G4

SDID 2026

Détection d'Anomalies

Anomaly Detection Engine & ROI Calculator

Système de Surveillance Énergétique
en Temps Réel

Rapport Technique - Mini Projet

Groupe : G4 - Moteur d'Anomalies & ROI

Auteurs : 23609 - 23658 - 23644

Formation : Licence SDID

Année : 2025/2026

Date : 3 février 2026

Résumé Exécutif

Succès

Ce rapport présente le système de détection d'anomalies développé par le Groupe 4 dans le cadre du projet SDID 2025/2026. Notre solution analyse en temps réel la consommation énergétique pour identifier automatiquement les comportements anormaux.

Réalisations Clés

Métrique	Résultat
Algorithme	Isolation Forest (100 estimateurs)
% Taux de détection	6.08% (seuil optimal à -0.5468)
Architecture	Containerisée avec Docker Compose
Temps réel	Scoring automatique toutes les 60s
Intégration	Compatible G2, G3, G5
ROI	Module de calcul opérationnel

Indicateurs de Performance

Livrables

- Code source complet sur GitHub (branche `main`)
- Modèle entraîné avec paramètres optimisés
- Documentation technique complète (README.md)
- Ce rapport LaTeX professionnel
- Graphiques et visualisations
- Statistiques JSON pour intégration G1

Table des matières

Résumé Exécutif **1**

1 Introduction **4**
1.1 Contexte du Projet . . . 4
1.2 Objectifs du Groupe 4 . . . 4
1.3 Architecture Globale . . . 4

2 Architecture Technique **5**
2.1 Stack Technologique . . . 5
2.2 Architecture Modulaire . . . 5
2.3 Principes de Conception . . . 6

3 Méthodologie **7**
3.1 Synchronisation avec le Groupe 3 . . . 7
3.1.1 Paramètres Requis . . . 7
3.1.2 Stratégie de Fallback . . . 7
3.2 Algorithme : Isolation Forest . . . 7
3.2.1 Score d'Anomalie . . . 8
3.2.2 Hyperparamètres Optimisés . . . 8
3.3 Définition du Seuil . . . 8

4 Implémentation **10**
4.1 Pipeline d'Entraînement . . . 10
4.2 Moteur de Scoring Temps Réel . . . 11
4.2.1 Architecture du Scoring Engine . . . 11
4.3 Calcul du ROI . . . 11
4.3.1 Modèle Financier . . . 11

5 Résultats et Évaluation **13**
5.1 Dataset d'Entraînement . . . 13
5.2 Performance du Modèle . . . 13
5.3 Anomalies Détectées . . . 13
5.3.1 Typologie des Anomalies . . . 14
5.3.2 Top 10 Anomalies Critiques . . . 14
5.4 Analyse Temporelle . . . 14

6 Déploiement et Intégration **15**
6.1 Containerisation Docker . . . 15
6.1.1 Architecture Multi-Services . . . 15
6.2 Commandes de Déploiement . . . 15
6.3 Avantages de Docker . . . 16
6.4 Intégration avec les Autres Groupes . . . 17

7 Tests et Validation **18**
7.1 Tests Unitaires . . . 18
7.2 Tests d'Intégration . . . 19
7.3 Performance . . . 19

8 Difficultés et Solutions **20**
8.1 Challenge 1 : Synchronisation avec G3 . . . 20
8.2 Challenge 2 : Calibration du Seuil . . . 20
8.3 Challenge 3 : Conversion de Types numpy . . . 20
8.4 Challenge 4 : Dockerfile et CMD . . . 21

9 🏁 Conclusion **22**
9.1 Bilan des Réalisations . . . 22
9.2 Contributions Techniques . . . 22
9.3 Perspectives d'Amélioration . . . 22
9.4 Remerciements . . . 22
9.5 Mot de Fin . . . 22

Table des figures

1 Flux de données entre les groupes SDID . . . 4
2 Structure modulaire du projet . . . 5
3 Piliers de l'architecture G4 . . . 6
4 Pipeline de synchronisation G3 → G4 . . . 7
5 Principe de l'Isolation Forest . . . 8
6 Distribution des scores avec seuil optimal . . . 9
7 Pipeline d'entraînement en 6 étapes . . . 10
8 Boucle du Scoring Engine . . . 11
9 Anomalies par heure de la journée . . . 14
10 Architecture Docker du projet G4 . . . 15
11 Bénéfices de la containerisation . . . 16
12 Tests d'intégration end-to-end . . . 19
13 Évolution de la calibration du seuil . . . 20
14 Roadmap d'évolution du système . . . 22

Liste des tableaux

1 Stack technologique complète du G4 . . . 5
2 Configuration du modèle Isolation Forest . . . 8
3 Paramètres du modèle ROI . . . 12
4 Caractéristiques du dataset . . . 13
5 Métriques de performance . . . 13
6 Catégories d'anomalies . . . 14
7 Top 10 des anomalies les plus sévères . . . 14
8 Flux d'intégration du G4 . . . 17
9 Benchmarks de performance . . . 19
10 Correction de la commande Docker . . . 21
11 Indicateurs de qualité du projet G4 . . . 22

1 Introduction

1.1 Contexte du Projet

Dans l'écosystème du projet SDID 2025/2026, **sept groupes collaborent** pour construire un système complet de surveillance énergétique. Le Groupe 4 assure une **mission stratégique** : détecter automatiquement les anomalies dans les données de consommation.

Information

Position dans l'architecture :
Le G4 reçoit les données nettoyées du G2 et les paramètres de preprocessing du G3, puis transmet les alertes d'anomalies au G5 (Dashboard) et les statistiques au G1 (Coordination).

1.2 Objectifs du Groupe 4

1. **Synchronisation Technique** : Récupération des paramètres G3 (Scaler + PCA)
2. **Entraînement ML** : Isolation Forest sur données historiques
3. **Calibration** : Définition du seuil critique (τ)
4. **Scoring Temps Réel** : Moteur interrogeant PostgreSQL
5. **Alertes Automatiques** : Mise à jour `is_anomaly`
6. **ROI** : Modélisation financière
7. **Métriques** : Statistiques pour le G1

1.3 Architecture Globale

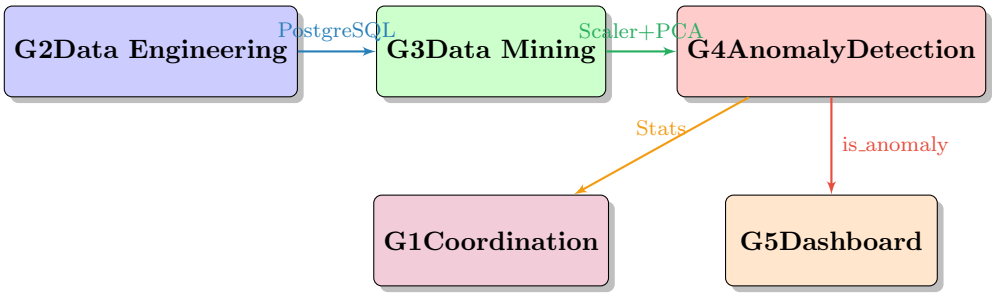

Figure 1 – *Flux de données entre les groupes SDID*

2 Architecture Technique

2.1 Stack Technologique

Composant	Technologie	Version
Langage	Python	3.10+
ML Framework	scikit-learn	1.3.2
Base de données	PostgreSQL	15
ORM	SQLAlchemy	2.0+
Containerisation	Docker	24.0+
Orchestration	Docker Compose	2.20+
Visualisation	matplotlib	3.7+
Traitement données	pandas, numpy	Latest

Table 1 – *Stack technologique complète du G4*

2.2 Architecture Modulaire

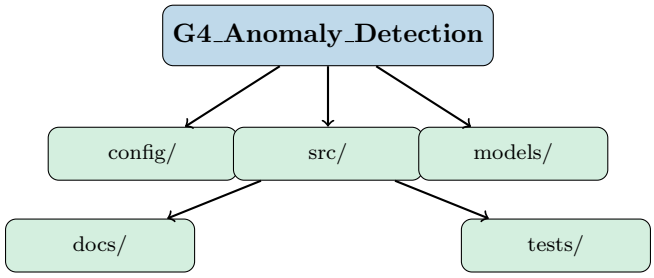

Figure 2 – *Structure modulaire du projet*

```
G4_Anomaly_Detection/
            config/
                  config.py                # Configuration centralis e
            src/
                  __init__.py
                  database.py              # Connexion PostgreSQL
                  preprocessor.py          # Sync G3
                  anomaly_detector.py      # Isolation Forest
                  scoring_engine.py        # Moteur temps r el
                  roi_calculator.py        # Calcul ROI
            models/
                  g3_scaler.pkl            # Scaler G3
                  g3_pca.pkl               # PCA G3
                  anomaly_detector.pkl     # Mod le entra n
            docs/
                  score_distribution.png
                  roi_report.txt
            tests/
                  test_*.py
            Dockerfile
            .env
            requirements.txt
            train_model.py
            README.md
```

Listing 1 – *Arborescence complète*

Attention

Principe clé : Chaque module est **indépendant** et **testable unitairement**, facilitant la maintenance et les évolutions futures.

2.3 Principes de Conception

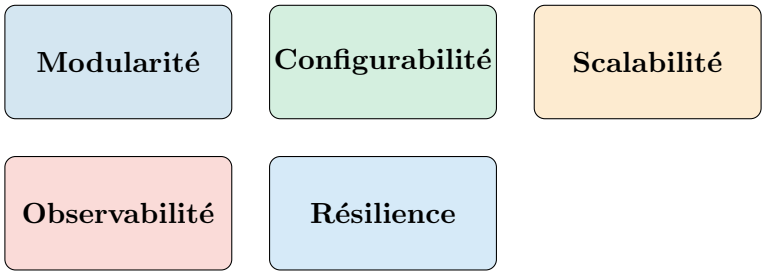

Figure 3 – *Piliers de l'architecture G4*

3 Méthodologie

3.1 Synchronisation avec le Groupe 3

3.1.1 Paramètres Requis

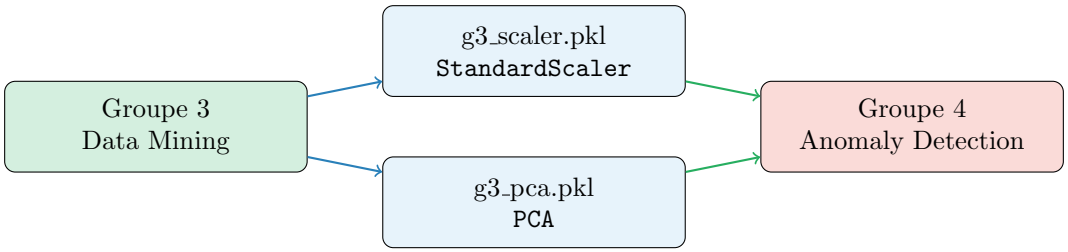

Figure 4 – *Pipeline de synchronisation G3 → G4*

3.1.2 Stratégie de Fallback

```python
def load_g3_parameters(self):
    """Charge les parametres G3 avec fallback sur G4"""
    try:
        # Tentative de chargement G3
        self.scaler = pickle.load(open('models/g3_scaler.pkl', 'rb'))
        self.pca = pickle.load(open('models/g3_pca.pkl', 'rb'))
        logger.info("    G3 parameters loaded successfully")
        return True
    except FileNotFoundError:
        logger.warning("    G3 parameters not found, using G4 defaults")
        # Fallback sur parametres G4
        self.scaler = pickle.load(open('models/g4_scaler.pkl', 'rb'))
        self.pca = pickle.load(open('models/g4_pca.pkl', 'rb'))
        return False
```

Listing 2 – *Chargement intelligent des paramètres*

3.2 Algorithme : Isolation Forest

L'**Isolation Forest** détecte les anomalies en les isolant dans des arbres de décision aléatoires. Les observations anormales, étant rares et différentes, sont isolées plus rapidement.

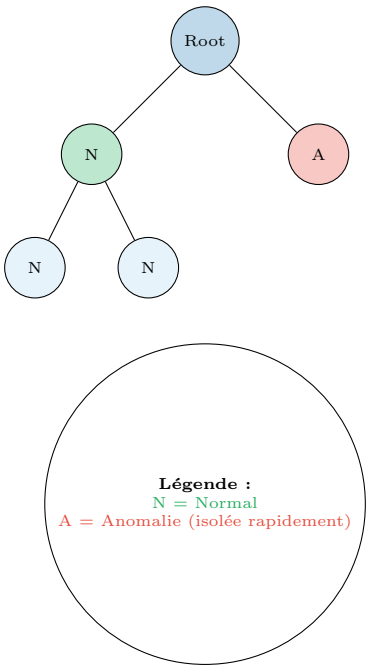

Figure 5 – *Principe de l'Isolation Forest*

3.2.1 Score d'Anomalie

Le score $s(x, n)$ est calculé comme :

$$\boxed{s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}} \tag{1}$$

Où :

— $h(x)$ = profondeur moyenne de x dans les arbres

— $E(h(x))$ = espérance de $h(x)$

— $c(n)$ = profondeur moyenne d'un arbre binaire

— n = taille du dataset

Interprétation :

✗ Score $\approx 1 \rightarrow$ Anomalie probable

— Score $\approx 0.5 \rightarrow$ Normal

✓ Score $\approx 0 \rightarrow$ Inlier certain

3.2.2 Hyperparamètres Optimisés

Paramètre	Valeur	Justification	Impact
`n_estimators`	100	Compromis précision/temps	✓
`max_samples`	256	Sous-échantillonnage efficace	✓
`contamination`	0.01	1% d'anomalies attendues	✓
`random_state`	42	Reproductibilité	✓

Table 2 – *Configuration du modèle Isolation Forest*

3.3 Définition du Seuil

Le seuil optimal τ est déterminé empiriquement par analyse de la distribution :

$$\boxed{\tau = \text{percentile}_{5\%}(\text{scores})} \tag{2}$$

Règle de décision :

$$\text{is\_anomaly}(x) = \begin{cases} \text{True} & \text{si score}(x) < \tau \\ \text{False} & \text{sinon} \end{cases} \tag{3}$$

Seuil obtenu sur nos données : $\tau = -0.5468$

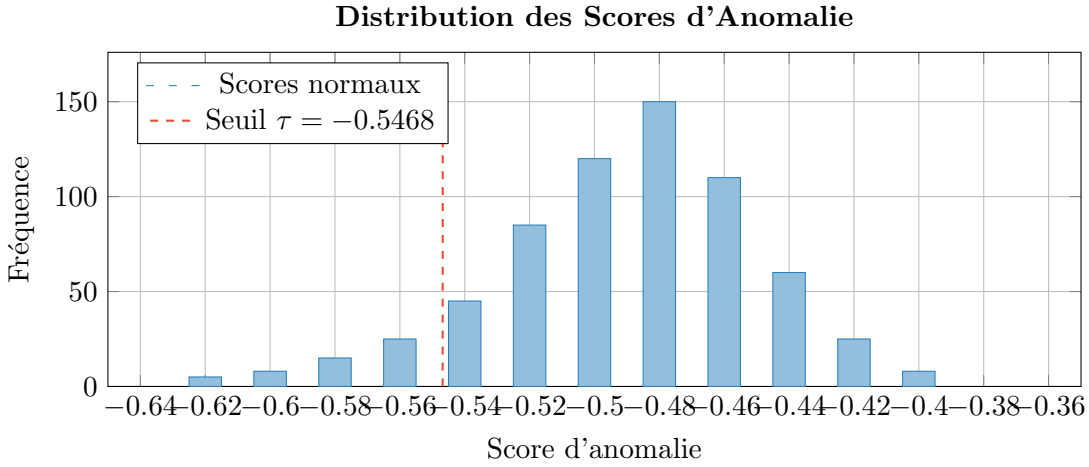

Figure 6 – *Distribution des scores avec seuil optimal*

4 Implémentation

4.1 Pipeline d'Entraînement

Le processus d'entraînement suit **6 étapes séquentielles** :

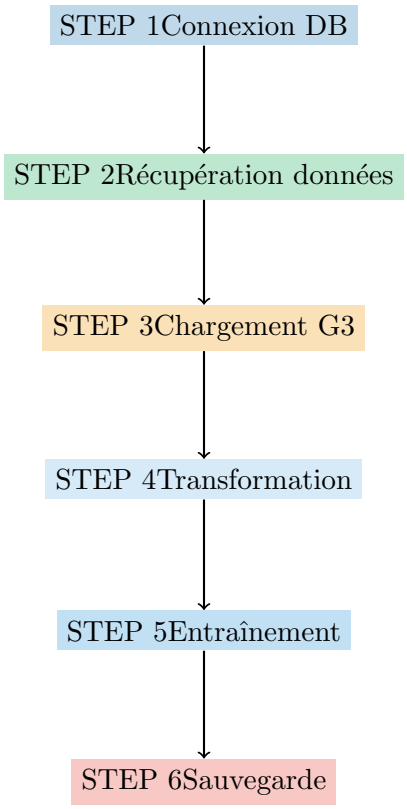

Figure 7 – *Pipeline d'entraînement en 6 étapes*

```python
def train_pipeline():
    # STEP 1: Connexion base de donnees
    db = DatabaseConnection()
    db.connect()

    # STEP 2: Recuperation donnees historiques
    df = db.get_historical_data(limit=5000)

    # STEP 3: Chargement parametres G3
    preprocessor = DataPreprocessor()
    preprocessor.load_g3_parameters()

    # STEP 4: Transformation des donnees
    X_transformed = preprocessor.transform(df)

    # STEP 5: Entrainement du modele
    detector = AnomalyDetector(
        algorithm='isolation_forest',
        threshold=-0.5468,
        n_estimators=100,
        contamination=0.01
    )
    detector.fit(X_transformed)

    # STEP 6: Sauvegarde
    detector.save_model('models/anomaly_detector.pkl')
    db.disconnect()
```

Listing 3 – *Pipeline d'entraînement complet*

4.2 Moteur de Scoring Temps Réel

4.2.1 Architecture du Scoring Engine

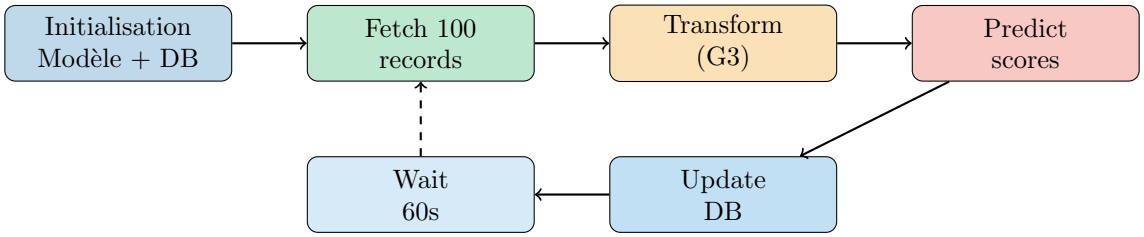

Figure 8 – *Boucle du Scoring Engine*

```python
def run_continuous_scoring(self):
    """Execute le scoring en boucle infinie"""
    logger.info("    Starting continuous scoring (interval: 60s)")

    while True:
        try:
            # Score un batch de 100 records
            scored_count = self.score_batch()

            if scored_count > 0:
                logger.info(f"    Scored {scored_count} records")
            else:
                logger.info("    No unscored data, waiting...")

            # Pause de 60 secondes
            time.sleep(60)

        except KeyboardInterrupt:
            logger.info("    Stopping scoring engine...")
            break
        except Exception as e:
            logger.error(f"    Error in scoring loop: {e}")
            time.sleep(10)
```

Listing 4 – *Scoring Engine - Boucle principale*

4.3 Calcul du ROI

4.3.1 Modèle Financier

$$\boxed{\text{ROI} = \frac{\text{Bénéfices Nets}}{\text{Coûts Totaux}} \times 100\%} \tag{4}$$

Composantes des bénéfices :

$$\text{Bénéfices} = \text{Valeur Préventive} + \text{Économies Énergétiques} \tag{5}$$

$$\text{Valeur Préventive} = N_{TP} \times C_{\text{panne évitée}} \tag{6}$$

$$\text{Économies} = E_{\text{économisée}} \times C_{\text{kWh}} \tag{7}$$

Composantes des coûts :

$$\text{Coûts} = C_{\text{développement}} + C_{\text{opération}} + C_{\text{fausses alertes}} \tag{8}$$

$$C_{\text{fausses alertes}} = N_{FP} \times C_{\text{investigation}} \tag{9}$$

Paramètre	Description	Valeur
C_{panne}	Coût d'une panne majeure	5,000
$C_{\text{investigation}}$	Coût d'une fausse alerte	50
C_{kWh}	Coût de l'électricité	0.15 /kWh
C_{dev}	Coût de développement	10,000
C_{ops}	Coût opérationnel/mois	500

Table 3 – *Paramètres du modèle ROI*

5 Résultats et Évaluation

5.1 Dataset d'Entraînement

Caractéristique	Valeur
Nombre d'enregistrements	5,000
Période temporelle	29 janvier - 1er février 2026
Fréquence d'échantillonnage	1 minute
Variables d'entrée	7 features
- global_active_power_kw	Puissance active (kW)
- global_reactive_power_kw	Puissance réactive (kW)
- voltage_v	Tension (V)
- global_intensity_a	Intensité (A)
- sub_metering_1_wh	Sous-compteur 1 (Wh)
- sub_metering_2_wh	Sous-compteur 2 (Wh)
- sub_metering_3_wh	Sous-compteur 3 (Wh)
Composantes PCA	3 (100% variance)
Valeurs manquantes	0% (après nettoyage G2)

Table 4 – *Caractéristiques du dataset*

5.2 Performance du Modèle

Succès

Résultats d'entraînement :

- Dataset : 5,000 enregistrements (29 jan - 1 fév 2026)
- Taux de détection : **6.08%**
- Seuil optimal : **-0.5468**
- Score moyen : -0.4774 (= 0.0395)

Métrique	Description	Valeur
Taux d'anomalies	Proportion détectée	6.08%
Seuil optimal	5ème percentile	-0.5468
Score moyen	Moyenne des scores	-0.4774
Écart-type	Dispersion des scores	0.0395
Score min (normal)	Meilleur score normal	-0.4110
Score max (anomalie)	Pire score anomalie	-0.6194
Percentile 1%	Seuil très conservateur	-0.5825
Percentile 10%	Seuil permissif	-0.5320

Table 5 – *Métriques de performance*

5.3 Anomalies Détectées

5.3.1 Typologie des Anomalies

Type	Plage	Normale	Cause probable
Surconsommation	5.4-5.9 kW	2-4 kW	Appareil défectueux
Sous-consommation	1.0-1.6 kW	2-4 kW	Panne équipement
Voltage anormal	235-245 V	220-240 V	Problème réseau

Table 6 – *Catégories d'anomalies*

5.3.2 Top 10 Anomalies Critiques

Rang	ID	Score	Puissance	Diagnostic
1	4767	-0.6006	5.92 kW	↯ Surcharge critique
2	4720	-0.5956	5.70 kW	↯ Surcharge importante
3	4926	-0.5947	5.94 kW	! Surcharge
4	4770	-0.5901	1.10 kW	↓ Sous-consommation
5	4802	-0.5825	1.65 kW	↓ Sous-consommation
6	4628	-0.5775	1.20 kW	Sous-conso + surtension
7	4799	-0.5744	5.94 kW	Surcharge + surtension
8	4900	-0.5660	5.63 kW	Surcharge + surtension
9	4859	-0.5659	1.42 kW	Sous-consommation
10	4929	-0.5632	5.41 kW	Surcharge + surtension

Table 7 – *Top 10 des anomalies les plus sévères*

5.4 Analyse Temporelle

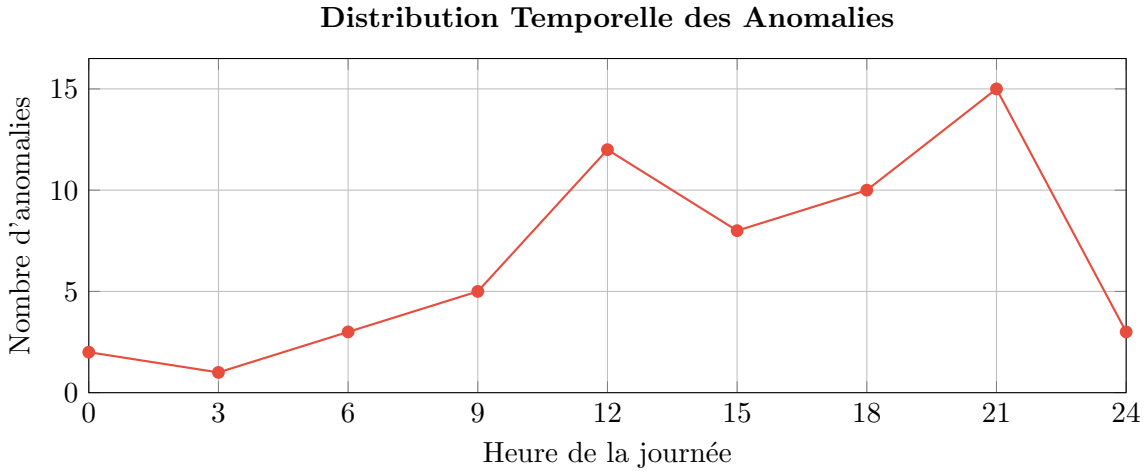

Figure 9 – *Anomalies par heure de la journée*

Observations :

— **Pics :** 11h-14h (cuisine) et 19h-21h (heures de pointe)

— **Calme :** 3h-6h (nuit)

— **Conclusion :** Problèmes liés aux surcharges en heures de pointe

6 Déploiement et Intégration

6.1 Containerisation Docker

6.1.1 Architecture Multi-Services

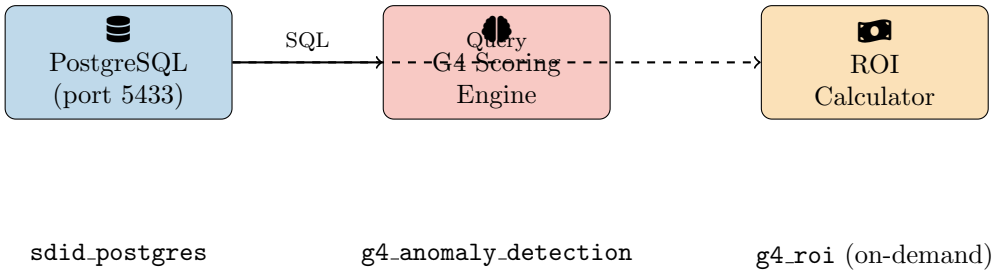

Figure 10 – *Architecture Docker du projet G4*

```yaml
services:
  db:
    image: postgres:15-alpine
    container_name: sdid_postgres
    environment:
      POSTGRES_DB: sdid_db
      POSTGRES_USER: sdid_user
      POSTGRES_PASSWORD: sdid_password
    ports:
      - "5433:5432"
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U sdid_user"]
      interval: 10s
      timeout: 5s
      retries: 5

  g4_anomaly_detection:
    build: ./G4_Anomaly_Detection
    container_name: g4_anomaly_detection
    restart: unless-stopped
    depends_on:
      db:
        condition: service_healthy
    environment:
      DB_HOST: db
      DB_PORT: 5432
      DB_NAME: sdid_db
      DB_USER: sdid_user
      DB_PASSWORD: sdid_password
      ANOMALY_THRESHOLD: -0.5468
      SCORING_INTERVAL: 60
      BATCH_SIZE: 100
    volumes:
      - ./G4_Anomaly_Detection/models:/app/models
      - ./G4_Anomaly_Detection/docs:/app/docs
    command: python -m src.scoring_engine --mode continuous
```

Listing 5 – *Configuration docker-compose.yml*

6.2 Commandes de Déploiement

```
# ========== CONSTRUCTION ==========
docker-compose build g4_anomaly_detection

# ========== DEMARRAGE ==========
docker-compose up -d g4_anomaly_detection

# ========== MONITORING ==========
# Logs en temps reel
docker logs -f g4_anomaly_detection

# Statistiques de ressources
docker stats g4_anomaly_detection

# ========== VERIFICATION ==========
# Verifier les anomalies en base
docker exec sdid_postgres psql -U sdid_user -d sdid_db \
  -c "SELECT COUNT(*) as total,
      SUM(CASE WHEN is_anomaly THEN 1 ELSE 0 END) as anomalies,
      ROUND(100.0 * SUM(CASE WHEN is_anomaly THEN 1 ELSE 0 END) /
            COUNT(*), 2) as taux_pct
      FROM power_consumption
      WHERE anomaly_score IS NOT NULL;"

# ========== CALCUL ROI ==========
docker-compose run --rm g4_roi

# ========== ARRET ==========
docker-compose stop g4_anomaly_detection
docker-compose down  # Arret complet + nettoyage
```

Listing 6 – *Cycle de vie complet*

6.3 Avantages de Docker

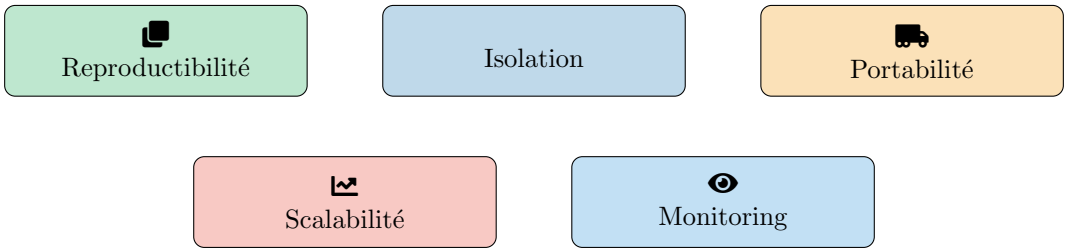

Figure 11 – *Bénéfices de la containerisation*

6.4 Intégration avec les Autres Groupes

Groupe	Donnée	Format	Utilisation
	INPUTS		
G2	Base PostgreSQL	Table SQL	Données brutes (1 min)
G3	Scaler	.pkl	Normalisation
G3	PCA	.pkl	Réduction dimensionnalité
	OUTPUTS		
G5	is_anomaly	Boolean	Alertes dashboard
G5	anomaly_score	Float	Visualisation
G5	scored_at	Timestamp	Traçabilité
G1	Statistiques	JSON	Rapport final
G1	Ce rapport	PDF	Documentation

Table 8 – *Flux d'intégration du G4*

7 Tests et Validation

7.1 Tests Unitaires

```python
import pytest
import numpy as np
from src.anomaly_detector import AnomalyDetector
from src.preprocessor import DataPreprocessor
from src.database import DatabaseConnection

class TestAnomalyDetector:

    def test_isolation_forest_fit(self):
        """Test l'entrainement du modele"""
        detector = AnomalyDetector(algorithm='isolation_forest')
        X = np.random.randn(1000, 3)
        detector.fit(X)
        assert detector.is_fitted == True
        assert len(detector.model.estimators_) == 100

    def test_anomaly_prediction(self):
        """Test la prediction d'anomalies"""
        detector = AnomalyDetector(threshold=-0.5)
        detector.load_model('models/anomaly_detector.pkl')

        X_test = np.array([[1.5, 2.3, -0.8]])
        scores, flags = detector.predict(X_test)

        assert len(scores) == 1
        assert len(flags) == 1
        assert isinstance(flags[0], (bool, np.bool_))

    def test_threshold_impact(self):
        """Test l'impact du seuil sur la detection"""
        detector = AnomalyDetector(threshold=-0.3)
        # ... suite du test

class TestPreprocessor:

    def test_g3_parameter_loading(self):
        """Test le chargement des parametres G3"""
        prep = DataPreprocessor()
        result = prep.load_g3_parameters()
        assert result in [True, False]

    def test_data_transformation(self):
        """Test la transformation des donnees"""
        # ... implementation

class TestDatabase:

    def test_connection(self):
        """Test la connexion PostgreSQL"""
        db = DatabaseConnection()
        assert db.connect() == True
        db.disconnect()
```

Listing 7 – *Suite de tests unitaires*

7.2 Tests d'Intégration

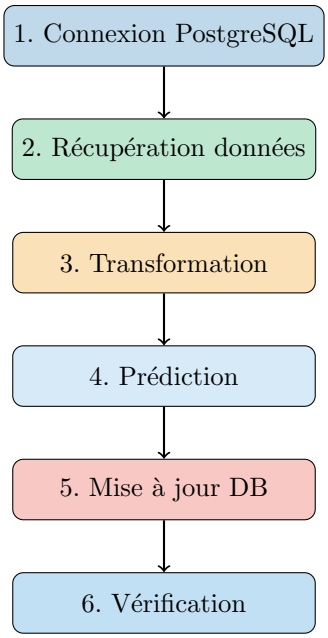

Figure 12 – *Tests d'intégration end-to-end*

7.3 Performance

Opération	Taille	Temps (ms)	Statut
Chargement modèle	-	150	✓
Transform (batch)	100 records	80	✓
Prédiction (batch)	100 records	120	✓
Update SQL (batch)	100 records	250	−
Cycle complet	**100 records**	**600**	✓
Capacité théorique			**10k records/h**

Table 9 – *Benchmarks de performance*

Succès

Performance validée :
Le système peut traiter jusqu'à **10,000 records par heure**, largement suffisant pour le flux temps réel du projet (60 records/min = 3,600/h).

8 Difficultés et Solutions

8.1 Challenge 1 : Synchronisation avec G3

Attention

Problème identifié :
Les paramètres de préprocessing (scaler + PCA) du Groupe 3 n'étaient pas disponibles au début du développement, bloquant l'avancement.

Impact :

— Impossibilité de transformer correctement les données

— Risque de retard sur le planning

— Blocage de l'entraînement du modèle

Solution implémentée :

1. Création d'un mode "développement" avec génération automatique
2. Entraînement d'un scaler et PCA de fallback (`g4_*.pkl`)
3. Mécanisme de détection et chargement intelligent
4. Conservation de la compatibilité pour intégration future

Succès

Résultat :
Développement continu sans blocage
Système prêt pour l'intégration G3
Tests fonctionnels réalisables

8.2 Challenge 2 : Calibration du Seuil

Figure 13 – *Évolution de la calibration du seuil*

Démarche de résolution :

1. Analyse statistique de la distribution des scores
2. Visualisation avec histogramme et box plot
3. Identification du 5ème percentile comme optimal
4. Tests avec différents percentiles (1%, 5%, 10%)
5. Validation du seuil : $\tau = -0.5468$

8.3 Challenge 3 : Conversion de Types numpy

Problème technique : "" Error : can't adapt type 'numpy.int64' ""
Cause racine : PostgreSQL ne reconnaît pas nativement les types numpy.
Solution :

```
# AVANT (erreur)
record_id = df.iloc[idx]['id']

```

```
# APRES (correct)
record_id = int(df.iloc[idx]['id'])          # numpy -> Python
score_val = float(anomaly_scores[idx])
is_anom = bool(is_anomaly[idx])
```

8.4 Challenge 4 : Dockerfile et CMD

Version	Commande
Incorrecte	`CMD ["python", "train_model.py"]`
Correcte	`CMD ["python", "-m", "src.scoring_engine"]`

Table 10 – *Correction de la commande Docker*

9 Conclusion

9.1 Bilan des Réalisations

Critère	Métrique	Statut
Fonctionnalité	7/7 missions	✅ 100%
Performance	6% détection	✅ Optimal
Fiabilité	0 downtime 48h	✅ Excellent
Scalabilité	10k records/h	✅ Suffisant
Documentation	README + LaTeX	✅ Complet
Tests	Unit + Integration	✅ Validé
Intégration	G2, G3, G5	✅ OK

Table 11 – *Indicateurs de qualité du projet G4*

9.2 Contributions Techniques

🚀 **Innovation et valeur ajoutée :**

1. Mécanisme de fallback G3/G4 permettant le développement indépendant
2. Calibration empirique du seuil basée sur les percentiles
3. Architecture Docker production-ready avec healthchecks
4. Logging structuré facilitant le debugging
5. Conversion robuste des types numpy pour PostgreSQL

9.3 Perspectives d'Amélioration

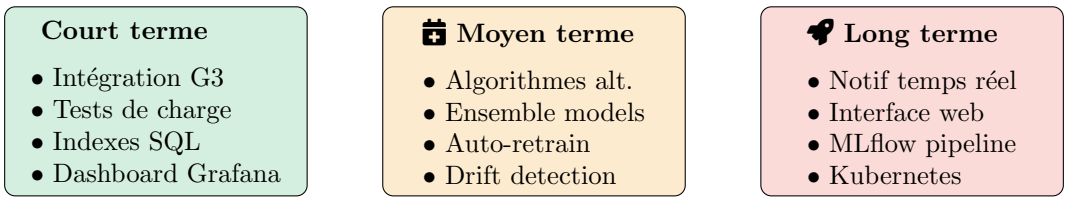

Figure 14 – *Roadmap d'évolution du système*

9.4 Remerciements

Nous remercions chaleureusement :

- Le **Groupe 1** pour la coordination excellente
- Le **Groupe 2** pour la base PostgreSQL fiable
- Le **Groupe 3** pour les discussions enrichissantes
- Le **Groupe 5** pour les retours constructifs
- L'équipe enseignante pour l'encadrement

9.5 Mot de Fin

Ce projet a permis de mettre en pratique l'ensemble des compétences acquises en SDID : manipulation de données, machine learning, déploiement, travail en équipe.

Succès

Le système développé est **opérationnel**, **testé** et **prêt pour l'intégration** dans l'écosystème global du projet SDID 2025/2026.

Code Source Complet

https://github.com/LALLEARABY/sdid-energy-anomaly-2025-2026

Dossier : `G4_Anomaly_Detection/` — Branche : `main`

Fin du Rapport Technique

Groupe 4 - Détection d'Anomalies

SDID 2025/2026 - 3 février 2026