# Documentation Technique Approfondie

Projet : Détection d'Anomalies Comportementales et Géographiques dans les Logs de Connexion via des Modèles de Graphe Neuronaux (GAE/VGAE)

Documentation Technique

20/06/2025

**Résumé**

Ce document expose en détail l'architecture technique, les choix méthodologiques et l'implémentation d'un pipeline de détection d'anomalies dans des journaux d'événements de connexion. L'objectif principal est d'identifier des activités malveillantes sophistiquées, telles que le **credential stuffing**, les attaques par **brute force distribuée**, ou les **compromissions de comptes**, qui sont souvent difficiles à détecter avec des méthodes statistiques classiques ou basées sur des règles prédéfinies. En modélisant les interactions comme un graphe hétérogène et en appliquant des modèles de Graphe Auto-Encodeur (GAE/VGAE), nous apprenons la structure « normale » des connexions pour identifier les déviations significatives.

## Table des matières

# 1 Introduction Générale et Contexte Métier

Ce document a pour vocation de servir de référence pour les ingénieurs, les data scientists et les architectes impliqués dans le projet, en fournissant une traçabilité complète des décisions techniques.

Les approches traditionnelles de détection d'anomalies échouent car elles analysent les événements de manière isolée. Or, la nature des attaques modernes est relationnelle : un même acteur (IP) peut cibler plusieurs comptes, ou un même compte peut être attaqué depuis de multiples origines (IPs, User-Agents). L'hypothèse fondamentale de ce projet est que la structure des relations entre les entités de connexion (utilisateurs, adresses IP, user-agents, pays) recèle des informations cruciales pour distinguer un comportement légitime d'un comportement anormal.

Pour capturer cette complexité relationnelle, nous avons modélisé le système sous la forme d'un **graphe hétérogène**. Les modèles de **Graphe Auto-Encodeur (GAE)** et **Graphe Auto-Encodeur Variationnel (VGAE)** sont ensuite utilisés pour apprendre une représentation latente (embedding) de la structure « normale » de ce graphe. Les anomalies sont alors identifiées comme des nœuds ou des sous-graphes que le modèle peine à reconstruire, indiquant une déviation par rapport aux motifs appris.

# 2 Prétraitement et Ingénierie des Caractéristiques (`pretraitement.ipynb`)

Cette étape est fondamentale car la qualité du graphe et, par conséquent, du modèle, dépend directement de la richesse et de la pertinence des données qui lui sont fournies.

## 2.1 Source des Données

— **Fichier source** : `logs/user_events_with_geoip_25k.csv`
— **Description** : Ce fichier CSV contient des enregistrements d'événements de connexion.
— **Colonnes clés utilisées** : `username`, `ip_address`, `user_agent`, `event_type`, `timestamp`, `country_name`, `city_name`.

## 2.2 Nettoyage et Préparation

1. **Chargement des données** avec la bibliothèque `pandas`.
2. **Gestion des valeurs manquantes** : Suppression des lignes avec `username` ou `ip_address` manquants.
3. **Conversion des types** : Le `timestamp` est converti en objet `datetime`.

## 2.3 Ingénierie des Caractéristiques (Feature Engineering)

### 2.3.1 Caractéristiques Comportementales

**Fréquence de Connexion (`connection_frequency`)** Pour chaque utilisateur, on compte le nombre total d'événements.

$$\mathit{freq}(\text{user}_u) = |\{e \mid e.\text{username} = \text{user}_u\}|$$

**Diversité des User-Agents (`user_agent_diversity`)** Pour chaque utilisateur, on compte le nombre d'`user_agent` uniques.

$$\mathit{diversity}(\text{user}_u) = |\{\text{ua} \mid \exists e : e.\text{username} = \text{user}_u \wedge e.\text{user\_agent} = \text{ua}\}|$$

**Taux d'Échec par IP (`ip_failure_rate`)** Pour chaque IP, on calcule le ratio d'échecs.

$$\mathit{fail\_rate}(\text{ip}_i) = \frac{|\{e \mid e.\text{ip} = \text{ip}_i \wedge e.\text{type} = \text{fail}\}|}{|\{e \mid e.\text{ip} = \text{ip}_i\}|}$$

#### 2.3.2 Agrégation et Sauvegarde

Le DataFrame enrichi est sauvegardé dans `logs/logs_events_clean.csv`.

# 3 Construction du Graphe Hétérogène (`construcion_graphe.ipynb`)

Cette étape traduit les données tabulaires en une structure de graphe via **PyTorch Geometric (PyG)**.

## 3.1 Définition Formelle du Graphe

Nous construisons un graphe hétérogène $G = (V, E)$ où :

— $V$ est l'ensemble des nœuds, partitionné en types : $V = V_{\text{user}} \cup V_{\text{ip}} \cup V_{\text{ua}} \cup V_{\text{country}}$.

— $E$ est l'ensemble des arêtes typées, représentant les interactions.

## 3.2 Typologie des Nœuds et de leurs Attributs

**Nœuds** `user` Attributs : `connection_frequency`, `user_agent_diversity`.

**Nœuds** `ip` Attributs : `ip_failure_rate`.

**Nœuds** `user_agent` **et** `country` Attributs constants.

## 3.3 Création des Arêtes (Relations)

— (`user`, `logs_in_from`, `ip`)

— (`user`, `uses_agent`, `user_agent`)

— (`ip`, `located_in`, `country`)

## 3.4 Encodage et Normalisation des Caractéristiques

Une **normalisation Min-Max** est appliquée, ramenant les valeurs à l'intervalle $[0, 1]$.

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

## 3.5 Sauvegarde de l'Objet Graphe

— `construction/credential_stuffing_graph_v4.pt`

— `construction/node_mapping_v4.pt`

# 4 Modélisation et Apprentissage (`gae_model.ipynb`)

## 4.1 Fondements Théoriques : GAE et VGAE

### 4.1.1 Le Graphe Auto-Encodeur (GAE)

— **Encodeur** : Un GCN produit des embeddings de nœuds $Z$ à partir de $A$ et $X$.

$$H^{(l+1)} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)})$$

— **Décodeur** : Reconstruit la matrice $\hat{A}$ via le produit scalaire : $\hat{A}_{ij} = \sigma(z_i^T z_j)$.

— **Fonction de Perte** : Entropie croisée binaire entre $A$ et $\hat{A}$.

$$\mathcal{L}_{\text{BCE}} = -\sum_{i,j}\Big[A_{ij}\log(\hat{A}_{ij}) + (1 - A_{ij})\log(1 - \hat{A}_{ij})\Big]$$

#### 4.1.2 Le Graphe Auto-Encodeur Variationnel (VGAE)

Extension probabiliste où l'encodeur apprend une distribution $(\mu, \sigma^2)$ pour chaque embedding.

— **Fonction de Perte (ELBO)** :

$$\mathcal{L}_{\text{VGAE}} = \mathcal{L}_{\text{BCE}} + \beta \cdot D_{\text{KL}}(q(Z|X, A) \,\|\, p(Z))$$

### 4.2 Implémentation du Modèle

Le modèle est implémenté en PyTorch/PyG et entraîné en *full-batch* avec l'optimiseur `Adam`.

### 4.3 Calcul du Score d'Anomalie

Le score est l'erreur de reconstruction individuelle de chaque nœud.

## 5 Analyse des Résultats et Validation (`results_v4_behavioral/`)

### 5.1 Identification des Anomalies Principales

Les nœuds sont triés par score d'anomalie. Les plus suspects sont dans `results_v4_behavioral/top_anomali`

### 5.2 Validation et Interprétabilité

L'analyse combine le score, les caractéristiques et les relations pour comprendre pourquoi un nœud est anormal.

### 5.3 Visualisation (t-SNE)

L'algorithme t-SNE projette les embeddings en 2D pour une validation visuelle.

## 6 Architecture, Reproductibilité et Évolutions

### 6.1 Dépendances et Environnement

Les dépendances sont listées dans `requirements.txt`.

### 6.2 Instructions pour la Reproductibilité

1. Installer les dépendances : `pip install -r requirements.txt`.
2. Exécuter les notebooks : `pretraitement.ipynb`, `construcion_graphe.ipynb`, `gae_model.ipynb`.

### 6.3 Pistes d'Amélioration et Travaux Futurs

— **Passage à l'échelle** : Entraînement par mini-batch (*neighbor sampling*).
— **Modèles plus complexes** : GAT, RGCN, HGT.
— **Aspect temporel** : Graphes dynamiques (Dynamic GNNs).
— **Enrichissement des caractéristiques** : Données externes (réputation d'IP, ASN).