Données publiques

- Stimulation d'anomalies sur données publiques

```
# Injection d'anomalies
def inject_anomalies(X, frac=0.15, seed=42):
    np.random.seed(seed)
    n = X.shape[0]
    n_anom = int(n*frac)
    idx = np.random.choice(n, n_anom, replace=False)
    X_ = X.copy()
    for i in idx:
        feats = np.random.choice(len(num_cols), size=2, replace=False)
        X_[i, feats] = X_[i, feats] * np.random.uniform(6,30)
    labels = np.zeros(n)
    labels[idx] = 1
    return X_, labels

X_test_with_anom, y_test = inject_anomalies(X_test)

input_dim = X_train.shape[1]
encoding_dim = max(4, input_dim//3)
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-1 : Injection anomalies simulées*

Définition modèle et adaptation rapide du modèle SparseMetaAE

```python
# ==========================
# 2️⃣ Sparse Autoencoder
# ==========================
def sparse_autoencoder():
    inputs = layers.Input(shape=(input_dim,))
    e = layers.Dense(encoding_dim*2, activation='relu',
                     activity_regularizer=tf.keras.regularizers.l1(1e-5))(inputs)
    e = layers.Dense(encoding_dim, activation='relu')(e)
    bottleneck = layers.Dense(encoding_dim//2, activation='relu')(e)
    d = layers.Dense(encoding_dim, activation='relu')(bottleneck)
    outputs = layers.Dense(input_dim, activation='linear')(d)
    return models.Model(inputs, outputs)

# Adaptation rapide (meta-learning)
def adapt_ae(model, X_support, lr=1e-3, steps=5):
    optimizer = tf.keras.optimizers.Adam(lr)
    X_tensor = tf.convert_to_tensor(X_support, dtype=tf.float32)
    for _ in range(steps):
        with tf.GradientTape() as tape:
            recon = model(X_tensor, training=True)
            loss = tf.reduce_mean(tf.square(recon - X_tensor))
        grads = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-2 : entrainement de l'algorithme SparseMetaAE sur données publiques*

- entrainement de l'algorithme SparseMetaAE sur données publiques

```python
# 3 Entraînement + Test
# ========================
model = sparse_autoencoder()
model.compile(optimizer='adam', loss='mse')
model.fit(X_train, X_train, epochs=30, batch_size=32, verbose=1)

support_idx = np.random.choice(X_test_with_anom.shape[0], size=10, replace=False)
X_support = X_test_with_anom[support_idx]
adapt_ae(model, X_support)

recon = model.predict(X_test_with_anom)
mse_test = np.mean(np.square(X_test_with_anom - recon), axis=1)
threshold = np.percentile(np.mean(np.square(X_train - model.predict(X_train)), axis=1), 95)
y_pred = (mse_test > threshold).astype(int)

f1_macro = f1_score(y_test, y_pred, average='macro')
recall_macro = recall_score(y_test, y_pred, average='macro')

print("=== Résultats Sparse Autoencoder ===")
print(f"F1-macro : {f1_macro:.4f}, Recall-macro : {recall_macro:.4f}")
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-3 : entrainement de l'algorithme SparseMetaAE sur données publiques*

Voici le résultat

```
1271/1271 [==============================] - 3s 2ms/step
=== Résultats Sparse Autoencoder ===
F1-macro : 0.9235, Recall-macro : 0.9704
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-4 : résultat de l'algorithme SparseMetaAE sur données publiques*

Données privées

Même procédure pour les données privées

```python
# 3 Split train/test
# ========================
X_train, X_test = train_test_split(X_all, test_size=0.3, random_state=42)
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-5 : séparation des données en train et test*

```python
# 4️⃣ Injection anomalies simulées
# ==========================
def inject_anomalies(X, frac=0.15, seed=42):
    np.random.seed(seed)
    n = X.shape[0]
    n_anom = int(n*frac)
    idx = np.random.choice(n, n_anom, replace=False)
    X_ = X.copy()
    n_num = len(num_cols)
    for i in idx:
        feats = np.random.choice(n_num, size=max(1,n_num//2), replace=False)
        X_[i, feats] = X_[i, feats] * np.random.uniform(6,30) + np.random.uniform(10,300)
    labels = np.zeros(n)
    labels[idx] = 1
    return X_, labels

X_test_with_anom, y_test = inject_anomalies(X_test)
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-6 : Injection anomalies simulées*

```python
# ==========================
# 5️⃣ Sparse Autoencoder
# ==========================
input_dim = X_train.shape[1]
encoding_dim = max(4, input_dim // 3)

def sparse_meta_ae():
    inputs = layers.Input(shape=(input_dim,))
    e = layers.Dense(encoding_dim*2, activation='relu', activity_regularizer=tf.keras.regularizers.l1(1e-5))(inputs)
    e = layers.Dense(encoding_dim, activation='relu')(e)
    bottleneck = layers.Dense(encoding_dim//2, activation='relu')(e)
    d = layers.Dense(encoding_dim, activation='relu')(bottleneck)
    outputs = layers.Dense(input_dim, activation='linear')(d)
    return models.Model(inputs, outputs)
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-7 : entrainement de l'algorithme SparseMetaAE sur données privées*

```python
# ==========================
# 6️⃣ Adaptation rapide (Meta-Learning)
# ==========================
def adapt_ae(model, X_support, lr=1e-3, steps=5):
    optimizer = tf.keras.optimizers.Adam(learning_rate=lr)
    X_tensor = tf.convert_to_tensor(X_support, dtype=tf.float32)
    for _ in range(steps):
        with tf.GradientTape() as tape:
            recon = model(X_tensor, training=True)
            loss = tf.reduce_mean(tf.square(recon - X_tensor))
        grads = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-8 : entrainement de l'algorithme SparseMetaAE sur données privées*

```python
# 7️⃣ Entraînement + Test
# ========================
model = sparse_meta_ae()
model.compile(optimizer='adam', loss='mse')
model.fit(X_train, X_train, epochs=30, batch_size=32, verbose=1)

support_idx = np.random.choice(X_test_with_anom.shape[0], size=10, replace=False)
X_support = X_test_with_anom[support_idx]
adapt_ae(model, X_support)

recon = model.predict(X_test_with_anom)
mse_test = np.mean(np.square(X_test_with_anom - recon), axis=1)
threshold = np.percentile(np.mean(np.square(X_train - model.predict(X_train)), axis=1), 95)
y_pred = (mse_test > threshold).astype(int)

f1_macro = f1_score(y_test, y_pred, average='macro')
recall_macro = recall_score(y_test, y_pred, average='macro')

print("=== Résultats Sparse Meta Autoencoder (Private Data) ===")
print(f"F1-macro : {f1_macro:.4f}, Recall-macro : {recall_macro:.4f}")
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-9 : entrainement de l'algorithme SparseMetaAE sur données privées*

```
=== Résultats Sparse Meta Autoencoder (Private Data) ===
F1-macro : 0.9135, Recall-macro : 0.9707
```

Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-10 : résultat de l'algorithme SparseMetaAE sur données privées