

## Données publiques

- On fait une simulation d'anomalies sur les 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.-l : Injection anomalies simulées

On definit l'algorithme VariationalMetaAE

```

# [2] Variational Autoencoder
# =====
def variational_autoencoder():
    inputs = layers.Input(shape=(input_dim,))
    z = layers.Dense(encoding_dim, activation='relu')(inputs)
    z_mean = layers.Dense(encoding_dim//2)(z)
    z_log_var = layers.Dense(encoding_dim//2)(z)

    def sampling(args):
        mean, log_var = args
        epsilon = tf.random.normal(shape=tf.shape(mean))
        return mean + tf.exp(0.5*log_var)*epsilon

    z_samp = layers.Lambda(sampling)([z_mean, z_log_var])
    outputs = layers.Dense(input_dim, activation='linear')(z_samp)
    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 VariationalMetaAE sur données publique*

Ici on entraîne et teste le modèle

```

# -----#
# 3 Entraînement + Test
# =====
model = variational_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 Variational 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 : entraînement de l'algorithme VariationalMetaAE sur données publique*

Voici le résultat

```

1271/1271 [=====] - 2s 2ms/step
== Résultats Variational Autoencoder ==
F1-macro : 0.9195, Recall-macro : 0.9596

```

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

Données privées

- Simulation infections d'anomalies

```

# 4 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.-5 : Injection anomalies simulées*

- On entraîne l'algorithme VariationalMetaAE sur données privées

```

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

def variational_meta_ae():
    inputs = layers.Input(shape=(input_dim,))
    z = layers.Dense(encoding_dim, activation='relu')(inputs)
    z_mean = layers.Dense(encoding_dim//2)(z)
    z_log_var = layers.Dense(encoding_dim//2)(z)

    def sampling(args):
        mean, log_var = args
        epsilon = tf.random.normal(shape=tf.shape(mean))
        return mean + tf.exp(0.5*log_var) * epsilon

    z_samp = layers.Lambda(sampling)([z_mean, z_log_var])
    outputs = layers.Dense(input_dim, activation='linear')(z_samp)
    return models.Model(inputs, outputs)

```

*Figure 5-Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.-6 : entraînement de l'algorithme VariationalMetaAE sur données privées*

```

# -----#
# 6 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.-7 : entraînement de l'algorithme VariationalMetaAE sur données privées*

```

# -----#
# 7 Entraînement + Test
# -----
model = variational_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 Variational 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.-8 : entraînement de l'algorithme VariationalMetaAE sur données privées*

```

== Résultats Variational Meta Autoencoder (Private Data) ==
F1-macro : 0.9368, Recall-macro : 0.9793

```

*Figure5- Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.-9 : résultat de l'algorithme VariationalMetaAE sur données privées*