils Python ont été utilisés :

- **Pandas** : manipulation et traitement des données.
- **Keras / TensorFlow** : conception et entraînement des modèles de deep learning, notamment les autoencodeurs LSTM.
- **Scikit-learn** : prétraitement des données, normalisation, et évaluation des modèles (ex. train/test split, StandardScaler, MinMaxScaler, calcul de l'AUC).
- **Joblib** : sauvegarde et chargement des modèles.
- **Matplotlib & Seaborn** : visualisation des données et des résultats.
- **NumPy** : calculs numériques et manipulation de matrices.

L'ensemble de la pipeline a été principalement exécuté sur un ordinateur **HP Pavilion** équipé d'un processeur **11^{e} génération Intel Core™ i5-1155G7** et d'une mémoire vive de 12.0 Go.

3.2 Description du jeu de données

```
# Lecture du fichier CSV contenant les données de maintenance prédictive
df = pd.read_csv("../data/ai4i2020.csv")
df.head()  # AFFICHAGE DES 5 PREMIÈRES LIGNES DU DATASET
```

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF
0	1	M14860	M	298.1	308.6	1551	42.8	0	0	0	0	0	0	0
1	2	L47181	L	298.2	308.7	1408	46.3	3	0	0	0	0	0	0
2	3	L47182	L	298.1	308.5	1498	49.4	5	0	0	0	0	0	0
3	4	L47183	L	298.2	308.6	1433	39.5	7	0	0	0	0	0	0
4	5	L47184	L	298.2	308.7	1408	40.0	9	0	0	0	0	0	0

FIGURE 1 – Data Exploration

Le jeu de données contient 10 000 observations décrivant le fonctionnement d'une machine industrielle à partir de 14 variables. Chaque ligne possède un identifiant unique `UID` et un identifiant produit `Product ID` (identifiant produit) indiquant la qualité du produit : L (50%), M (30%) ou H (20%).

Les variables incluent notamment `Air temperature [K]` (température de l'air), `Process temperature [K]` (température du processus), `Rotational speed [rpm]` (vitesse de rotation), `Torque [Nm]` (couple moteur) et `Tool wear [min]` (usure de l'outil). Les températures sont générées à partir de processus de marche aléatoire, la vitesse de rotation est calculée à partir d'une puissance nominale avec du bruit gaussien, et le couple suit une distribution normale centrée autour de 40 Nm. L'usure de l'outil dépend du type de produit.

La variable cible `Machine failure` (défaillance machine) est un label binaire indiquant si une panne est survenue. Une défaillance est enregistrée si au moins un des modes suivants est déclenché : `Tool Wear Failure (TWF)` (défaillance par usure de l'outil), `Heat Dissipation Failure (HDF)` (défaillance thermique), `Power Failure (PWF)` (défaillance de puissance), `Overstrain Failure (OSF)` (surcontrainte mécanique) ou `Random Failure (RNF)` (panne aléatoire). La cause exacte de la panne n'est pas fournie, seule la défaillance globale est disponible pour l'apprentissage.

3.3 Analyse statistique

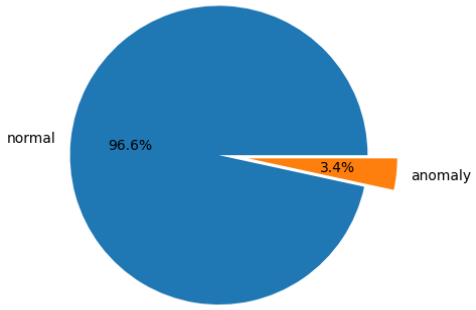
	UDI	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF
count	10000.00	10000.0	10000.00	10000.00	10000.00	10000.00	10000.00	10000.00	10000.00	10000.00	10000.00	10000.00
mean	5000.50	300.0	310.01	1538.78	39.99	107.95	0.03	0.00	0.01	0.01	0.01	0.00
std	2886.90	2.0	1.48	179.28	9.97	63.65	0.18	0.07	0.11	0.10	0.10	0.04
min	1.00	295.3	305.70	1168.00	3.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	2500.75	298.3	308.80	1423.00	33.20	53.00	0.00	0.00	0.00	0.00	0.00	0.00
50%	5000.50	300.1	310.10	1503.00	40.10	108.00	0.00	0.00	0.00	0.00	0.00	0.00
75%	7500.25	301.5	311.10	1612.00	46.80	162.00	0.00	0.00	0.00	0.00	0.00	0.00
max	10000.00	304.5	313.80	2886.00	76.60	253.00	1.00	1.00	1.00	1.00	1.00	1.00

FIGURE 2 – Statistique descriptive

On remarque que les températures sont centrées autour de **300 K** pour l'air et **310 K** pour le processus, avec des variations faibles, ce qui reflète un bruit léger autour des

valeurs nominales. La vitesse de rotation est en moyenne proche de **1 540 rpm**, mais certaines observations s'écartent fortement de cette valeur, atteignant jusqu'à **2 886 rpm**. Le couple moteur est en moyenne de **40 Nm** avec une dispersion modérée et quelques valeurs extrêmes pouvant aller jusqu'à **76 Nm**. L'usure de l'outil varie selon le type de produit, avec une moyenne d'environ 1**08 minutes** et un maximum à **253 minutes**, montrant que certains outils sont plus sollicités que d'autres.

La variable cible, `Machine failure`, est rare : seulement 3,4 % des observations correspondent à une panne. Les différents types de panne restent très peu fréquents, aucune n'étant dominante, ce qui est cohérent dans un contexte industriel où les anomalies sont naturellement rares.



Ces anomalies, bien que peu nombreuses, sont justement celles qu'il est important de détecter afin de pouvoir anticiper la maintenance et prévenir d'éventuelles pannes, illustrant ainsi l'intérêt de notre approche pour ce type de problématique industrielle.

3.4 Feature engineering et prétraitement

Avant d'entraîner le modèle, nous avons prétraité les données afin de faciliter l'apprentissage et de se concentrer sur les variables pertinentes pour la détection des anomalies.

Nous avons utilisé la colonne `UDI` comme index pour identifier de manière unique chaque observation, et la colonne `Product ID` a été supprimée car elle ne contribue pas directement à la détection des anomalies mais sert uniquement à catégoriser le produit. Nous avons également supprimées les colonnes correspondant aux différents modes de panne (`TWF`, `HDF`, `PWF`, `OSF`, `RNF`), puisque l'objectif de l'autoencodeur est d'apprendre le comportement normal à partir des données sans supervision directe sur les types de défaillances.

La variable catégorielle `Type` (L, M, H), quant à elle, a été encodée sous forme de variables indicatrices (*one-hot encoding*), en créant des colonnes binaires pour chaque catégorie tout en évitant la redondance grâce à l'option `drop_first=True` de la méthode `get_dummies` de pandas.

Après le prétraitement des données, nous avons créées plusieurs nouvelles covariables (*features*) afin de mieux représenter le comportement des machines et de capturer des interactions importantes entre les mesures existantes.

```
def build_features(df: pd.DataFrame) -> pd.DataFrame:
```

```
    logger.info("Construction des features")
    df = df.copy()

    df["Power_kw"] = (
        df["Torque [Nm]"] * df["Rotational speed [rpm]"]
    ) / 9550

    df["Temp_diff"] = (
        df["Process temperature [K]"] - df["Air temperature [K]"]
    )

    df["Speed_torque_ratio"] = (
        df["Rotational speed [rpm]"] / (df["Torque [Nm]"] + 1)
    )

    max_wear = df["Tool wear [min]"].max()

    df["Wear_level"] = pd.cut(
        df["Tool wear [min]"],
        bins=[0, 80, 160, max_wear + 1],
        labels=[0, 1, 2],
        include_lowest=True,
    )

    df["Wear_level"] = (
        pd.to_numeric(df["Wear_level"], errors="coerce")
        .fillna(0)
        .astype(int)
    )

    df["High_wear"] = (df["Tool wear [min]"] > 200).astype(int)

    df["Thermal_load"] = df["Temp_diff"] * df["Power_kw"]

    df["Mechanical_stress"] = (
        df["Torque [Nm]"] * (1 + df["Tool wear [min]"] / 250)
    )

    logger.info(
        f"Features construites - Nombre total de colonnes: {df.shape[1]}"
    )

    return df
```

— `Power_kw` : puissance mécanique développée par la machine, calculée à partir du couple moteur et de la vitesse de rotation.
— `Temp_diff` : différence de température entre le processus et l'air ambiant, reflétant la charge thermique de la machine.
— `Speed_torque_ratio` : rapport entre la vitesse de rotation et le couple appliqué, utile pour détecter des conditions de fonctionnement inhabituelles.
— `Wear_level` : usure de l'outil transformée en trois catégories correspondant à des niveaux faible, moyen et élevé.
— `High_wear` : variable binaire indiquant les cas d'usure très importante (>200 minutes).
— `Thermal_load` : combinaison de la différence de température et de la puissance

pour estimer la charge thermique.
— `Mechanical_stress` : combinaison du couple et de l'usure pour représenter la contrainte mécanique sur la machine.

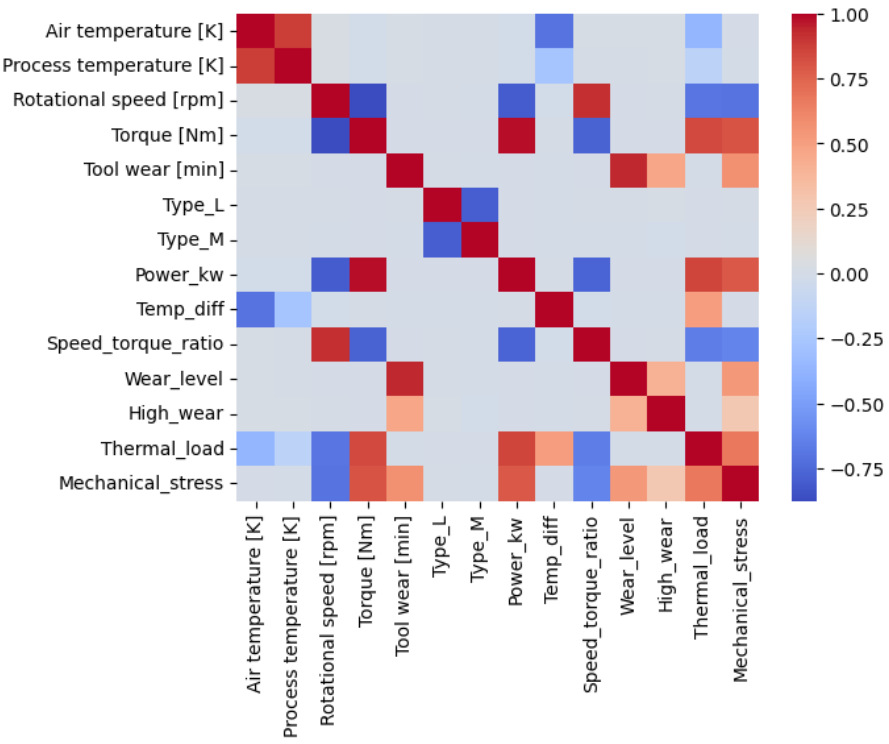


FIGURE 3 – Corrélation entre feature

Pour réduire la redondance (figure 3) et améliorer la qualité des données, une sélection des features a été réalisée en étudiant les corrélations entre elles. Les paires de variables présentant une corrélation très forte ont été identifiées, et, pour chaque paire, la variable ayant la plus faible variance a été supprimée. Après ce filtrage, le jeu de données final contient **11 features**.

3.5 Entraînement du modèle

Après séparation des données en ensembles d'entraînement et de validation (uniquement observations normales) pour éviter le surapprentissage (*overfitting*), et en ensemble de test contenant à la fois des observations normales et anormales, les données ont été transformées en *séquences temporelles* afin de permettre aux modèles d'apprendre les dépendances dans le temps. Pour chaque variable, des séquences de longueur `timesteps = 10` ont été construites à partir des observations normales de l'ensemble d'entraînement.

Deux architectures d'autoencodeurs ont été entraînées et évaluées : un `LSTM Autoencoder` et un `Conv + GRU Autoencoder`. Le schéma ci-dessous illustre la structure de chaque modèle, depuis l'entrée des séquences jusqu'à la reconstruction de la séquence d'origine.

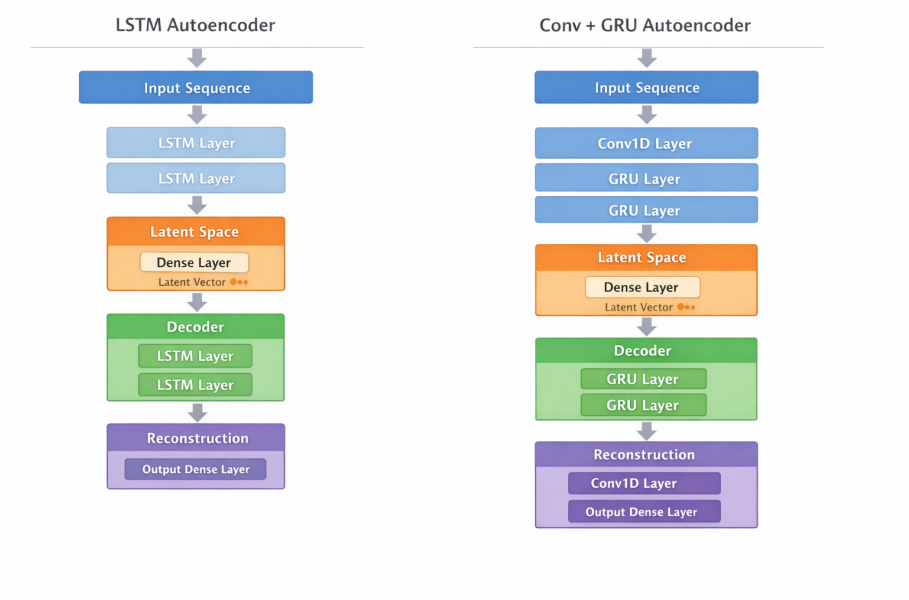


FIGURE 4 – Autoencodeurs

Les performances sur l'ensemble de test montrent que le modèle GRU fournit les meilleurs résultats parmi les architectures testées. En particulier :

- MSE moyen pour les observations normales : **0.0245 (écart-type 0.0114)**
- MSE moyen pour les anomalies : **0.0387 (écart-type 0.0174)**
- Meilleur seuil détecté : **0.0352**
- F1-Score : **0.490**
- Précision : **0.467**
- Rappel : **0.515**

On remarque que l'écart entre le MSE des observations normales et celui des anomalies n'est pas très important. Cela s'explique en partie par les contraintes matérielles : avec un PC aux ressources limitées, nous avons dû restreindre l'entraînement à des paramètres modérés, listés ci-dessous :

Paramètre	**Valeur**
Longueur des séquences (`TIMESTEPS`)	10
Nombre d'époques (`N_EPOCHS`)	100
Taille du batch (`BATCH_SIZE`)	32
Patience early stopping (`N_PATIENCE`)	5

TABLE 1 – Paramètres utilisés pour l'entraînement du modèle GRU

De plus, le jeu de données d'entraînement utilisé ne comptait que 7000 observations normales environ, ce qui reste insuffisant pour exploiter pleinement le potentiel des modèles séquentiels. Avec davantage de données, le modèle aurait pu mieux apprendre les comportements normaux et produire une séparation plus nette entre observations normales et anomalies.

Malgré ces limitations, l'architecture `Conv + GRU Autoencoder` a été retenue pour cette détection d'anomalies. Le modèle pourra ensuite être déployé dans un environnement conteneurisé pour une utilisation en production.

4 Conteneurisation avec Docker

4.1 Architecture du projet

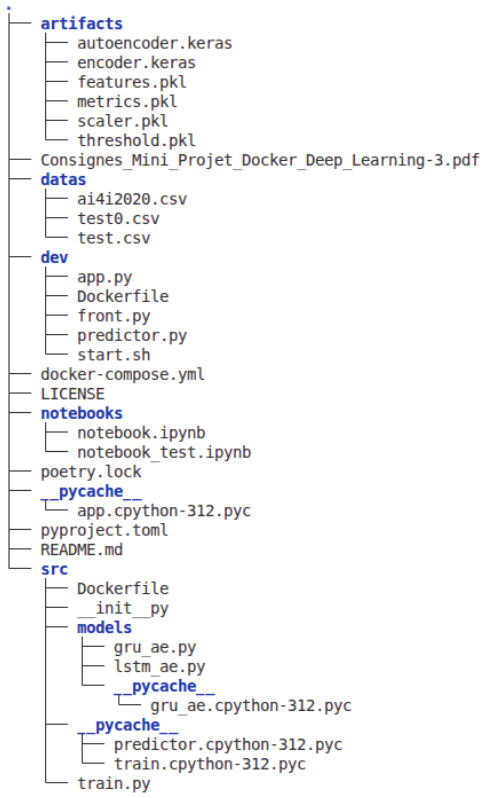


FIGURE 5 – Architecture du projet

4.2 Dockerfile et gestion des dépendances

Dans notre projet, nous avons créé deux Dockerfile distincts : l'un dédié à l'entraînement du modèle retenu et l'autre à l'exécution de l'application Streamlit (voir le code 1).

Extrait 1 – Dockerfile servant à l'entrainement

```
FROM python:3.12-slim

WORKDIR /app

RUN apt update && apt install -y git curl build-essential \
    && rm -rf /var/lib/apt/lists/*

RUN pip install poetry

# Copier les fichiers Poetry
COPY pyproject.toml poetry.lock* ./

# Installer les dependances
RUN poetry config virtualenvs.create false \
    && poetry install --with dev --no-root --no-ansi --no-directory
```

```
# Creer dossier pour artefacts
RUN mkdir -p artifacts

# Copier le reste des fichiers (code et donnees)
COPY datas/ai4i2020.csv datas/
COPY src/ /app/src/

ENV PATH="/root/.local/bin:$PATH"

VOLUME ["/app/artifacts"]

CMD ["python", "src/train.py"]
```

Extrait 2 – Utilisation de FastApi

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import List, Dict
import pandas as pd
from predictor import AnomalyPredictor

app = FastAPI()
predictor = AnomalyPredictor()

# Definir le modele de donnees attendu
class PredictRequest(BaseModel):
    rows: List[Dict]  # Liste de dictionnaires

@app.post("/predict")
def predict(request: PredictRequest):
    if not request.rows:
        raise HTTPException(status_code=400, detail="No data provided")

    df = pd.DataFrame(request.rows)
    results = predictor.predict(df)

    response = {
        "model_performance": predictor.get_metrics(),
        "n_predictions": len(results),
        "results": results,
    }

    return response
```

Pour le déploiement de l'application, nous avons conçu une architecture séparant le backend et le frontend afin de faciliter la maintenance et le déploiement. Le backend repose sur FastAPI (extrait 2) et gère toutes les requêtes liées à la prédiction et à la communication avec le modèle, tandis que le frontend utilise Streamlit pour fournir une interface utilisateur interactive et accessible (voir figure 7).

```
#!/bin/bash

cleanup() {
    echo "Stopping services..."
    if [[ -n "$UVICORN_PID" ]]; then
        kill -TERM "$UVICORN_PID" 2>/dev/null
        wait "$UVICORN_PID" 2>/dev/null
```

```
    fi
    if [[ -n "$STREAMLIT_PID" ]]; then
        kill -TERM "$STREAMLIT_PID" 2>/dev/null
        wait "$STREAMLIT_PID" 2>/dev/null
    fi
}

trap cleanup SIGINT SIGTERM

# Launch Uvicorn in background
uvicorn app:app --host 0.0.0.0 --port 5000 &
UVICORN_PID=$!
echo "Uvicorn started with PID $UVICORN_PID"

# Launch Streamlit in background
streamlit run front.py --server.port 8501 --server.address 0.0.0.0 &
STREAMLIT_PID=$!
echo "Streamlit started with PID $STREAMLIT_PID"

# Wait for both processes
wait $UVICORN_PID $STREAMLIT_PID
```

Le lancement simultané des deux serveurs est géré par le script `start.sh`. Ce script démarre d'abord Uvicorn en arrière-plan pour exécuter le backend sur le port `5000`, puis Streamlit sur le port `8501` pour le frontend.

```
FROM python:3.12-slim

WORKDIR /app

# Install system dependencies
RUN apt update && apt install -y git curl build-essential && rm -rf /var/lib/apt/lists
    /*

# Install Poetry
RUN pip install poetry

# Copy Poetry files
COPY pyproject.toml poetry.lock* /app/

# Install dependencies
RUN poetry config virtualenvs.create false \
    && poetry install --with visu,dev --no-root --no-ansi --no-directory

# Copy code and data
COPY artifacts/ /app/artifacts/
COPY src/ /app/src/
COPY dev/app.py /app/
COPY dev/front.py /app/
COPY dev/predictor.py /app/
COPY datas/ /app/datas/
COPY dev/start.sh /app/start.sh

# Add Poetry/local bin to PATH
ENV PATH="/root/.local/bin:$PATH"

# Expose ports
```

```
EXPOSE 5000 8501

RUN chmod +x /app/start.sh
CMD ["/app/start.sh"]
```

4.3 Utilisation de Docker Compose

Afin de garantir un déploiement simple, reproductible et sûr, nous avons utilisé **Docker Compose** qui assure l'orchestration des conteneurs.

```
services:

  train:
    build:
      context: .
      dockerfile: src/Dockerfile
    container_name: train-service
    volumes:
      - ./artifacts:/app/artifacts
    healthcheck:
      test: ["CMD", "test", "-f", "/app/artifacts/model.keras"]
      interval: 10s
      timeout: 5s
      retries: 10
    restart: "no"

  app:
    build:
      context: .
      dockerfile: dev/Dockerfile
    container_name: app-service
    depends_on:
      train:
        condition: service_completed_successfully
    volumes:
      - ./artifacts:/app/artifacts
    ports:
      - "5000:5000"
      - "8501:8501"
    restart: unless-stopped
```

Deux services sont ainsi définis : le service *d'entraînement* du modèle et le service de *l'application*. Le service de l'application ne démarre qu'une fois l'entraînement terminé, garantissant que le modèle est disponible. Nous avons utilisés des volumes partagés pour persister les fichiers générés lors de l'entraînement, notamment le modèle final et les artefacts associés (dossier `./artifacts`). Les ports nécessaires à l'accès au backend (*le port 5000* pour *uvicorn de FastApi*) et au frontend (*le port 8501* pour *Streamlit*) ont été exposés, afin de permettre une interaction facile avec l'application depuis l'extérieur du conteneur.

4.4 Exécution

Pour démarrer l'exécution, il suffit d'exécuter dans le terminal la commande :

```
docker compose up --build
```

en s'assurant que le moteur Docker est actif. Dans notre cas, Docker Desktop se charge de gérer le Docker Engine, ce qui permet de lancer facilement les services et de garantir leur fonctionnement correct.

4.4.1 Résultat sur terminal

```
(conteneurisation-et-d-ploiement-py3.12) etsuko@HP:~/Desktop/courses/options/Technologies IA/projects/Conteneurisation-et-D-ploiement$ docker compose up --build
Compose can now delegate builds to bake for better performance.
 To do so, set COMPOSE_BAKE=true.
[+] Building 3.2s (30/30) FINISHED                                                                              docker:desktop-linux
 => [train internal] load build definition from Dockerfile                                                                      0.0s
 => => transferring dockerfile: 646B                                                                                            0.0s
 => [app internal] load metadata for docker.io/library/python:3.12-slim                                                         1.2s
 => [train internal] load .dockerignore                                                                                         0.0s
 => => transferring context: 2B                                                                                                 0.0s
 => [app  1/14] FROM docker.io/library/python:3.12-slim@sha256:9e01bf1ae5db7649a236da7be1e94ffbbbdd7a93f867dd0d8d5720d9e1f89fab  0.1s
 => => resolve docker.io/library/python:3.12-slim@sha256:9e01bf1ae5db7649a236da7be1e94ffbbbdd7a93f867dd0d8d5720d9e1f89fab       0.0s
 => [train internal] load build context                                                                                         0.1s
 => => transferring context: 627B                                                                                               0.1s
 => CACHED [app  2/14] WORKDIR /app                                                                                             0.0s
 => CACHED [app  3/14] RUN apt update && apt install -y git curl build-essential && rm -rf /var/lib/apt/lists/*                 0.0s
 => CACHED [app  4/14] RUN pip install poetry                                                                                   0.0s
 => CACHED [train 5/9] COPY pyproject.toml poetry.lock* ./                                                                      0.0s
 => CACHED [train 6/9] RUN poetry config virtualenvs.create false && poetry install --with dev --no-root --no-ansi --no-directory 0.0s
 => CACHED [train 7/9] RUN mkdir -p artifacts                                                                                   0.0s
 => CACHED [train 8/9] COPY datas/ai4i2020.csv datas/                                                                           0.0s
 => CACHED [train 9/9] COPY src/ /app/src/                                                                                      0.0s
 => [train] exporting to image                                                                                                  0.1s
 => => exporting layers                                                                                                         0.0s
 => => exporting manifest sha256:5bad9683062836fb5ca0479fd91308762f425cc7e4685d40ae22cc4de1ad040b                               0.0s
 => => exporting config sha256:5b9c74999883ab314aad16dedcd441c8e42082edeafd237f29a72dec8f8ffb4e                                 0.0s
 => => exporting attestation manifest sha256:60ba4ce72089ac1a99836398c3ef5df3a818b8e3d1738ceed97d34b399052d5f                   0.0s
 => => exporting manifest list sha256:93399db250b8492eb95e6d20779b6bf045826465a6b94d2a2bf190a054bc413d                          0.0s
 => => naming to docker.io/library/conteneurisation-et-d-ploiement-train:latest                                                 0.0s
 => => unpacking to docker.io/library/conteneurisation-et-d-ploiement-train:latest                                              0.0s
 => [train] resolving provenance for metadata file                                                                              0.0s
 => [app internal] load build definition from Dockerfile                                                                        0.0s
 => => transferring dockerfile: 837B                                                                                            0.0s
 => [app internal] load .dockerignore                                                                                           0.0s
```

```
(conteneurisation-et-d-ploiement-py3.12) etsuko@HP:~/Desktop/courses/options/Technologies IA/projects/Conteneurisation-et-D-ploiement$ docker compose up --build
 => [app] resolving provenance for metadata file                                                                                0.0s
[+] Running 4/4
 ✔ app                      Built                                                                                               0.0s
 ✔ train                    Built                                                                                               0.0s
 ✔ Container app-service    Recreated                                                                                           4.8s
 ✔ Container train-service  Recreated                                                                                           1.8s
Attaching to app-service, train-service
train-service  | 2026-02-15 01:17:08.205786: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:31] Could not find cuda drivers on your machine, GPU will not be used.
train-service  | 2026-02-15 01:17:08.219399: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-o
ff errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
train-service  | 2026-02-15 01:17:08.926007: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical
operations.
train-service  | To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
train-service  | 2026-02-15 01:17:15.372242: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-o
ff errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
train-service  | 2026-02-15 01:17:15.375783: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:31] Could not find cuda drivers on your machine, GPU will not be used.
train-service  | 2026-02-15 01:17:21 - INFO - ============================================================
train-service  | 2026-02-15 01:17:21 - INFO - DÉMARRAGE DU PIPELINE
train-service  | 2026-02-15 01:17:21 - INFO - ============================================================
train-service  | 2026-02-15 01:17:21 - INFO - Paramètres:
train-service  | 2026-02-15 01:17:21 - INFO -   - CORRELATION_THRESHOLD: 0.92
train-service  | 2026-02-15 01:17:21 - INFO -   - TIMESTEPS: 10
train-service  | 2026-02-15 01:17:21 - INFO -   - N_EPOCHS: 100
train-service  | 2026-02-15 01:17:21 - INFO -   - BATCH_SIZE: 32
train-service  | 2026-02-15 01:17:21 - INFO -   - N_PATIENCE: 5
train-service  | 2026-02-15 01:17:21 - INFO - ============================================================
train-service  | 2026-02-15 01:17:21 - INFO - Chargement des données depuis datas/ai4i2020.csv
train-service  | 2026-02-15 01:17:21 - INFO - Données chargées - Shape: (10000, 14)
train-service  | 2026-02-15 01:17:21 - INFO - Début du prétraitement des données
train-service  | 2026-02-15 01:17:21 - INFO - Prétraitement terminé - Shape: (10000, 8)
train-service  | 2026-02-15 01:17:21 - INFO - Construction des features
train-service  | 2026-02-15 01:17:21 - INFO - Features construites - Nombre total de colonnes: 15
```

```
(conteneurisation-et-d-ploiement-py3.12) etsuko@HP:~/Desktop/courses/options/Technologies IA/projects/Conteneurisation-et-D-ploiement$ docker compose up --build
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0387 - val_loss: 0.0439 - learning_rate: 0.0010
train-service  | Epoch 21/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0390 - val_loss: 0.0440 - learning_rate: 0.0010
train-service  | Epoch 22/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0386 - val_loss: 0.0432 - learning_rate: 0.0010
train-service  | Epoch 23/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0383 - val_loss: 0.0426 - learning_rate: 0.0010
train-service  | Epoch 24/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 0.0378 - val_loss: 0.0421 - learning_rate: 0.0010
train-service  | Epoch 25/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0367 - val_loss: 0.0408 - learning_rate: 0.0010
train-service  | Epoch 26/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 0.0352 - val_loss: 0.0386 - learning_rate: 0.0010
train-service  | Epoch 27/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - loss: 0.0342 - val_loss: 0.0377 - learning_rate: 0.0010
train-service  | Epoch 28/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - loss: 0.0344 - val_loss: 0.0379 - learning_rate: 0.0010
train-service  | Epoch 29/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0330 - val_loss: 0.0368 - learning_rate: 0.0010
train-service  | Epoch 30/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - loss: 0.0334 - val_loss: 0.0376 - learning_rate: 0.0010
train-service  | Epoch 31/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 0.0336 - val_loss: 0.0373 - learning_rate: 0.0010
train-service  | Epoch 32/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0328 - val_loss: 0.0361 - learning_rate: 0.0010
train-service  | Epoch 33/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 0.0329 - val_loss: 0.0370 - learning_rate: 0.0010
train-service  | Epoch 34/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 0.0326 - val_loss: 0.0357 - learning_rate: 0.0010
train-service  | Epoch 35/100
```

```
(conteneurisation-et-d-ploiement-py3.12) etsuko@HP:~/Desktop/courses/options/Technologies IA/projects/Conteneurisation-et-D-ploiement$ docker compose up --build
train-service  | Epoch 98/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0219 - val_loss: 0.0269 - learning_rate: 1.2500e-04
train-service  | Epoch 99/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0218 - val_loss: 0.0269 - learning_rate: 1.2500e-04
train-service  | Epoch 100/100
210/210 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.0217 - val_loss: 0.0268 - learning_rate: 1.2500e-04
train-service  | Restoring model weights from the end of the best epoch: 100.
train-service  | 2026-02-15 01:22:20 - INFO - ============================================================
train-service  | 2026-02-15 01:22:20 - INFO - ENTRAÎNEMENT TERMINÉ
train-service  | 2026-02-15 01:22:20 - INFO - ============================================================
train-service  | 2026-02-15 01:22:20 - INFO - Optimisation du seuil de détection
train-service  | 2026-02-15 01:22:20 - INFO - Création de séquences avec timesteps=10
train-service  | 2026-02-15 01:22:20 - INFO - Shape des séquences: (1991, 10, 12)
train-service  | 2026-02-15 01:22:20 - INFO - Séquences de test - Normal: 1700, Anomalie: 291
train-service  | 2026-02-15 01:22:20 - INFO - Prédiction des reconstructions...
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step
train-service  | 2026-02-15 01:22:21 - INFO - MSE moyen (normal): 0.0253 (std: 0.0102)
train-service  | 2026-02-15 01:22:21 - INFO - MSE moyen (anomaly): 0.0379 (std: 0.0138)
train-service  | 2026-02-15 01:22:21 - INFO - Recherche du meilleur seuil...
train-service  | 2026-02-15 01:22:22 - INFO - ============================================================
train-service  | 2026-02-15 01:22:22 - INFO - MEILLEUR SEUIL: 0.0363
train-service  | 2026-02-15 01:22:22 - INFO - F1-Score: 0.478
train-service  | 2026-02-15 01:22:22 - INFO - Précision: 0.436
train-service  | 2026-02-15 01:22:22 - INFO - Rappel: 0.529
train-service  | 2026-02-15 01:22:22 - INFO - ============================================================
train-service  | 2026-02-15 01:22:22 - INFO - Sauvegarde des modèles...
train-service  | 2026-02-15 01:22:22 - INFO - Autoencoder sauvegardé dans artifacts/autoencoder.keras
train-service  | 2026-02-15 01:22:22 - INFO - Encoder sauvegardé dans artifacts/encoder.keras
train-service  | 2026-02-15 01:22:22 - INFO - ============================================================
train-service  | 2026-02-15 01:22:22 - INFO - PIPELINE TERMINÉ AVEC SUCCÈS
train-service  | 2026-02-15 01:22:22 - INFO - ============================================================
train-service exited with code 0
```

```
app-service    | Collecting usage statistics. To deactivate, set browser.gatherUsageStats to false.
app-service    |
app-service    |
app-service    |   You can now view your Streamlit app in your browser.
app-service    |
app-service    |   Local URL: http://localhost:8501
app-service    |   Network URL: http://172.18.0.2:8501
app-service    |   External URL: http://196.200.180.114:8501
app-service    |
app-service    | 2026-02-15 01:22:27.282912: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:31] Could not find cuda drivers on your machine, GPU will not be used.
app-service    | 2026-02-15 01:22:27.295129: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-o
ff errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
app-service    | 2026-02-15 01:22:27.850157: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical
operations.
app-service    | To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
app-service    | 2026-02-15 01:22:33.536982: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-o
ff errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
app-service    | 2026-02-15 01:22:33.539701: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:31] Could not find cuda drivers on your machine, GPU will not be used.
app-service    | 2026-02-15 01:22:39.239189: E external/local_xla/xla/stream_executor/cuda/cuda_platform.cc:51] failed call to cuInit: INTERNAL: CUDA error: Failed call to cuInit: UNKNOWN ERRO
R (303)
app-service    | INFO:     Started server process [6]
app-service    | INFO:     Waiting for application startup.
app-service    | INFO:     Application startup complete.
app-service    | INFO:     Uvicorn running on http://0.0.0.0:5000 (Press CTRL+C to quit)
app-service    | 2026-02-15 01:30:56 - INFO - Début du prétraitement des données
app-service    | 2026-02-15 01:30:56 - INFO - Prétraitement terminé - Shape: (3000, 8)
app-service    | 2026-02-15 01:30:56 - INFO - Construction des features
app-service    | 2026-02-15 01:30:56 - INFO - Features construites - Nombre total de colonnes: 15
app-service    | 2026-02-15 01:30:56 - INFO - Nettoyage des données avec seuil de corrélation: 0.9
app-service    | 2026-02-15 01:30:56 - INFO - Nombre de paires fortement corrélées: 2
app-service    | 2026-02-15 01:30:56 - INFO - Features supprimées: ['Power_kW', 'Wear_level']
app-service    | 2026-02-15 01:30:56 - INFO - Shape après nettoyage: (3000, 13)
app-service    | 2026-02-15 01:30:56 - INFO - Création de séquences avec timesteps=10
app-service    | 2026-02-15 01:30:56 - INFO - Shape des séquences: (2991, 10, 12)
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step
app-service    | INFO:     127.0.0.1:37602 - "POST /predict HTTP/1.1" 200 OK
```

4.4.2 Application Streamlit

L'application Streamlit est lancée depuis le terminal. La première étape consiste à charger les nouvelles données.

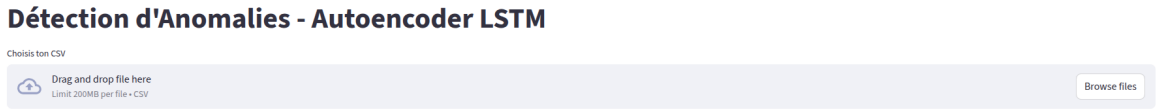


FIGURE 6 – Interface initiale de l'application

Nous avons, pour ce fait, généré deux types de jeux de données à partir de la base initiale :

1. Un jeu contenant uniquement une partie des données (fichier `test.csv` dans le dossier `data`).
2. Un jeu respectant la même distribution statistique que la base originale (fichier `test0.csv` dans le dossier `data`).

Une fois l'application lancée, il est possible de consulter les paramètres du modèle, notamment les scores de performance ainsi que le nombre d'anomalies détectées. Un tableau, identifié par le `Product Id`, indique pour chaque observation si une anomalie a été détectée.

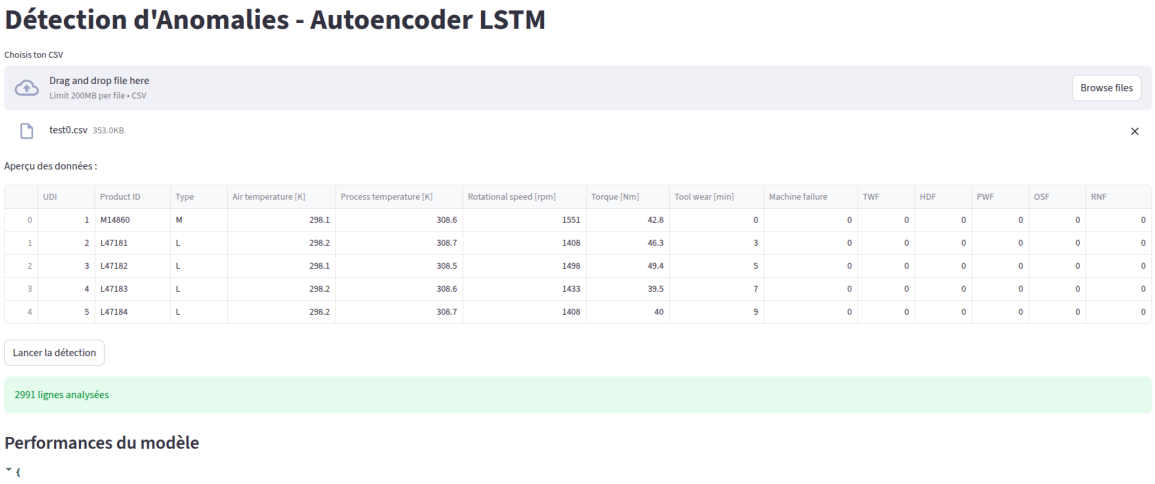


FIGURE 7 – Application streamlit

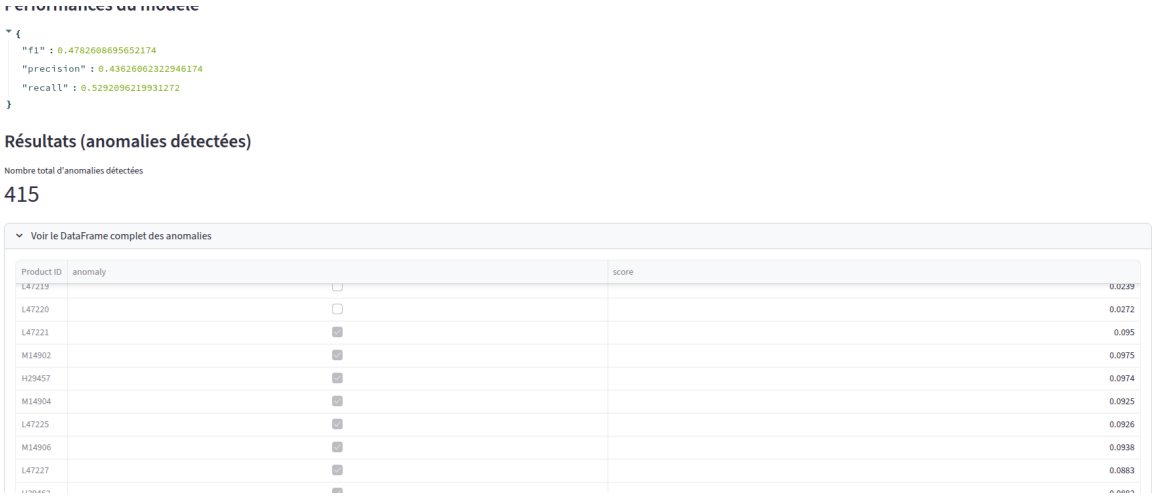


FIGURE 8 – Application streamlit

5 Conclusion et perspectives

Ce mini-projet a permis de mettre en œuvre une chaîne complète, depuis l'analyse et la préparation des données jusqu'à l'entraînement d'un modèle de Deep Learning pour la détection d'anomalies, puis son déploiement dans une application conteneurisée. L'approche par autoencodeur s'est révélée adaptée à un contexte de maintenance prédictive,

où les anomalies sont rares et où l'apprentissage se fait principalement à partir du comportement normal. Enfin, la dockerisation et l'orchestration via *Docker Compose* ont garanti la reproductibilité de l'environnement et ont simplifié l'exécution de bout en bout.

En perspective, plusieurs améliorations peuvent être envisagées. D'une part, l'évaluation pourrait être renforcée en testant d'autres métriques et stratégies de choix du seuil (par exemple, validation croisée temporelle, calibration du seuil par quantile ou méthodes bayésiennes). D'autre part, des modèles plus expressifs (Transformers temporels, autoencodeurs variationnels ou architectures hybrides) pourraient être explorés, tout en tenant compte des contraintes de calcul. Enfin, un approfondissement MLOps serait pertinent avec le suivi d'expériences, la gestion de versions des modèles, l'ajout de tests automatisés et la mise en place d'une surveillance en production (monitoring des dérives de données et de performance), afin de rendre la solution plus robuste et industrialisable.

Références

[1] UCI Machine Learning Repository, *AI4I 2020 Predictive Maintenance Dataset*, consulté le 15 février 2026. `https://archive.ics.uci.edu/dataset/601/ai4i+2020+predictive+maintenance+dataset`