N° candidat :
47250

La réduction du coût de maintenance par l'IoT et la maintenance prédictive

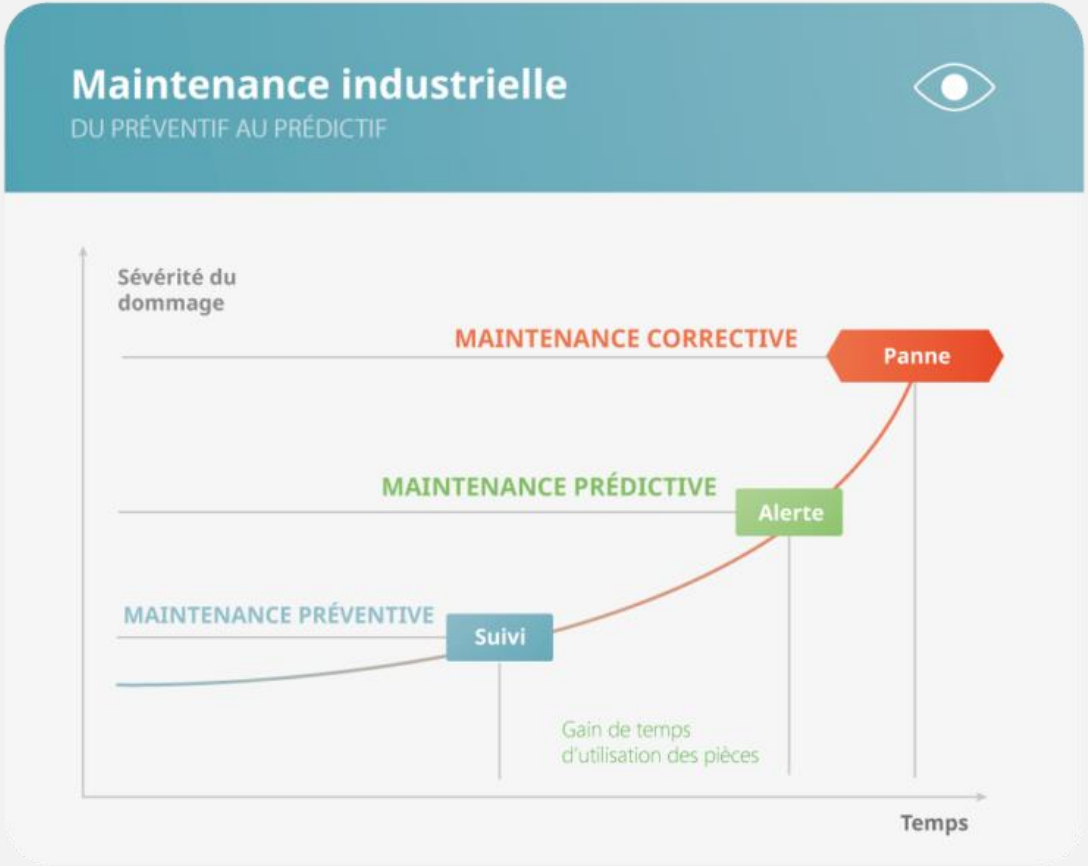
Utilisation de capteurs et IA pour analyser
les vibrations des moteurs

La maintenance traditionnelle

Coût des pannes :
22 milliards € par an en France.

90% des tâches de maintenance :
interventions d'urgence.

-10% couts maintenance = +40% ventes



Introduction à la Maintenance Prédictive

Réduire les coûts de maintenance avec l'IoT et la maintenance prédictive

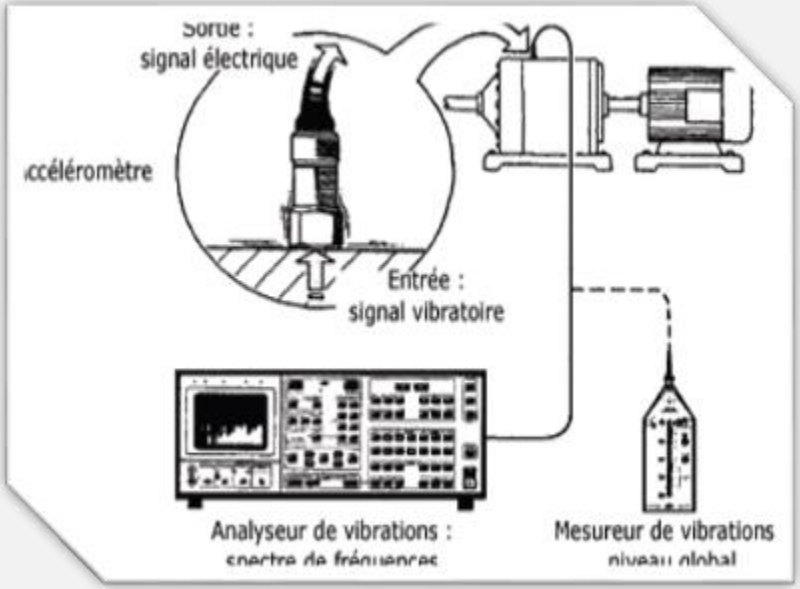
- Optimisation des coûts
  - Planification des interventions
  - Reduction du surstock de matériel
- Prévention proactive des pannes
  - Anticipation du type de défaillance
- Amélioration de la productivité
  - Suppression des temps d'arrêt offrant plus de fluidité
- Optimisation des ressources
  - Augmentation durée de vie des machines
  - Reduction des interventions humaines inutiles ou imprévues

Pourquoi et comment analyser les vibrations

Paramètre / Défaut	Température	Pression	Débit	Analyse de l'huile	Vibration
Déséquilibre					x
Désalignement ou arbre fléchi	x				x
Roulement endommagé	x			x	x
Palier lisse endommagé	x	x	x	x	x
Dentures endommagées ou usées				x	x
Jeux mécaniques excessifs					x

Tableau paramètres/défauts

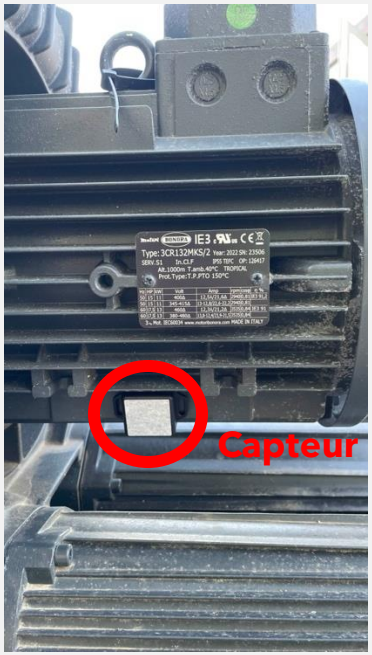
https://www.maxicours.com/se/cours/mesure-des-vibrations-introduction/



Mesure des vibrations

Mesure des paramètres de fonctionnement

- **Utilisation d'une turbine d'extraction de gaz**
- **Capteur de vibration**



Simulation de défaillance

- **Obstruction entrée/sortie**

- **Suppression phase**

Analyse des données

Vitesse de vibration sur 3 axes

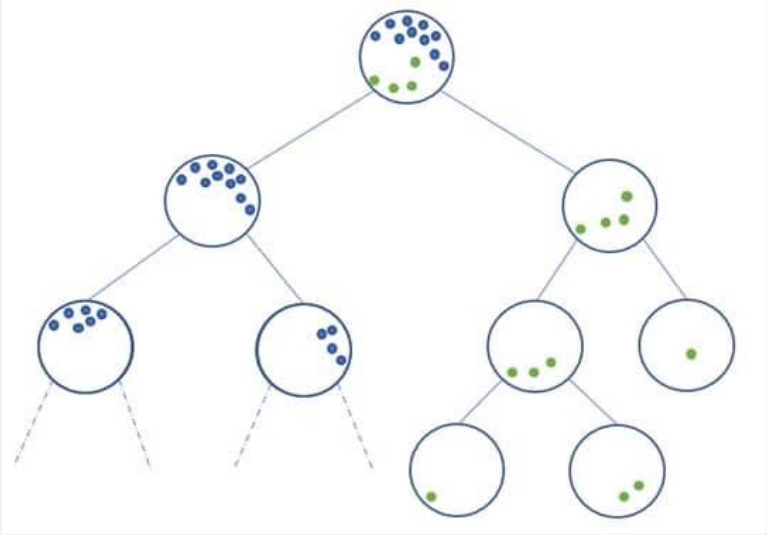
Algorithme d'identification d'anomalies

Isolation Forest :

Principe : Isole chaque point de données par des divisions aléatoires dans l'espace.

Identification des Anomalies : Les anomalies sont isolées rapidement car elles nécessitent moins de divisions.

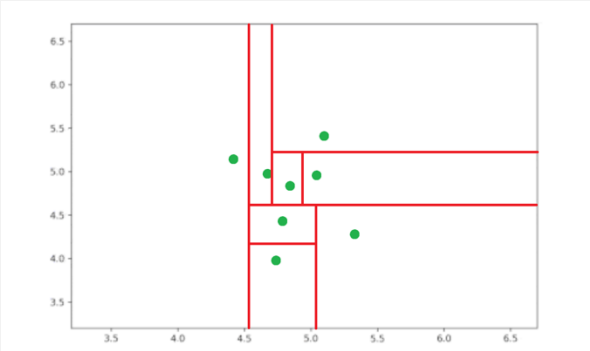
Avantages :
Rapide et efficace même pour de grands ensembles de données.

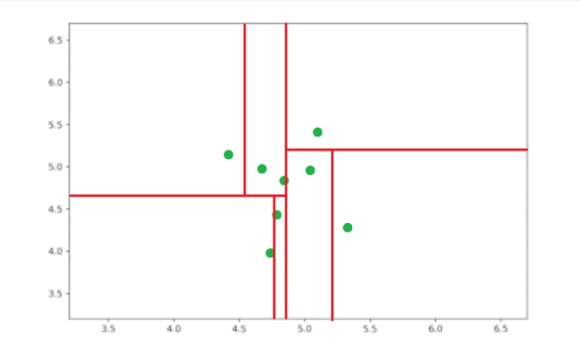


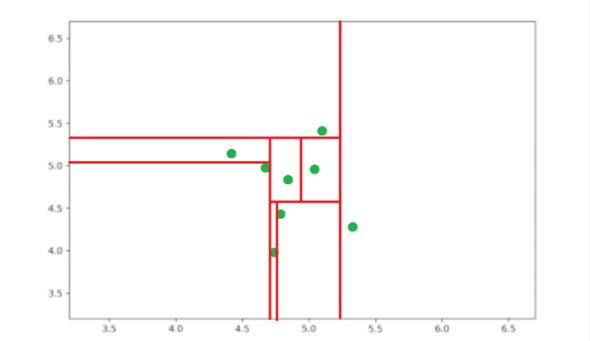
https://www.crossdata.tech/les-pannes-dans-lindustrie-4-0-lenjeu-de-la-maintenance-predictive-resolu-grace-a-lintelligence-artificielle/

Phase d'entrainement

- **Construire une forêt d'arbres de découpes.**

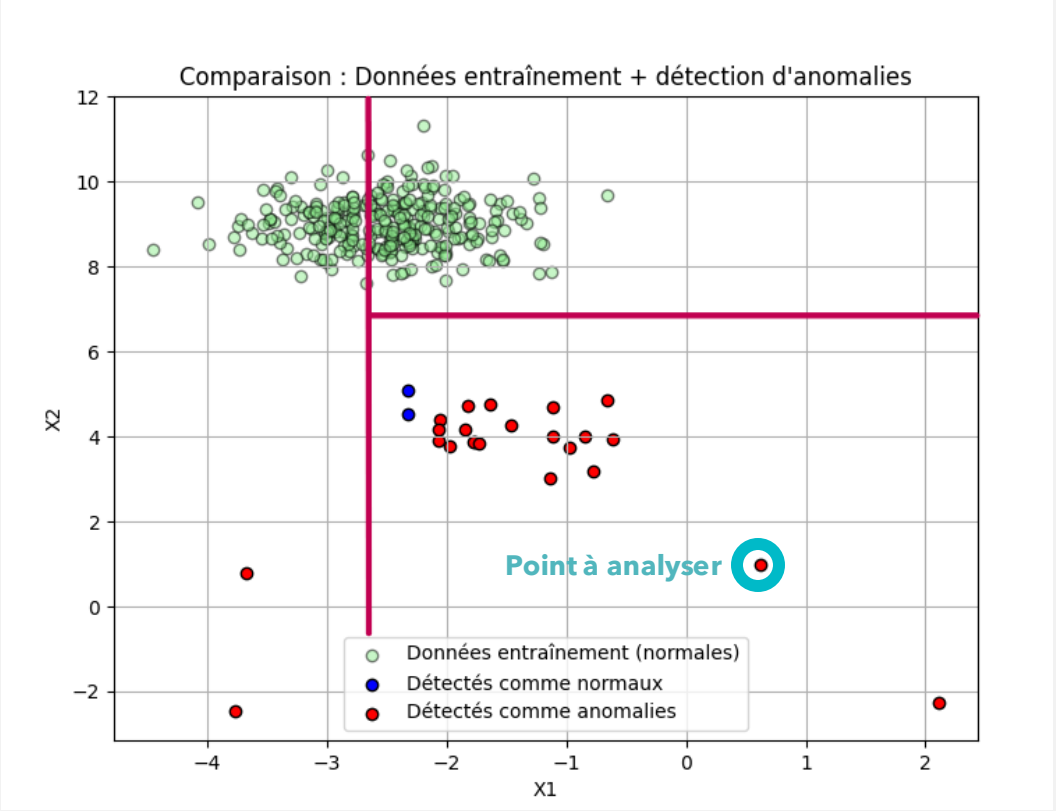




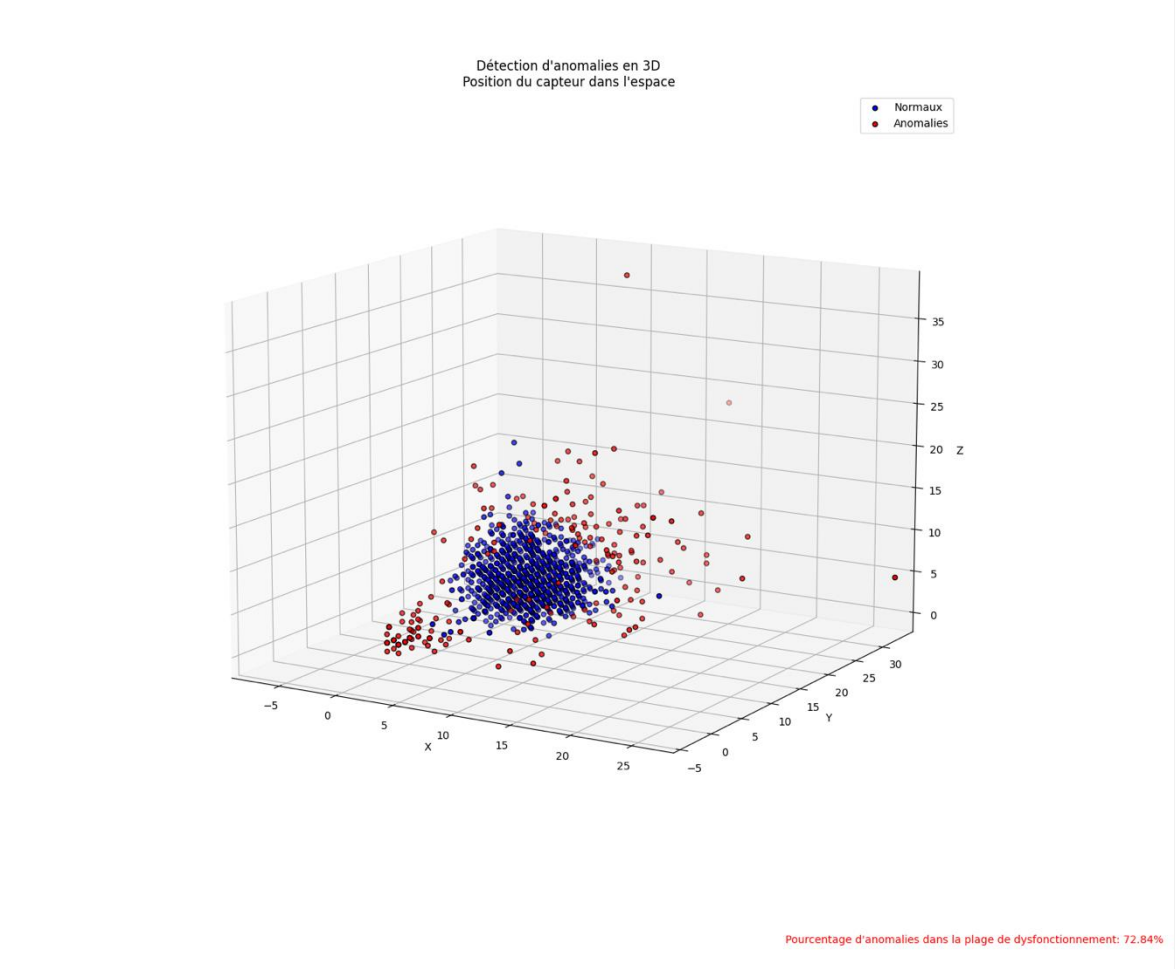


Phase de prédiction

- **Teste chaque point en calquant les slices d'entrainement.**
- **Attribue un score d'anomalies à chaque point.**
- **Score > 0.6 : anomalies**
  Sinon : normal



Visualisation de la prédiction sur nos données



Conclusion

- L’Isolation Forest permet une détection efficace des anomalies dans les vibrations moteur.
- Taux d'anomalies détectées : ~74%
- Perspectives prometteuses pour réduire les coûts de maintenance et améliorer la fiabilité.
- Temps d’exécution court par point (quelques millisecondes), adapté au traitement quasi temps réel

Perspectives futures

- **Ajout d'autres capteurs.**
- **Intégration sur les plateformes IoT industrielles.**
- **Classification des anomalies détéctées.**
- **Développer un système hybride qui apprend et s'adapte en continu.**

ANNEXES : CODE

```python
import numpy as np
import random
import math
import pickle

EULER_CONSTANT = 0.5772156649

# ISOLATION FOREST
class IsolationTree:
    def __init__(self, max_depth):
        self.max_depth = max_depth
        self.root = None

    class Node:
        def __init__(self, feature=None, split=None, left=None, right=None, size=0, is_leaf=False):
            self.feature = feature
            self.split = split
            self.left = left
            self.right = right
            self.size = size
            self.is_leaf = is_leaf

    def fit(self, X, current_depth=0):
        if current_depth >= self.max_depth or len(X) <= 1:
            return self.Node(size=len(X), is_leaf=True)

        q = random.randint(0, X.shape[1] - 1)  # dimension aleatoire
        min_val, max_val = X[:, q].min(), X[:, q].max()
        if min_val == max_val:
            return self.Node(size=len(X), is_leaf=True)

        p = random.uniform(min_val, max_val)  # seuil aleatoire

        left = X[X[:, q] < p]
        right = X[X[:, q] >= p]

        return self.Node(
            feature=q,
            split=p,
            left=self.fit(left, current_depth + 1),
            right=self.fit(right, current_depth + 1),
            size=len(X)
        )
```

```python
    def path_length(self, x, node=None, current_depth=0):
        if node is None:
            node = self.root
        if node.is_leaf:
            if node.size <= 1:
                return current_depth
            return current_depth + c_factor(node.size)
        if x[node.feature] < node.split:
            return self.path_length(x, node.left, current_depth + 1)
        else:
            return self.path_length(x, node.right, current_depth + 1)

    def train(self, X):
        self.root = self.fit(X)


class IsolationForestManual:
    def __init__(self, n_trees=100, sample_size=256):
        self.n_trees = n_trees
        self.sample_size = sample_size
        self.trees = []

    def fit(self, X):
        self.trees = []
        height_limit = math.ceil(math.log2(self.sample_size))
        for _ in range(self.n_trees):
            sample = X[np.random.choice(X.shape[0], self.sample_size, replace=False)]
            tree = IsolationTree(max_depth=height_limit)
            tree.train(sample)
            self.trees.append(tree)

    def anomaly_score(self, X):
        scores = []
        for x in X:
            path_lengths = [tree.path_length(x, tree.root) for tree in self.trees]
            avg_path_len = np.mean(path_lengths)
            score = 2 ** (-avg_path_len / c_factor(self.sample_size))
            scores.append(score)
        return np.array(scores)

    def predict(self, X, threshold=0.5):
        scores = self.anomaly_score(X)
        return (scores > threshold).astype(int)


def c_factor(n):
    if n <= 1:
        return 0
    return 2 * (math.log(n - 1) + EULER_CONSTANT) - (2 * (n - 1) / n)
```

```python
import pandas as pd
import numpy as np
import pickle
from isolation_forest import IsolationForestManual

df_train = pd.read_csv('donnees_entrainement.csv')
X_train = df_train[['X', 'Y', 'Z']].values

def generer_donnees_proches_alea(X, prob=0.5, bruit_std=0.05):
    np.random.seed(42)
    X_synthetique = []
    n_features = X.shape[1]

    for point in X:
        if np.random.rand() < prob:
            # Creation de points
            bruit = np.random.normal(0, bruit_std, size=n_features)
            nouveau_point = point + bruit
            X_synthetique.append(nouveau_point)
    return np.array(X_synthetique)


X_synthetique = generer_donnees_proches_alea(X_train, prob=0.5, bruit_std=0.05)

X_train_etendu = np.vstack([X_train, X_synthetique])

# Créer le modèle Isolation Forest
model = IsolationForestManual(n_trees=100, sample_size=64)
model.fit(X_train_etendu)

# Sauvegarde
with open('isolation_forest_model.pkl', 'wb') as f:
    pickle.dump(model, f)

print("Enregistré.")
```

```python
import pandas as pd
import pickle
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.spatial.distance import pdist, squareform
import time

# Charger le modèle entrainé
with open("isolation_forest_model.pkl", "rb") as f:
    model = pickle.load(f)

def afficher_comparaison_3D(model):
    df_test = pd.read_csv('donnees_a_predire.csv')
    X_test = df_test[['X', 'Y', 'Z']].values

    # Prédictions du modèle
    scores = model.anomaly_score(X_test)
    labels = (scores > 0.6).astype(int)  # 1 = anomalie, 0 = normal

    # Récupérer les indices des anomalies
    anomalies_indices = np.where(labels == 1)[0]

    # Plages de sysfonctionnement
    plage_1 = (1833 <= anomalies_indices) & (anomalies_indices <= 7670)
    plage_2 = (9757 <= anomalies_indices) & (anomalies_indices <= 12650)

    anomalies_dans_plage_1 = np.sum(plage_1)
    anomalies_dans_plage_2 = np.sum(plage_2)
    total_anomalies = len(anomalies_indices)

    # Calculer le pourcentage d'anomalies dans les plages
    pourcentage_plage_1 = (anomalies_dans_plage_1 / total_anomalies) * 100 if total_anomalies > 0 else 0
    pourcentage_plage_2 = (anomalies_dans_plage_2 / total_anomalies) * 100 if total_anomalies > 0 else 0
    pourcentage_plage_combinee = pourcentage_plage_2 + pourcentage_plage_1


    print(f"Pourcentage d'anomalies dans la plage 1833-7670 : {pourcentage_plage_1:.2f}%")
    print(f"Pourcentage d'anomalies dans la plage 9757-12650 : {pourcentage_plage_2:.2f}%")
    print(f"Pourcentage d'anomalies dans la plage de dysfonctionnement : {pourcentage_plage_1+pourcentage_plage_2:.2f}%")

```

```python
    # Regroupage par paquet de 10 (pour affichage plus clair)
    grouped_points = []
    grouped_labels = []

    for i in range(0, len(X_test), 10):
        group = X_test[i:i+10]
        group_labels = labels[i:i+10]

        if len(group) < 10:
            grouped_points.append(group)
            grouped_labels.append(group_labels)
        else:
            # Calcul des distances entre les points du groupe
            dist_matrix = squareform(pdist(group))
            seuil_distance = 3  # Distance pour définir la proximité
            proche = np.any(dist_matrix < seuil_distance, axis=1)

            if np.all(proche):  # Tous les points sont proches les uns des autres
                grouped_points.append(group[0:1])  # Garder un seul point du groupe
                grouped_labels.append(group_labels[0:1])  # Garder le label associé au point
            else:
                grouped_points.append(group)
                grouped_labels.append(group_labels)

    X_grouped = np.vstack(grouped_points)
    labels_grouped = np.concatenate(grouped_labels)
```

```python
    # Affichage
    fig = plt.figure(figsize=(10, 7))
    ax = fig.add_subplot(111, projection='3d')

    # Points testés normaux (en bleu)
    ax.scatter(X_grouped[labels_grouped == 0][:, 0], X_grouped[labels_grouped == 0][:, 1], X_grouped[labels_grouped == 0][:, 2],
               color='blue', edgecolors='k', label='Normaux')

    # Points annomalies (en rouge)
    ax.scatter(X_grouped[labels_grouped == 1][:, 0], X_grouped[labels_grouped == 1][:, 1], X_grouped[labels_grouped == 1][:, 2],
               color='red', edgecolors='k', label='Anomalies')

    ax.set_title("Détection d'anomalies en 3D\nPosition du capteur dans l'espace")
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")
    ax.legend()

    fig.text(0.72, 0.05, f"Pourcentage d'anomalies dans la plage de dysfonctionnement: {pourcentage_plage_combinee:.2f}%",
             ha='center', va='top', fontsize=10, color='red')

    plt.show()

afficher_comparaison_3D(model)
```

Score d'anomalies :

$$S(x_i, N) = 2^{\frac{-E(h(x_i))}{c(N)}}$$

Profondeur moyenne d'un arbre

$$c(N) = 2H_{N-1} - \frac{2(N-1)}{N}$$