Architecture Technique - Détection de Fraude Financière par Graphes

📋 Informations du Projet

Item	Valeur
Numéro de groupe	42
Membres	Malak El Idrissi, Joe Boueri
Cours	IA Exploratoire et Symbolique - ECE
Sujet	Détection de fraude financière par graphes
Version	1.0
Date	Janvier 2026

1. Choix Technologique Final

1.1 Stack Technologique Principale

Composant	Technologie	Justification
Langage	Python 3.10+	Langage standard pour le data science, excellent support pour les bibliothèques scientifiques
Bibliothèque de graphes	NetworkX	Bibliothèque Python pure, simple à installer, excellente documentation, adaptée aux datasets académiques (10k-100k transactions)
Manipulation de données	Pandas	Standard de facto pour le traitement de données tabulaires en Python
Visualisation	Matplotlib	Visualisation statique de qualité, compatible avec Jupyter notebooks

2. Structure des Dossiers

```
42. Détection de fraude financière par graphes/
│
├── src/                          # Code source principal
│   ├── __init__.py
│   ├── fraud_detector.py         # Point d'entrée principal
│   │
│   ├── data/                     # Module de chargement des données
│   │   ├── __init__.py
│   │   ├── loader.py             # Chargement depuis fichiers (CSV, JSON)
│   │   ├── generator.py          # Génération de données synthétiques
│   │   └── models.py             # Modèles de données (Transaction, Compte)
│   │
│   ├── graph/                    # Module de construction de graphes
│   │   ├── __init__.py
│   │   ├── builder.py            # Construction du graphe transactionnel
│   │   └── utils.py              # Utilitaires de graphes
│   │
│   ├── detection/                # Module de détection de fraudes
│   │   ├── __init__.py
│   │   ├── base.py               # Classe abstraite de détecteur
│   │   ├── cycle_detector.py     # Détection de cycles de blanchiment
│   │   ├── smurfing_detector.py  # Détection de smurfing
│   │   └── network_detector.py   # Détection d'anomalies de réseaux
│   │
│   ├── metrics/                  # Module d'analyse et de métriques
│   │   ├── __init__.py
│   │   ├── centrality.py         # Métriques de centralité
│   │   ├── community.py          # Détection de communautés
│   │   └── pagerank.py           # Calcul de PageRank
│   │
│   ├── visualization/            # Module de visualisation
│   │   ├── __init__.py
│   │   ├── plotter.py            # Visualisation des graphes
│   │   └── report.py             # Génération de rapports
│   │
│   └── evaluation/               # Module d'évaluation
│       ├── __init__.py
│       ├── evaluator.py          # Évaluation des détecteurs
│       └── metrics.py            # Métriques de performance
│
├── data/                         # Jeux de données
│   ├── synthetic/                # Données générées
│   │   ├── small_dataset.csv     # Dataset de test (~1000 transactions)
│   │   ├── medium_dataset.csv    # Dataset de démonstration (~10000 transactions)
│   │   └── large_dataset.csv     # Dataset de validation (~50000 transactions)
│   │
│   ├── real/                     # Données réelles (si disponibles)
│   │   └── .gitkeep
│   │
│   └── ground_truth/             # Annotations pour l'évaluation
│       ├── small_labels.csv
│       └── medium_labels.csv
│
├── notebooks/                    # Jupyter notebooks pour exploration
│   ├── 01_exploration.ipynb      # Exploration des données
│   ├── 02_graph_construction.ipynb  # Construction de graphes
│   ├── 03_cycle_detection.ipynb  # Détection de cycles
│   ├── 04_smurfing_detection.ipynb # Détection de smurfing
│   ├── 05_network_anomalies.ipynb # Anomalies de réseaux
│   └── 06_visualization.ipynb    # Visualisations avancées
│
├── tests/                        # Tests unitaires
│   ├── __init__.py
│   ├── test_loader.py
│   ├── test_graph_builder.py
│   ├── test_cycle_detector.py
│   ├── test_smurfing_detector.py
│   ├── test_network_detector.py
│   └── test_evaluation.py
│
├── docs/                         # Documentation technique
│   ├── ARCHITECTURE.md           # Ce document
│   ├── API.md                    # Documentation de l'API
│   └── GUIDE_UTILISATION.md      # Guide d'utilisation
│
├── slides/                       # Support de présentation
│   ├── slides.md                 # Structure des slides
│   ├── images/                   # Images et diagrammes pour présentation
│   └── export/                   # PDF/PPT exportés
│
├── plans/                        # Plans et spécifications
│   └── .gitkeep
│
├── requirements.txt              # Dépendances Python
├── setup.py                      # Script d'installation
├── .gitignore                    # Fichiers ignorés par Git
├── README.md                     # Présentation du projet
└── RECHERCHE.md                  # Document de recherche
```

3. Architecture des Composants

3.1 Vue d'Ensemble

```
graph TB
    subgraph Sources
        CSV[Données CSV]
        JSON[Données JSON]
        SYNTH[Données Synthétiques]
    end

    subgraph src_data
        LOADER[loader.py]
        GEN[generator.py]
        MODELS[models.py]
    end

    subgraph src_graph
        BUILDER[builder.py]
        GRAPH[Structure Graphe]
    end

    subgraph src_detection
        CYCLE[cycle_detector.py]
        SMURF[smurfing_detector.py]
        NET[network_detector.py]
    end

    subgraph src_metrics
        CENTR[centrality.py]
        COMM[community.py]
        PR[pagerank.py]
    end

    subgraph src_visualization
        PLOT[plotter.py]
        REPORT[report.py]
    end

    subgraph src_evaluation
        EVAL[evaluator.py]
        METRICS[metrics.py]
    end

    CSV --> LOADER
    JSON --> LOADER
    SYNTH --> GEN
    LOADER --> MODELS
    GEN --> MODELS
    MODELS --> BUILDER
    BUILDER --> GRAPH

    GRAPH --> CYCLE
    GRAPH --> SMURF
    GRAPH --> NET

    GRAPH --> CENTR
    GRAPH --> COMM
    GRAPH --> PR

    CENTR --> NET
    COMM --> NET
    PR --> NET

    CYCLE --> PLOT
    SMURF --> PLOT
    NET --> PLOT

    CYCLE --> REPORT
    SMURF --> REPORT
    NET --> REPORT
```

4. Flux de Données

4.1 Diagramme de Flux

```
sequenceDiagram
    participant User
    participant Loader
    participant Builder
    participant Graph
    participant Detectors
    participant Visualization
    participant Evaluator

    User->>Loader: Charger données (CSV/JSON/Synthétique)
    Loader->>Loader: Validation des données
    Loader-->>Builder: Transactions validées
    Builder->>Builder: Construction du graphe
    Builder-->>Graph: Graphe transactionnel

    par Détection parallèle
        Graph->>Detectors: Graphe
        Detectors->>Detectors: CycleDetector
        Detectors->>Detectors: SmurfingDetector
        Detectors->>Detectors: NetworkAnomalyDetector
    end

    Detectors-->>Visualization: Résultats de détection
    Visualization->>Visualization: Génération des visualisations
    Visualization-->>User: Graphes et rapports

    Note over User,Evaluator: Mode évaluation
    User->>Evaluator: Définition ground truth
    Evaluator->>Detectors: Exécution sur dataset test
    Detectors->>Evaluator: Prédictions
```

5. Interface Utilisateur

5.1 Interface en Ligne de Commande (CLI)

Le système fournit une interface CLI pour exécuter les détections directement depuis le terminal.

```
# Exécution avec dataset par défaut
python src/fraud_detector.py

# Exécution avec fichier spécifique
python src/fraud_detector.py --input data/medium_dataset.csv

# Exécution avec génération de données
python src/fraud_detector.py --generate --n_transactions 10000

# Exécution en mode évaluation
python src/fraud_detector.py --evaluate --ground_truth data/ground_truth/medium_labels.csv

# Configuration des détecteurs
python src/fraud_detector.py --config config.json --threshold 0.7

# Export des résultats
python src/fraud_detector.py --output results/ --format json
```

6. Métriques d'Évaluation

6.1 Métriques de Classification

Métrique	Formule	Description
Précision	TP / (TP + FP)	Proportion de fraudes correctement identifiées parmi celles détectées
Rappel (Recall)	TP / (TP + FN)	Proportion de fraudes réellement détectées
F1-Score	2 × (Precision × Recall) / (Precision + Recall)	Moyenne harmonique de précision et rappel
Spécificité	TN / (TN + FP)	Proportion de transactions normales correctement identifiées
Accuracy	(TP + TN) / (TP + TN + FP	Taux de classifications correctes

7. Patterns de Conception

7.1 Pattern Strategy (Détection)

Chaque type de fraude est détecté par un détecteur spécifique implémentant une interface commune.

```
class BaseDetector(ABC):
    @abstractmethod
    def detect(self, graph: nx.DiGraph) -> List[FraudType]:
        pass

    @abstractmethod
    def get_name(self) -> str:
        pass
```

7.2 Pattern Builder (Construction de Graphes)

Le GraphBuilder utilise le pattern Builder pour permettre différentes configurations de

8. Risques et Mitigations

Risque	Impact	Mitigation
Performance sur grands datasets	Détection lente	Optimisation des algorithmes, traitement par batch
Faux positifs élevés	Mauvaise expérience utilisateur	Ajustement des seuils, apprentissage continu
Complexité de visualisation	Graphes illisibles	Filtrage, zoom sur sous-graphes, layouts hiérarchiques
Qualité des données	Détections incorrectes	Validation rigoureuse, nettoyage des données
Portabilité du code	Difficulté d'exécution sur autres machines	Virtualisation (Docker), requirements stricts

9. Roadmap d'Implémentation

```
gantt
    title Roadmap d'Implémentation
    dateFormat YYYY-MM-DD
    section Phase 1
    Structure de projet         :done, p1, 2024-01-10, 1d
    section Phase 2
    Module de chargement        :active, p2, 2024-01-11, 2d
    Module de construction      :p3, after p2, 2d
    section Phase 3
    Détection cycles            :p4, after p3, 3d
    Détection smurfing          :p5, after p4, 3d
    Détection anomalies         :p6, after p5, 3d
    section Phase 4
    Module de visualisation     :p7, after p6, 2d
    Module d'évaluation         :p8, after p7, 2d
    section Phase 5
    Tests unitaires             :p9, after p8, 2d
    Notebooks démonstration     :p10, after p9, 2d
    Documentation finale        :p11, after p10, 2d
```

10. Conclusion

Cette architecture technique propose une solution adaptée aux contraintes d'un projet académique tout en permettant une démonstration solide des concepts de détection de fraude par graphes.

Points clés de l'architecture :

1. **Simplicité** : Utilisation de NetworkX et bibliothèques Python standard
2. **Modularité** : Séparation claire des responsabilités entre modules
3. **Extensibilité** : Facile d'ajouter de nouveaux détecteurs ou métriques
4. **Démonstrabilité** : Interface CLI, API Python et notebooks Jupyter
5. **Évaluabilité** : Module d'évaluation complet avec métriques standards

L'architecture est conçue pour être implémentée progressivement, chaque module pouvant être développé et testé indépendamment avant l'intégration finale.

Document d'architecture technique rédigé pour le groupe 42 - ECE Ing4 - Janvier 2026