Données publiques

- Définir le model Prototypical Networks

```
# ============================
# 2️⃣ Encoder ProtoNet
# ============================
def create_encoder():
    return models.Sequential([
        layers.Input(shape=(input_dim,)),
        layers.Dense(64, activation='relu'),
        layers.Dense(32, activation='relu')
    ])

#
```

*Figure 5-**Erreur ! Il n'y a pas de texte répondant à ce style dans ce document.**-1 : entrainement de l'algorithme **ProtoNet** sur données publiques*

```
# ========================
def proto_train(X_train, y_train, epochs=5, k_support=10, k_query=20):
    encoder = create_encoder()
    opt = optimizers.Adam(1e-3)
    for epoch in range(epochs):
        idx_normal = np.where(y_train==0)[0]
        idx_anom = np.where(y_train==1)[0]
        supp_norm = np.random.choice(idx_normal, k_support//2, replace=False)
        supp_anom = np.random.choice(idx_anom, k_support//2, replace=False)
        query_norm = np.random.choice(np.setdiff1d(idx_normal, supp_norm), k_query//2, replace=False)
        query_anom = np.random.choice(np.setdiff1d(idx_anom, supp_anom), k_query//2, replace=False)

        X_supp = np.vstack([X_train[supp_norm], X_train[supp_anom]])
        y_supp = np.array([0]*(k_support//2) + [1]*(k_support//2))
        X_query = np.vstack([X_train[query_norm], X_train[query_anom]])
        y_query = np.array([0]*(k_query//2) + [1]*(k_query//2))

        emb_supp = encoder(X_supp)
        emb_query = encoder(X_query)
        proto_norm = tf.reduce_mean(emb_supp[y_supp==0], axis=0)
        proto_anom = tf.reduce_mean(emb_supp[y_supp==1], axis=0)
        protos = tf.stack([proto_norm, proto_anom])

        dists = tf.reduce_sum((tf.expand_dims(emb_query,1) - protos)**2, axis=2)
        logits = -dists
        loss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y_query, logits=logits))

        grads = tf.gradients(loss, encoder.trainable_variables)
        opt.apply_gradients(zip(grads, encoder.trainable_variables))

        print(f"[ProtoNet Epoch {epoch+1}] Loss = {loss.numpy():.4f}")
    return encoder
```

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

- Evaluation et exécution du model

```python
# ==========================
# 4️⃣ Évaluation
# ==========================
def eval_protonet(encoder, X_train, y_train, X_test, y_test):
    emb_train = encoder(X_train).numpy()
    proto_norm = emb_train[y_train==0].mean(axis=0)
    proto_anom = emb_train[y_train==1].mean(axis=0)
    protos = np.stack([proto_norm, proto_anom])

    emb_test = encoder(X_test).numpy()
    dists = np.sum((emb_test[:,None,:] - protos[None,:,:])**2, axis=2)
    y_pred = np.argmin(dists, axis=1)

    return f1_score(y_test, y_pred, average='macro'), recall_score(y_test, y_pred, average='macro')

# ==========================
# 5️⃣ Exécution
# ==========================
print("=== Entraînement ProtoNet ===")
proto_model = proto_train(X_train, y_train)
print("=== Évaluation finale ProtoNet ===")
f1, rec = eval_protonet(proto_model, X_train, y_train, X_test, y_test)
print(f"ProtoNet -> F1: {f1:.4f}, Recall: {rec:.4f}")
```

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

- Voici le résultat

```
ProtoNet  -> F1: 0.9747, Recall: 0.9747
```

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

Données privées

- Même procédure pour les données privées

```python
# Autoencoder (embedding backbone)
def build_autoencoder(input_dim, encoding_dim):
    inputs = layers.Input(shape=(input_dim,))
    e = layers.Dense(encoding_dim*2, activation='relu')(inputs)
    e = layers.Dense(encoding_dim, activation='relu')(e)
    bottleneck = layers.Dense(max(1,encoding_dim//2), activation='relu')(e)
    d = layers.Dense(encoding_dim, activation='relu')(bottleneck)
    d = layers.Dense(encoding_dim*2, activation='relu')(d)
    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.**-5 : entrainement de l'algorithme **ProtoNet** sur données privées*

```python
# ProtoNet class
class ProtoNet:
    def __init__(self, input_dim, encoding_dim, lr=1e-3):
        self.model = build_autoencoder(input_dim, encoding_dim)
        self.opt = tf.keras.optimizers.Adam(lr)

    def fit(self, X_train, support, epochs=5):
        X_train_tf = tf.convert_to_tensor(X_train, dtype=tf.float32)
        support_tf = tf.convert_to_tensor(support, dtype=tf.float32)
        for epoch in range(epochs):
            with tf.GradientTape() as tape:
                # embeddings
                emb_supp = self.model(support_tf, training=True)
                emb_train = self.model(X_train_tf, training=True)
                proto = tf.reduce_mean(emb_supp, axis=0, keepdims=True)
                loss = tf.reduce_mean(tf.square(emb_train - proto))
            grads = tape.gradient(loss, self.model.trainable_variables)
            self.opt.apply_gradients(zip(grads, self.model.trainable_variables))
            print(f"[ProtoNet] Epoch {epoch+1}/{epochs}, Loss={loss.numpy():.4f}")

    def predict(self, X):
        return self.model.predict(X)
```

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

```python
# Support set
support_idx = np.random.choice(X_test_with_anom.shape[0], size=20, replace=False)
X_support = X_test_with_anom[support_idx]

# --------------------------
# Entraînement & évaluation
protonet = ProtoNet(input_dim, encoding_dim)
protonet.fit(X_train, support=X_support, epochs=8)

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

print("=== Résultats ProtoNet ===")
print(f"F1: {f1_score(y_test, y_pred):.4f}, Recall: {recall_score(y_test, y_pred):.4f}")
```

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

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