Master Mathématiques Appliquées Statistique, Parcours Statistique, Modélisation et Sciences des Données (**SMSD**)

Détection d'anomalies sur les graphes de connaissances pour l'identification de signaux faibles

Analyse structurelle de graphes temporels au sein de l'équipe DM2L (Data Mining et Machine Learning)

Étudiant :
Antoine SALAZAR

Sous la direction de :
Ludovic MONCLA
Enseignant Chercheur

Laboratoire d'**I**nfo**R**matique en **I**mage et **S**ystèmes d'information

Année universitaire 2025 – 2026

Table des matières

Remerciements

Ce rapport fait état des missions auxquelles j'ai pu participer durant mon stage de fin de master. Je remercie mon encadrant principal, M. **Ludovic MONCLA**, qui par sa gentillesse et ses conseils avisés m'ont permis de pouvoir réaliser ce stage dans les meilleures conditions. Mes remerciements s'adressent également à M. **Yassir LAIRGI**, doctorant à l'INSA Lyon, M. **Remy CAZABET**, maître de conférences à l'université Lyon 1 ainsi qu'à M. **Khalid BENABDESLEM**, maître de conférences à l'université Lyon 1 et co-encadrants avec qui j'ai eu le plaisir de travailler pendant cette période.

Parce que ce rapport constitue le dernier projet que je réalise en temps qu'étudiant en mathématiques, c'est avec beaucoup de satisfaction que j'écris celui-ci, fier d'être arrivé jusqu'ici dans mon parcours. Je souhaiterais donc remercier les différents responsables de formations que j'ai côtoyés durant ces 3 années passées à l'Université Claude Bernard Lyon 1 : M. **Lorenzo BRANDOLESE**, responsable de la licence de mathématiques, M. **Christophe POQUET**, responsable du M1 Mathématiques appliquées, statistique, et Mme. **Gabriela CIUPERCA** responsable du M2 SMSD.

De manière générale, mes remerciements s'adressent à tous les enseignants du département de mathématiques que j'ai pu rencontrer lors de mon parcours au sein de cette université et qui m'ont fait comprendre, à leur manière, que le raisonnement mathématique a des frontières bien plus larges que celles d'un amphithéâtre ou d'une salle de classe.

1 Contexte et environnement professionnel

1.1 Présentation du LIRIS

Le **L**aboratoire d'**I**nfo**R**matique en **I**mage et **S**ystèmes d'information (**LIRIS**) est une unité mixte de recherche (UMR 5205) du **CNRS**, de l'**INSA** de Lyon, de l'Université Claude Bernard **Lyon 1**, de l'Université Lumière **Lyon 2** et de l'**Ecole Centrale** de Lyon. Il compte 330 membres. Les recherches du **LIRIS** concernent un large spectre de la science informatique au sein de ses douze équipes de recherche structurées en six pôles de compétences :

- Données, Système et Sécurité (équipes BD, DRIM, SOC et DM2L)
- Informatique Graphique et Géométrie (équipe ORIGAMI)
- Images, Vision et Apprentissage (équipe IMAGINE)
- Interactions et cognition (équipes SICAL, SyCoSMA et TWEAK)
- Algorithmique et Combinatoire (équipe GOAL)
- Simulation et Sciences du Vivant (équipes SAARA et BEAGLE)

Les recherches menées relèvent les défis du monde numérique, notamment ceux posés par l'intelligence artificielle (**IA**), l'analyse de données volumineuses (Big Data), la vision par ordinateur, la cyber-sécurité, la transformation digitale ou l'apprentissage humain. Une partie des activités du **LIRIS** se situent aux interfaces des sciences humaines et sociales, de l'ingénierie, de la médecine, des sciences de la vie et des sciences de l'environnement. Par ailleurs, le **LIRIS** accorde aussi une grande importance à la médiation scientifique en informatique pour le grand public.

Enfin, le **LIRIS** s'implique dans les défis sociétaux de la souveraineté numérique et du développement durable, principalement à travers l'utilisation responsable des technologies numériques et la prise de conscience de l'impact carbone des activités de recherche usuelles.

1.2 Présentation de l'équipe DM2L

Lors de ce stage j'étais intégré à l'équipe **DM2L** (**D**ata **M**ining et **M**achine **L**earning).
DM2L est une équipe scientifique créée en 2012 dont les activités de recherche sont consacrées à l'extraction de connaissances à partir de données, grâce à des techniques automatiques ou semi-automatiques. Ces techniques incluent l'exploration de données, l'apprentissage automatique, la reconnaissance de formes, l'analyse de données, le traitement automatique du langage naturel, etc.
Cette équipe se compose de 31 membres et une organisation simplifiée au sein de celle-ci peut être représentée suivant :

FIGURE 1 – Logo de l'équipe D2ML

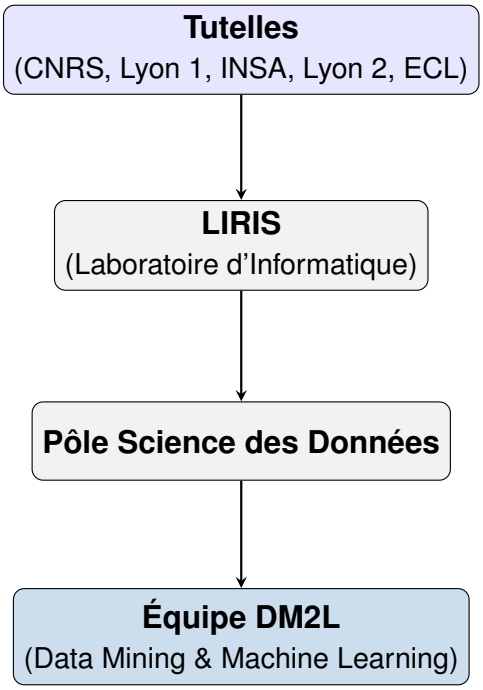


FIGURE 2 – Environnement de recherche au LIRIS

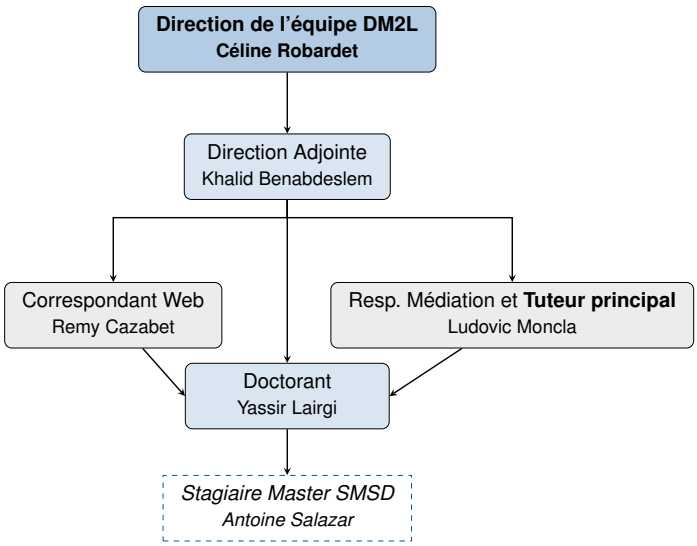


FIGURE 3 – Organisation interne de l'équipe DM2L

1.3 Contexte et objectifs du stage

L'analyse et l'exploitation des données constitue un élément essentiel notamment dans les secteurs de la santé, de la finance ou encore de la défense par exemple.

Parmi toutes les manières de synthétiser les données, une retient notre attention : **les Graphes de connaissance** (en anglais **Knowledge Graphs**). L'objectif de ces graphes est de synthétiser l'ensemble des relations (représentées par les arêtes) liant des entités (représentées par des nœuds). Ces graphes peuvent inclure une dimension temporelle - **Temporal Knowledge Graphs (TKG)** - et dynamique - **Dynamic Temporal Knowledge Graphs (DTKG)**.

L'objectif majeur des premiers travaux réalisés avant mon arrivée résidait dans la capacité à **construire des graphes de connaissances capables de s'adapter dynamiquement aux derniers changements**, notamment dans le domaine de la finance.
LAIRGI et al. [LMB+24] ont donc proposé une première méthode, **iText2KG**, qui permet de construire un graphe de connaissance à partir de n'importe quelle source de texte brut.

Cette méthode a ensuite été améliorée dans un deuxième article [LMB+26], qui présente la méthode **ATOM**. Cette dernière consiste à décomposer les évènements entrants en faits "atomiques" et utiliser une modélisation double temps (dual-time modeling) pour prendre en compte la temporalité des faits et ainsi faciliter la modification du graphe.

L'objectif de ce stage s'inscrit donc dans la continuité des travaux précédemment menés par notre équipe. Après avoir tracé des graphes de connaissances prenant en compte la dimension temporelle des évènements, il est désormais question de **détecter les signaux faibles** présents dans celui-ci, permettant d'améliorer les prédictions.

Lors de ce stage, j'ai donc un rôle d'équipier de **Yassir LAIRGI**, doctorant à l'INSA Lyon qui dans le cadre de sa thèse, travaille sur les graphes de connaissances. Nos recherches se font sous la responsabilité de **Ludovic MONCLA**, tuteur principal de ce stage et maître de conférences à l'INSA Lyon, **Khalid BENABDESLEM**, maître de conférences et directeur adjoint de l'équipe DM2L et **Remy CAZABET**, maître de conférences à l'INSA Lyon.

2 Premières définitions

L'apprentissage automatique est un sous-domaine de l'intelligence artificielle (**IA**) qui se concentre sur la conception et le développement d'algorithmes capables d'apprendre à partir de données. Contrairement à la programmation classique où les règles sont explicitement dictées par l'humain, un modèle de Machine Learning identifie des motifs (patterns) et extrait des règles implicites directement à partir d'exemples. Dans le cadre de ce projet, ces techniques (notamment via l'apprentissage supervisé) sont exploitées pour apprendre la structure et la sémantique des graphes de connaissances afin de détecter des anomalies potentielles.

Nous définissons tout d'abord un graphe de connaissances :

Un graphe de connaissances $\mathcal{G}$ (**Knowledge Graph** en anglais) est un graphe représentant une collection de triplets (e_s, r, e_o) où $e_s \in \mathcal{E}$ (l'ensemble des entités) représente l'entité sujet, $e_o \in \mathcal{E}$ désigne l'entité objet et $r \in \mathcal{R}$ (l'ensemble des relations) représente la relation reliant ces deux entités.
L'objectif d'un graphe de connaissances est donc de synthétiser l'ensemble des relations et/ou des interactions entre différents objets. Ils sont donc très utilisés dans les domaines de la santé, où encore pour l'analyse de marchés financiers par exemple.

Cependant, dans ce dernier domaine notamment, la prise en compte de la dimension temporelle est nécessaire. On introduit pour cela la définition d'un graphe de connaissances temporel.

Selon [WSL+24], **un graphe de connaissances temporel** (**Temporal Knowledge Graph** en anglais) $\mathcal{G} = \{\mathcal{E}, \mathcal{R}, \mathcal{T}, \mathcal{Q}\}$ est la réunion d'un ensemble $\mathcal{E}$ désignant l'ensemble des entités, de $\mathcal{R}$ désignant l'ensemble des relations, et enfin d'un intervalle de temps $\mathcal{T}$. En particulier, chaque quadruplet $(e_s, r, e_o, t) \in Q$ où $e_s, e_o \in \mathcal{E}$ représentent les entités, $r \in \mathcal{R}$ représente la relation et $t \in \mathcal{T}$ représente l'horodatage.

Enfin, nous définissons la notion de signal faible. **Un signal faible** est une information de faible intensité, un indice discret ou un fait isolé qui annonce à l'avance une tendance, un changement important ou un risque futur (crise, défaillance d'entreprise, changement politique...). Pris isolément, il semble insignifiant, mais combiné à d'autres, il permet d'anticiper un événement majeur.

3 Etat de l'art des différents articles étudiés sur les graphes de connaissances

Dans cette section seront présentés les différents articles étudiés lors de ce stage et qui traitent du raisonnement sur les graphes de connaissances.

Il existe aujourd'hui 3 grandes thématiques :

- L'amélioration de la construction des graphes de connaissances.
- Le raisonnement sur les graphes de connaissances, dans le but de détecter des anomalies par exemple, ou de rechercher efficacement des informations sur les graphes.
- Les raisonnements qui prennent en compte la dimension temporelle en s'intéressant aux graphes de connaissances temporelles.

Ce stage a donc débuté par une lecture d'articles scientifiques centrés sur ces 3 thématiques. Ces articles sont disponibles dans la bibliographie en fin de rapport.
Les articles étudiés portant sur la première thématique (l'amélioration de la construction des graphes de connaissances), retracent les travaux de **Yassir Lairgi**, en présentant deux méthodes de construction de graphes de connaissances temporelles (**iText2KG** [LMB+24] et **ATOM** [LMB+26]). Ces travaux, de par leur capacité à améliorer la construction de ce type de graphes, permettent de simplifier le raisonnement et la recherche d'informations sur ceux-ci par les modèles de Deep Learning mais également par les **LLM**.

Notre travail de synthèse bibliographique a surtout porté sur les deux dernières thématiques. Nous allons revenir sur celles-ci dans les sous-sections suivantes.

3.1 Le raisonnement sur les graphes de connaissances (KG reasoning)

Le raisonnement sur les graphes de connaissances (KG Reasoning) consiste en l'ensemble des actions réalisées sur les graphes de connaissances permettant entre autres, de rechercher l'information, d'incorporer de nouveaux faits ou encore de détecter des anomalies.

Plusieurs méthodes existent pour améliorer le raisonnement sur les graphes de connaissances. On peut les rassembler en deux groupes :

— Un premier groupe qui améliore la combinaison des **LLM** (Large Language Model) avec les graphes de connaissances.
— Un second groupe rassemblant les méthodes qui créent une structure intermédiaire à partir du graphe originel pour effectuer leur raisonnement, en s'appuyant notamment sur des **réseaux de neurones sur graphes** (en anglais **G**raph **N**eural **N**etwork, abrégé en **GNN**).

3.1.1 L'amélioration de la collaboration LLM-KG

Les **LLM** (**L**arge **L**anguage **M**odel) sont un type de programme d'intelligence artificielle (**IA**) capable, entre autres tâches, de reconnaître et de générer du texte. Les LLM sont entraînés sur d'immenses ensembles de données, d'où l'emploi du terme « large » (grand) dans la dénomination anglaise. Ils reposent majoritairement aujourd'hui sur l'architecture Transformers, basée sur un mécanisme d'attention, qui permet notamment de comprendre la sémantique des mots dans un corpus de texte et donc de gagner en précision dans le raisonnement. Ils peuvent donc être utilisés pour le raisonnement sur graphe de connaissances, où chaque triplet (sujet, relation, objet) peut être interprété comme une phrase (ex : "Paris", "Capitale de", "France").

Cependant, **les LLM possèdent deux limites majeures** : la première concerne leur **manque d'interprétabilité**. Les LLM agissent en "boîte noire" ; s'il est facile de déterminer ce qui est analysé par le LLM (input), il est difficile d'expliquer le raisonnement qui lui permet de déterminer la réponse (output). Ce manque d'interprétabilité peut parfois conduire à des hallucinations de la part du LLM, qui va générer des réponses fausses.
La deuxième limite concerne **la difficulté à paramétrer les LLM**. Leur architecture reposant sur un très grand nombre de paramètres (généralement au-dessus d'un milliard), le réglage fin (fine tuning) de ces modèles est très coûteux en calcul et en énergie.

L'objectif est donc de faire collaborer les **LLM** avec les **KG** (**K**nowledge **G**raphs) pour combler les limites engendrées par l'utilisation des **LLM** seuls. Les **KG**, en servant de base de connaissances pour les **LLM** permettent d'obtenir plus de transparence quant au raisonnement généré par ceux-ci. Par ailleurs, en cas d'incohérence du **LLM**, il est plus facile de modifier le graphe de connaissances pour faciliter le raisonnement du **LLM** que de modifier directement ce dernier en raison de la complexité du réglage fin.

Par conséquence, l'efficacité de la collaboration entre les LLM et les graphes de connaissances résulte d'un équilibre fragile :

— Une surdépendance vis-à-vis des LLM entraînera des difficultés d'interprétabilité et de transparence quant au raisonnement effectué, et des difficultés de paramétrages du modèle.
— Une surdépendance vis-à-vis du graphe de connaissances entraînera un manque d'efficacité de par la non-utilisation des pleines capacités du LLM. Enfin, un raisonnement axé sur les graphes dépend principalement de la qualité de construction de ces derniers. Or un graphe mal construit entraînera toujours une baisse de qualité dans le raisonnement qui suivra.

Historiquement, la collaboration entre les LLM et les KG fonctionnait ainsi :

— L'utilisateur pose une question (invite)
— Les informations du KG sont récupérées
— L'invite est augmentée en conséquence
— On introduit cette nouvelle invite dans les LLM pour qu'il puisse formuler sa réponse

Cette approche pourrait être nommée comme le système **LLM + KG**. Cependant, cette approche possède une limite : ici le LLM ne joue que le rôle de traducteur qui transfère les questions pour le raisonnement KG et renvoie les réponses à partir des informations récupérées sur le graphe par le système. Le LLM dans une telle approche ne participe pas au raisonnement sur graphe.

Ce pourquoi l'article "**Think-On-Graph (TOG) : deep and responsible reasoning of Large Language Model On Knowledge Graph, J.Sun et al, ICLR (2024)**" [S+24] propose une méthode dans laquelle le LLM va "penser" (**Think**) le long des chemins du graphe (**on Graph**). Ce nouveau raisonnement s'appelle désormais **LLM** $\times$ **KG**. Les LLM et les KG fonctionnent en tandem. Cette méthode fonctionne en **3 temps** :

Dans un premier temps consacré à l'**initialisation**, le modèle identifie d'abord l'entité de départ dans le graphe à partir de la question posée. Cette étape définit le point d'ancrage nécessaire pour construire les premiers chemins de raisonnement.

Ensuite, le modèle effectue une **exploration itérative**. Le système construit progressivement des chemins de raisonnement en explorant le graphe étape par étape. À chaque itération, le LLM analyse les relations et les entités voisines disponibles dans le graphe pour sélectionner celles qui sont les plus pertinentes par rapport à la question. Ce processus de sélection (recherche et tri) permet d'étendre les chemins de manière ciblée, en s'assurant que chaque ajout apporte une valeur ajoutée au raisonnement.

Enfin, après chaque étape d'exploration, le **LLM évalue la pertinence des chemins construits**. Si les informations recueillies sont jugées suffisantes, le modèle génère une réponse finale basée sur ces preuves. Dans le cas contraire, le processus d'exploration se poursuit jusqu'à l'obtention d'un résultat satisfaisant ou jusqu'à ce qu'une limite d'itérations soit atteinte. En l'absence de chemin concluant, le modèle s'appuie alors sur ses connaissances internes pour

formuler une réponse.

En résumé, ToG transforme la recherche d'information en un voyage structuré à travers le graphe, où le LLM agit comme un guide qui décide de la direction à prendre à chaque intersection, garantissant ainsi que la réponse finale est appuyée par des faits vérifiables extraits du graphe.

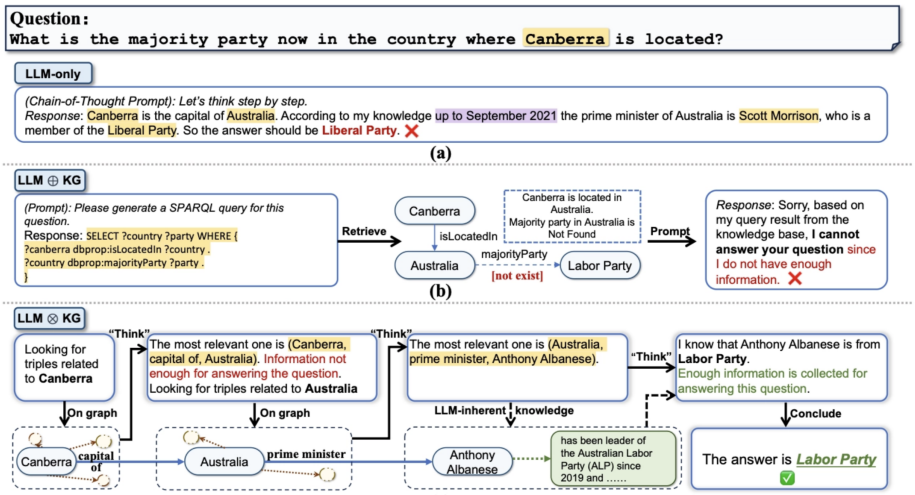


FIGURE 4 – Description de la méthode TOG (illustration tirée de l'article)

Dans une démarche similaire, l'article "**Debate on Graph (DOG) : a flexible and reliable reasoning framework for Large Language Models, J. Ma et al, AAAI (2024)**" [M+24] propose un **Knowledge Graph Question Answering (KGQA)** interactif et itératif qui tire parti des capacités d'apprentissage interactif des LLM pour effectuer le raisonnement et le débat sur les graphes (**DOG**).

La méthode (**DOG**) consiste en un mécanisme de mise au point sur les sous graphes permettant au LLM d'effectuer des essais de réponses à chaque étape du raisonnement atténuant ainsi l'impact des longs chemins de raisonnement.

DOG utilise également une équipe de "débatteurs" multirôles pour simplifier progressivement les questions complexes réduisant ainsi l'influence des relations faussement positives.

Etant donné une question k-hop q et l'entité initiale e_i^l dans q, la méthode DoG raisonne en 4 étapes :

— KG invoking : Elle invoque d'abord les KG pour récupérer l'ensemble des relations candidates R liées à e_i^l
— Relation filtering : les LLM vont filtrer la relation la plus pertinente en fonction de l'apprentissage et du contexte.
— Le KG est ensuite utilisé pour compléter la triple info de $(e_i^l, r_l, ?)$ vers (e_i^l, r_l, e_{i+1}^l)

— DoG se concentre sur l'état actuel du raisonnement et utilise le LLM pour décider de l'action suivante en fonction du triple obtenu. De là, 2 actions sont possibles : fournir une réponse directe à la question ou effectuer une réflexion + approfondie avec d'autres itérations.

En ce qui concerne la dernière c'est ici que la notion de débat prend tout son sens. Si le LLM ne peut pas apporter de réponse à la question initiale, il va la simplifier pour transformer la question initiale (k-hop) en question $k-1$ hop. 3 agents vont donc intervenir dans ce cadre :

— Question simplifying expert
— Critic
— Linguist

Ces 3 agents vont débattre afin d'éviter les raccourcis erronés et permettre d'avoir la meilleure simplification possible sur lequel le LLM pourra à nouveau raisonner.

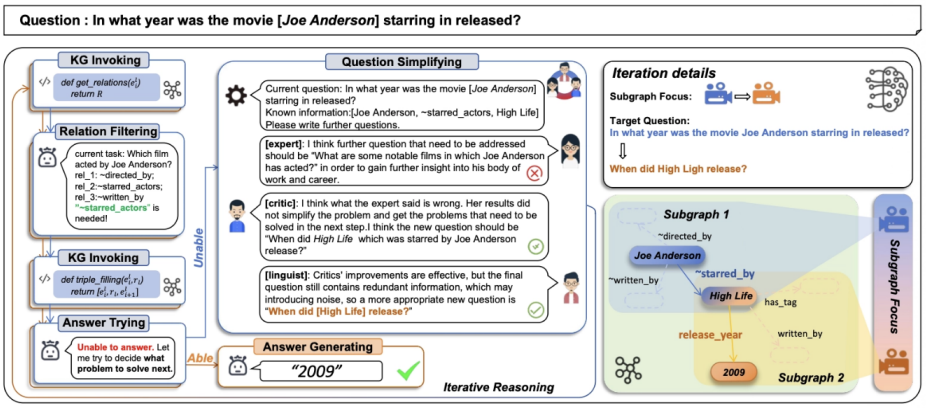


FIGURE 5 – Description de la méthode DoG

Les articles présentés constituent des alternatives cohérentes au problème d'interprétabilité des LLM en s'appuyant sur les graphes de connaissances comme support.
Toutefois, lors de ce stage d'autres méthodes ont été explorées, notamment celles qui permettent un raisonnement sur les graphes de connaissances par le biais d'une construction de structures intermédiaires, plus propices à l'application de réseaux de neurones (GNN). Nous allons revenir en détail sur les principaux modèles étudiés lors de ce stage.

3.1.2 Les modèles de raisonnement sur des structures intermédiaires

Au regard de la complexité de certains graphes de connaissances, et dans un objectif d'explicabilité des raisonnements appliqués sur ceux-ci, certains articles proposent de créer une structure intermédiaire dans ces graphes pour pouvoir utiliser des outils de raisonnement plus transparents que les **LLM**. Ces outils sont généralement des réseaux de neurones, qui, appliqués sur les graphes, sont couramment appelés **GNN** (de l'anglais **G**raph **N**eural **N**etwork). Cette découverte lors de ce stage a donc constitué le point de départ de nos recherches. Nous avons donc essayé d'optimiser ces méthodes dans le but de les faire répondre à notre problématique : **la détection de signaux faibles**.

Dans cette sous partie, nous allons présenter les modèles que nous avons étudiés dans cette thématique. Dans un souci de transparence et de cohérence, nous évoquerons tous les articles étudiés, mais nous n'insisterons que sur les articles essentiels pour la suite de notre travail.

L'article "**Don't Generate, Discriminate : Proposal for Grounding Language Models to Real-World Environments, Y. Gu et al, ACL (2023)**" [G+23] est un premier exemple de l'efficacité de la construction de structures intermédiaires pour le KG reasoning. Il présente une nouvelle méthode : **PANGU**.

PANGU est un cadre générique pour la compréhension du langage qui se base sur la capacité discriminatoire du modèle de langage (**LM**) plutôt que sur sa capacité générative.

Dans ce contexte, un agent symbolique (qui génère des règles) et un LM neuronal travaillent de manière concertée.

L'agent explore l'environnement pour construire de manière incrémentale des plans valides, et le **LM** évalue la plausibilité des plans candidats pour guider le processus de recherche.

L'objectif de la méthode est de diminuer la dépendance envers le **LM** en lui enlevant des responsabilités et le laisser être ce qu'il est à l'origine (c'est à dire un outil attribuant des probabilités à une séquence de tokens).

— En partant d'un point initial p_0, à chaque étape, l'agent interagit avec son environnement E pour étendre les plans actuels dans un nouvel ensemble de plans candidats
— Ensuite le LM note les plans candidats et sélectionne le top K des plans pour une exploration plus poussée à la prochaine étape.

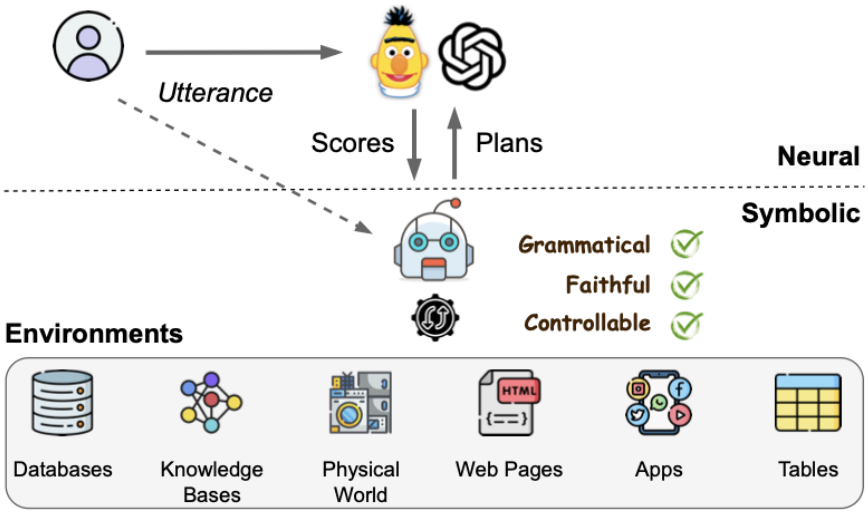


FIGURE 6 – Illustration de la méthode PANGU

Dans une démarche similaire, l'article "**Reasoning On Graphs (ROG) : Faithful and interpretable Large Language Model Reasoning, L. Luo et al, ICLR (2024)**" [L+24] propose également de générer des plans plutôt que de raisonner directement sur le graphe.

Cette nouvelle méthode qui combine **LLM** et **KG** pour un raisonnement fidèle et interprétable repose sur 3 piliers : **Planification - Récupération - Raisonnement**.

Planification : le modèle génère des chemins de relations fondés sur les KG en tant que plans fidèles.
Récupération : utilisation de ces plans pour récupérer des chemins de raisonnement valides des KG pour que les **LLM** mènent un raisonnement rapide.
Raisonnement : **RoG** utilise non seulement les connaissances du KG pour améliorer la capacité de raisonnement des **LLM** grâce à la formation, mais permet également une intégration transparente avec le **LLM** pendant l'inférence.

Cette méthode s'inscrit également dans un contexte de **KGQA** : On rappelle la définition figurant dans l'article.

— Etant donné une question q et un KG $\mathcal{G}$, la tâche vise à concevoir une fonction f pour prédire les réponses $a \in \mathcal{A}_q$ basée sur les connaissances de $\mathcal{G}$ (ie $a = f(q, \mathcal{G})$).

L'objectif ensuite est de répondre en langage naturel à cette question.

En traitant les chemins de relation comme des plans, nous pouvons nous assurer que ces plans sont ancrés sur des KG, ce qui permet aux **LLM** de mener un raisonnement fidèle et interprétable sur les graphes.

Autrement dit : **RoG** est un problème d'optimisation qui vise à maximiser la probabilité de raisonner la réponse à partir d'un KG $\mathcal{G}$ et d'une question q en générant des chemins de relations z.

A noter que cet article présente un double intérêt : en plus de créer des structures intermédiaires pour améliorer le raisonnement, il améliore également lors de son processus la combinaison entre les LLM et les KG. Cet article aurait pu également figurer dans la sous-partie précédente.

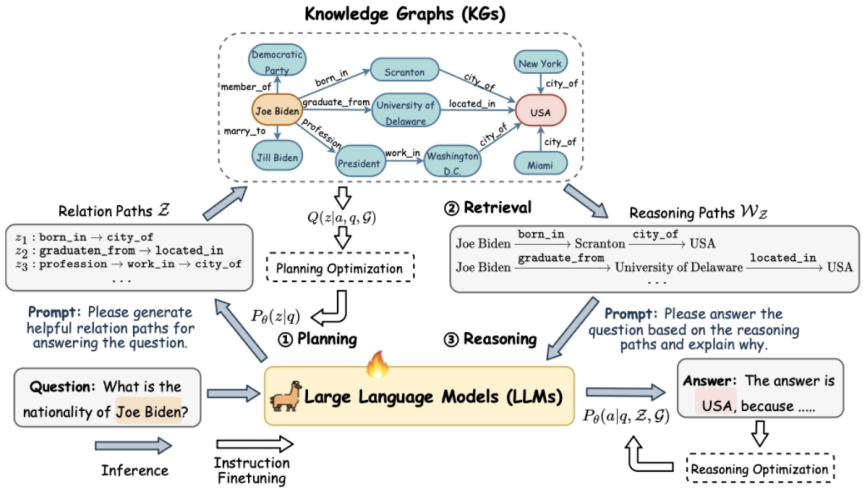


FIGURE 7 – Description de la méthode RoG

L'article "**G-Reasoner : Foundation models for unified reasoning over graph structured knowledge, L. Luo et al, ICLR (2026)**" [L+26] propose un raisonnement similaire. La différence réside dans la manière de construire le graphe intermédiaire sur lequel le **LLM** va raisonner.

G-reasoner un nouveau cadre qui intègre les graphes et les bases de modèles de langage pour permettre un raisonnement unifié sur diverses connaissances structurées en graphe.

Il repose sur 2 étapes :

- La construction de la structure du graphe
- La récupération améliorée par graphe et raisonnement LLM.

Concernant la structure du graphe, une nouvelle méthode apparaît, **QuadGraph**. L'objectif est de standardiser les diverses structures de KG de différents domaines dans un format unifié.

Quadgraph est une structure graphique composée de **4 couches** :

- **Attribute layer** (couche des attributs) : elle capture les attributs communs des nœuds ;
- **KG layer** (couche du KG) : elle représente les entités et leurs relations comme des triplets ;
- **Document layer** (couche des documents) : elle contient l'info textuelle non structurée ;
- **Community layer** (couche des "voisins" ou des groupes) : elle groupe les noeuds en communautés basées sur leurs similarités sémantiques ou leur connectivité structurelle pour fournir un niveau global d'information.

Une fois le modèle unifié, **G-Reasoner** propose un modèle de fondation alimenté par **GNN** qui raisonne sur le **QuadGraph** et fait des prédictions polyvalentes. **G-Reasoner** libère davantage la puissance des **GNN** en concevant un modèle de base graphique (**GFM**) plus généralisable qui :
1) Allie la topologie graphique et la sémantique du texte pour le raisonnement ;
2) Permet des prédictions polyvalentes sur les types de nœuds arbitraires.

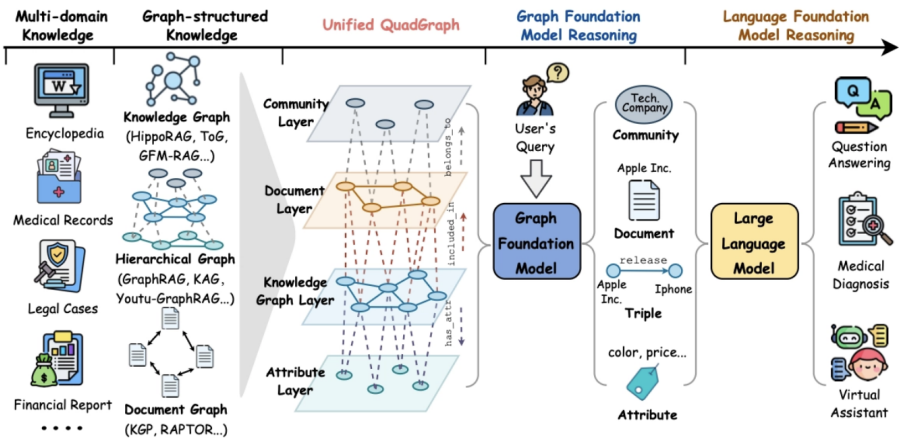


FIGURE 8 – Description du modèle G-Reasoner

Dans cette même optique de transformation de la structure originelle du graphe pour en extraire une meilleure représentation avant d'y appliquer un raisonnement, l'article "**LoGNet : Local and Global Triple Embedding Network, G. Pirro, ISWC (2022)**" [Pir22] propose une approche qui ne se concentre plus sur l'intégration (embedding) des nœuds de manière isolée, mais sur l'intégration des triplets eux-mêmes en tant qu'entités topologiques.

Cet article a constitué le point de départ de la méthode que nous avons implémentée dans la suite de ce stage pour la détection des signaux faibles. Ce pourquoi nous choisissons dans ce rapport d'en présenter de manière précise le cadre mathématique qui sera réutilisé par la suite dans notre méthode.

L'idée fondamentale de LoGNet est que **la sémantique d'un triplet** (h, r, t) **dépend fortement de son voisinage local (les triplets adjacents) mais aussi de sa position globale dans la structure du graphe**. Pour capturer cela, la méthode construit un réseau de neurones qui évolue sur une structure intermédiaire : un graphe de triplets (appelé line graph ou triple line graph). Le fonctionnement de LoGNet se divise en deux niveaux d'agrégation :

— **L'agrégation locale (Local)** : Pour un triplet donné, le modèle agrège les informations des éléments qui constituent celui-ci.

Etant donné un triple $t = (s, p, o) \in \mathcal{G}$, on définit une fonction de mapping

$$Emb_L : Agg(Emb(s), Emb(p), Emb(o)) \rightarrow \mathbb{R}^d$$

où $Emb(.)$ est une fonction d'embedding qui mappe les entités ou les prédicats dans un vecteur D-dimensionnel.

Toutefois, cette approche seule comporte des limites : par exemple, elle ignore souvent la nature séquentielle du triple, qui a un rôle important dans les KG. Egalement, il y a une nécessité de prendre en compte le contexte des triples en termes de voisinage. Ce pourquoi on considère une approche plus globale, expliquée dans le point suivant.

— **L'agrégation globale (Global)** : Le modèle capture des motifs structurels plus larges à l'échelle du graphe en utilisant des mécanismes d'attention pour pondérer l'importance de triplets plus éloignés mais sémantiquement liés.

On donne le cadre mathématique simplifié de cette méthode :

Soit un triplet cible $T_i = (e_h, r, e_t)$. On définit son voisinage local $\mathcal{N}(T_i)$ comme l'ensemble des triplets partageant au moins une entité avec T_i.

La mise à jour de l'embedding du triplet $\boldsymbol{h}_{T_i}$ à la couche l s'effectue via un réseau de neurones sur graphe (GNN) adapté :

$$h_i^{(l+1)} = \sigma\left(\sum_{j=1}^{N} \alpha(v_i, v_j) \cdot h_j^{(l)} \cdot W^{(l)}\right), \quad l = 0, \ldots, L-1$$

où $h_i^{(l)}$ est l'embedding du noeud v_i à la couche l ; $\alpha(v_i, v_j) \in \mathbb{R}^{N \times N}$ est une matrice de poids ($\neq$ si v_j est un voisin direct de v_i, 0 sinon) ; $W(l) \in \mathbb{R}^{D(l)} \times \mathbb{R}^{D(l+1)}$ est la matrice de transformation à la couche l ; σ est la fonction d'activation.

On construit à présent le line-graph, dont on présente maintenant la définition.

Line-graph : Etant donné un graphe non orienté $\mathcal{G} = \{\mathcal{V}_G, \mathcal{E}_G\}$, le line graph correspondant $\mathcal{G}_L$ est tel que :
— Chaque triplet de $\mathcal{G}_L$ représente un nœud de $\mathcal{G}$.
— Deux noeuds de $\mathcal{G}_L$ ont un noeud de $\mathcal{G}$ en commun.

Détection des prédicats anormaux :

Cette méthode permet également de détecter des prédicats anormaux en calculant des scores de plausibilité locaux (local plausibility score) et globaux (global plausibility score).

La plausibilité se définit enfin comme la combinaison linéaire de la plausibilité locale et globale.

On a donc : $c(s, p, o) = \alpha c_l + \beta c_g$

où c_l est la plausibilité locale, c_g la plausibilité globale et α et $\beta = 1 - \alpha$ mesurent l'importance des 2 scores normalisés.

Pour optimiser le modèle, on tire parti d'une fonction de perte basée sur la marge pour distinguer les triples positifs des négatifs.

$\mathcal{L} = \sum_{(s,p,o) \in T_G^+} \sum_{(s',p,o') \in T_G^-} [\theta + c(s, p, o) - c(s', p, o')]$

où θ est l'hyperparamètre de marge T_G^+ est l'ensemble des triples positifs et T_G^- est l'ensemble des triples négatifs.

A ce stade, avec le modèle entraîné, on peut évaluer l'anomalie d'un prédicat dans un triple (s, p, o) en calculant son score de plausibilité. On a donc : $P_s(s, p, o) = c(s, p, o)$

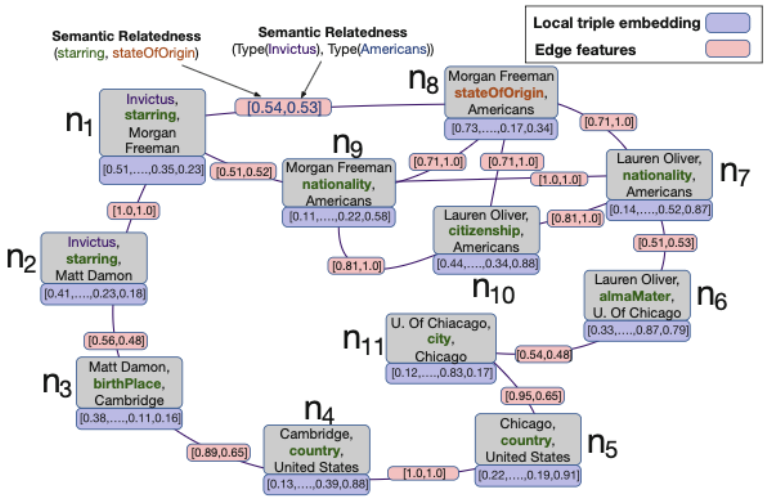


FIGURE 9 – Description de la structure "triple line graph" présentée dans le modèle LogNet

Enfin, un dernier modèle peut être évoqué à titre informatif, bien que peu utilisable au regard de l'objectif initial fixé lors de ce stage.

Dans une démarche poussant la restructuration du graphe encore plus loin pour résoudre le problème spécifique de l'apprentissage inductif, l'article "**INGRAM : Inductive Knowledge Graph Embedding via Relation Graphs, Lee et al, NeurIPS (2023)**" [L+23] propose une approche permettant de générer des embeddings non seulement pour de nouvelles entités mais également pour de nouvelles relations au moment de l'inférence.

La majorité des modèles classiques d'intégration de graphes de connaissances (KGE) opèrent dans un cadre **transductif**, supposant que toutes les entités et relations sont observées durant l'entraînement. Si des méthodes inductives récentes ont permis de traiter de nouvelles entités (souvent via des **GNN** ou des règles logiques), presque toutes échouent lorsque des **nouvelles relations** apparaissent, ce qui limite leur application sur des graphes de connaissances réels en constante évolution.

Pour contourner ce problème, **INGRAM** s'affranchit de l'utilisation de modèles de langage coûteux ou de descriptions textuelles en se basant uniquement sur la structure topologique. Le modèle fonctionne en **quatre étapes mathématiques clés** :

— **Construction du graphe de relations (Relation Graph)**

INGRAM définit un graphe de relations, qui est un graphe pondéré où chaque nœud correspond à une relation, et le poids de chaque arête indique l'affinité entre deux relations. Cette affinité est calculée en fonction du nombre d'entités partagées et de la fréquence de ces partages (tête et queue) entre les relations.

— **Agrégation au niveau des relations (Relation-level Aggregation)**

Un réseau de neurones sur graphe (**GNN**) met à jour la représentation de chaque

relation en agrégeant les vecteurs de ses voisins dans le graphe de relations. Soit $z_i^{(l)}$ le vecteur de représentation caché de la relation r_i à la couche l. La mise à jour s'écrit :

$$z_i^{(l+1)} = \sigma\left(\sum_{r_j \in \mathcal{N}_i} \alpha_{ij}^{(l)} W^{(l)} z_j^{(l)}\right)$$

où σ est une fonction d'activation, $\mathcal{N}_i$ est l'ensemble des relations voisines de r_i dans le graphe de relations, $W^{(l)}$ est une matrice de poids apprenable, et $\alpha_{ij}^{(l)}$ est un coefficient d'attention.

— **Agrégation au niveau des entités (Entity-level Aggregation)**

Les embeddings d'entités (qu'elles soient connues ou nouvelles) sont générés dynamiquement. L'embedding de l'entité h_i est mis à jour en concaténant son propre état avec la représentation agrégée des relations de la couche finale ($z^{(L)}$), ainsi qu'avec l'état de ses voisins.

— **Modélisation des interactions (Scoring)**

Pour évaluer la validité (prédiction) d'un triplet final (v_i, r_k, v_j), INGRAM utilise une variante de la fonction de score DistMult. L'interaction entre l'embedding de l'entité et de la relation est calculée ainsi :

$$f(v_i, r_k, v_j) := h_i^\top \text{diag}(\bar{W} z_k) h_j$$

où $\bar{W}$ est une matrice de projection essentielle qui ajuste la dimension de l'embedding de la relation (z_k) à celle des entités (h). Le modèle est entraîné via une fonction de perte de classement basée sur la marge (margin-based ranking loss).

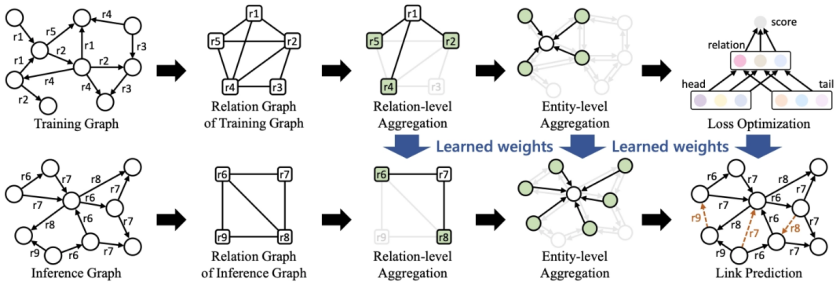


FIGURE 10 – Description de la méthode **INGRAM**

Tous ces articles étudiés présentent l'avantage d'utiliser des structures intermédiaires pour effectuer diverses opérations de raisonnement. De la détection d'anomalies à la prise en compte de nouveaux évènements pour le raisonnement, ces méthodes constituent un point central dans la suite de notre stage. Toutefois, si ces articles améliorent le raisonnement sur les graphes de connaissances, ils ne prennent pas en compte la dimension temporelle de ces graphes. Une étude bibliographique supplémentaire sur ce thème est donc nécessaire.

3.2 Le raisonnement sur les Temporal Knowledge Graph (TKG)

Dans un graphe de connaissances temporelles, il n'est plus question de triplets, mais de quadruplets (s, p, o, t), où s représente l'entité sujet, p représente le prédicat (ou la relation liant les entités, o représente l'entité objet, et t représente la date à laquelle ce fait est observé.

En réalité, un **TKG** peut être décomposé en un ensemble de KG, pris à des dates t différentes. C'est cette solution qui a été envisagé dans l'article "**Temporal knowledge graph reasoning using global and recent history information, C.Wang et al, Sci Rep (2026)**" [W+26] qui propose de prédire des évènements futurs en utilisant l'historique global et récent des données.

Il propose donc le modèle **GRHNet** qui se fonde sur 2 outils :

- Le **Global History Learner**, qui vise à apprendre la répétabilité des évènements en utilisant un modèle convolutionnel amélioré.
- Le **Recent History Learner**, qui étudie la variabilité du temps pour tous les évènements récents. Il s'appuie sur des outils statistiques telles que des fréquences.

On pose le cadre mathématique de cette méthode :

Soient $\mathcal{E}, \mathcal{R}, \mathcal{T}$ des ensembles finis d'entités, de relations et d'horodatages. **Un TKG $\mathcal{G}$ peut être vu comme une séquence de sous graphes historiques** (ex : $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_t\}$) où $\mathcal{G}_t$ = est le KG au temps t où $\mathcal{F}_t$ est l'ensemble des faits au temps t.

Chaque fait $\mathcal{F}_t$ peut s'écrire comme (s, p, o, t) où s, p, o désignent respectivement l'entité sujet, la relation, l'entité objet et t désigne l'horodatage.

Etant donné s et p, si le triplet (s, p, o) apparaît dans la période de temps $[t_0, t_k]$, la fréquence de o est de 1, et 0 sinon. On a donc la fonction suivante :

$$F_o^{(s,p)} = \begin{cases} 1 & \text{si } (s,p,o) \in \mathcal{G} \\ 0 & \text{sinon} \end{cases}$$

On peut voir aussi cette fonction comme un classifieur binaire pour simplifier.

Dans un premier temps, **GRHNet** classe tous les évènements à l'aide de ce classifieur.
En calculant la fréquence de chaque $o \in \mathcal{E}$ dans l'historique global, on obtient la table de fréquence globale de (s, p, t) :

$$FT_{global} = \{F_{oi}^{(s,p,t)}, i = 1, 2, \ldots, N, t = 1, 2, \ldots, t_n\}$$

où $N = \mid \mathcal{E} \mid$

En appliquant la même démarche pour l'historique récent, on obtient la table de fréquence récente pour (s, p, t). Celle-ci se calcule en regardant l'historique des K sous graphes adjacents :

$$FT_{recent} = \{F_{oi}^{(s,p,t)}, i = 1, 2, \ldots, N, t = t_{n-k}, \ldots, t_n\}$$

Ensuite, ces tables de fréquence sont utilisées pour alimenter le Global et le Recent History Learner.

Pour déterminer le score final, **GRHNet** combine la probabilité de prédiction des entités apprises par le Global History Learner et le Recent History Learner.

Cet article présente également d'autres intérêts :

D'abord, à la manière du modèle **LogNet**, qui étudiait l'embedding local et global des triplets pour la détection d'anomalies, on constate avec le modèle GRHNet la distinction entre l'historique récent et global pour la prédiction d'évènements.

Ensuite, il est intéressant de remarquer comment les outils statistiques, tels que les fréquences par exemple permettent de simplifier la tâche des modèles de Deep Learning en permettant d'avoir une vision plus claire des graphes complexes ayant plusieurs temporalités.

Il existe également des articles qui améliorent la collaboration LLM + KG mais sur des graphes de connaissances temporelles. On pourrait parler alors de raisonnement LLM + TKG.
Dans cette optique l'article "**Large Language Models-guided Dynamic Adaptation for Temporal Knowledge Graph Reasoning, Wang et al, NeurIPS (2024)**" [WSL+24] introduit une méthode novatrice (**LLM-DA**) permettant de raisonner sur ces graphes temporels (**TKG**) en utilisant les **LLM** comme des analystes de tendances historiques.

L'objectif de cet article est de combler les défauts des **LLM**, à savoir leur fonctionnement en boîte noire et la difficulté pour les mettre à jour.

Comme expliqué précédemment, Dans un **TKG** (**T**emporal **K**nowledge **G**raph), un fait est représenté par un quadruplet (e_s, r, e_o, t), où t est un horodatage (timestamp). La tâche de raisonnement temporel consiste souvent à prédire une entité future, par exemple $(e_s, r, ?, t_{futur})$.

Cette méthode exploite les capacités des **LLM** pour analyser les données et extraire des règles logiques temporelles, en dévoilant des motifs temporels et facilitant un raisonnement interprétable. Ensuite, cette méthode propose une adaptation dynamique (**D**ynamic **A**daptation), qui met à jour itérativement les règles générées par les **LLM** avec les évènements plus récents.

Les **TKGR** (**T**emporal **K**nowledge **G**raph **R**easoning) se présentent sous **deux grandes familles** :

- **Les TKGR basés sur les règles** : ils améliorent l'inférence des TKG en s'appuyant sur des règles logiques temporelles pour prédire les évènements futurs.
- **Les TKGR utilisant des méthodes de deep learning**, qui permettent de capturer des motifs temporels cachés afin de prédire les évènements futurs dans les TKG.

Chacune de ces familles possèdent ces limites : pour les méthodes basées sur les règles, l'extraction de ces dernières sur les **TKG** reste un défi majeur et peut parfois être impossible

suivant la forme et la complexité du graphe. En ce qui concerne les méthodes basées sur le deep learning, celles-ci souffrent d'un manque d'interprétabilité, ce qui rend difficile la vérification des prédictions.

La méthode **LLM-DA** propose pour remédier à cela un **TKGR** basé sur les règles, sur lesquelles le **LLM** va pouvoir s'appuyer pour réaliser ses prédictions. Ainsi, en se basant sur des règles explicites, **LLM-DA** comble une limite majeure des **TKGR** basés sur le deep learning qui concernait le manque d'interprétabilité.

La méthode **LLM-DA** fonctionne en 4 étapes :

— **Echantillonnage de règles temporelles** : Exploration des marches aléatoires markoviennes pour extraire des règles logiques temporelles à partir des données historiques. Une **marche aléatoire markovienne** est un **processus qui décrit une suite de différents "sauts" entre différents "états"**. Elle respecte la **propriété de Markov** qui stipule que la probabilité de passer à l'état suivant ne dépend que de l'état actuel. Autrement dit, elle respecte l'**absence de mémoire**.
Afin de permettre le bon raisonnement des LLM pour générer les règles, on pose deux contraintes sur les marches aléatoires :

  — L'ordre temporel : on ne remonte pas le temps.
  — Les intervalles temporels : On préfère des intervalles courts (en donnant des poids + importants).

  LLM-DA doit s'assurer que les marches aléatoires respectent la propriété de Markov et ces contraintes.

— **Génération de règles** : Utilisation des LLM pour extraire des motifs temporelles significatifs et des dépendances temporelles complexes à partir des données historiques.
Le LLM utilise un sélecteur de relations naturelles pour identifier les K relations les plus pertinentes. Pour une règle donnée, Sentence-Bert teste la tête de la règle pour sortir des vecteurs de sortie à partir desquels le score sera calculé. Un tri par ordre décroissant permet de sélectionner les K-meilleures.

— **Adaptation dynamique** : La méthode exploite les LLM pour mettre à jour les règles générales générées par le LLM en utilisant les données actuelles.

  En raison de la nature évolutive des **TKG**, les règles sélectionnées par les **LLM** deviennent moins adaptées aux nouvelles données. Les règles de bonne qualité vont ainsi devenir des règles de basse qualité.

  Pour cela LLM-DA extrait les règles temporelles à partir des données actuelles et utilise ces règles pour mettre à jour les règles devenues de basse qualité.

  Ces règles extraites des données actuelles s'obtiennent à partir de l'exploration des marches aléatoires markoviennes (évoquées dans le 1) en regardant les données actuelles en plus des données historiques. C'est donc un **processus itératif** qui permet

de mettre à jour les règles du TKGR.

— **Raisonnement sur les candidats** : Combine le raisonnement sur les règles et celui sur les graphes pour générer des candidats.
Pour un candidat $e_{o'}$, le score final permettant de trier les règles les plus pertinentes s'obtient ainsi en calculant :

$$Score_{final} = \alpha \cdot Score(\rho, e_{o'} + (1 - \alpha) \cdot Score(graph, e_{o'})$$

où $Score < graph, eo' >=< f_g(Query), e_{o'} >$
avec $f_g(Query)$ une fonction de raisonnement basée sur les graphes.

En conclusion, **LLM-DA** exploite un sélecteur de relations contextuelles pour identifier les K relations les plus pertinentes. Ensuite, il utilise les capacités génératives des LLM pour analyser les données historiques et dériver des règles plus générales. Il repose enfin une stratégie d'adaptation dynamique pour mettre à jour les règles générées par les LLM avec les évènements les plus récents.

De cette importante bibliographie, des concepts importants se sont dégagés pour la suite de notre projet. Dans l'objectif de détecter des signaux faibles sur les graphes de connaissances temporels, nous avons d'abord orienté nos recherches sur la détection d'anomalies sur des graphes de connaissances. Ces premiers travaux seront présentés dans la prochaine section.

4 Premiers travaux sur la détection d'anomalies dans les graphes de connaissances

En étudiant le modèle **LoGNet** [Pir22] (cf p.15), la première mission post-bibliographie a consisté à tester ce modèle pour observer la cohérence de celui-ci dans la détection d'anomalies. Ce modèle a été choisi comme base pour notre projet car son objectif premier correspondait à ce que nous cherchions. En effet, la détection de signaux faibles passe d'abord par la détection d'anomalies.

Il est important également de comprendre ce qu'est une anomalie, car celle-ci peut prendre plusieurs sens :

Une anomalie peut être un évènement faux, mais elle peut aussi être un évènement vrai mais rare. Bien que la dernière définition semble plus aboutie et fut finalement retenue pour notre projet final, nous travaillerons avec ces deux définitions lors de ce stage.

Dans les sous-sections suivantes, nous retracerons les différentes étapes concernant les premiers travaux sur le modèle **LoGNet**. De la compréhension du code avec une nouvelle base de données à la modification de celui-ci pour améliorer la détection d'anomalies, en passant par les différents problèmes rencontrés lors de cette étude.

Cette seconde partie de stage a été centrale car les limites rencontrées lors de l'étude de **LoGNet** a permis d'envisager de nouvelles solutions.

4.1 Etude du modèle LoGNet

4.1.1 Récupération du code et premières constatations

Le code permettant de tester le modèle LoGNet est joint dans l'article qui présente la méthode nous avons pu ainsi récupérer celui-ci. L'ensemble de la méthode se présente sous la forme d'un dossier GitHub où sont joints les codes permettant de récupérer la base de données, les codes construisant le modèle et le programme permettant d'entraîner ce dernier. Cependant, deux limites majeures ont freiné notre étude de LoGNet.

— Tout d'abord, **beaucoup de fonctions du code original faisaient appel à une bibliothèque, "DGL" qui comprend des éléments en C++, qui ne peuvent pas être exécutées sur un éditeur de code Python classique**. Puisqu'une des conditions au bon déroulement de ce projet était que chacun des participants travaille avec le même langage de programmation, **Python**, une réécriture de la méthode a été nécessaire pour poursuivre nos travaux.

— Ensuite, **Le code joint pour récupérer la base de données ne nous a pas permis de récupérer celle-ci**. Il a donc été décidé de travailler sur une nouvelle base de données, que nous présenterons plus tard.

Ces premières constatations nous montrent que l'étude d'un modèle peut s'étendre sur une période de temps plus ou moins longue puisque cela dépend de la qualité des éléments dont nous disposons. Dans le cas où tous les éléments permettant de comprendre et d'exécuter la méthode ne sont pas réunis, il faut donc trouver des alternatives pour essayer de reproduire le plus fidèlement possible la méthode étudiée.

4.1.2 Présentation des données

La sélection de jeux de données adéquats fut une tache délicate lors de notre étude. Cette méthode reposant sur un apprentissage supervisé, il nous fallait donc un jeu de données assez important pour permettre à notre méthode de comprendre les différentes règles. Dans un premier temps, les **Knowledge Graph** (**KG**) étaient la cible de nos recherches puisque ce genre de graphes avait été initialement utilisée dans la méthode originelle.

Cependant, dans un objectif de faire évoluer cette méthode dans un cadre temporel, un **Temporal Knowledge Graph (TKG)**, incluant des périodes de temps associées aux différents évènements a finalement été choisi. C'est dans ce cadre que les travaux réalisés précédemment par **Yassir Lairgi** se sont avérés essentiels, car nous avons pu choisir un **TKG** directement construit grâce à sa méthode [LMB+24], nous permettant une meilleure vision de la structure du graphe et des données qui le composent.

Dans un premier temps, afin de combler la non prise en compte de la temporalité par notre méthode, nous choisissons de décomposer notre graphe de connaissances en 11 sous graphes (**snapshots**), qui retracent chacun l'ensemble des faits s'étant produits à une date t. Cette approche est assez semblable à ce qui est présenté dans la méthode GRHNet [W+26]. Dans chacun de ces sous graphes, nous ne prêtons attention qu'à la nature des faits et ne prenons pas en compte la période à laquelle ces faits se sont produits.

On donne ci-dessous un tableau récapitulatif des différents snapshots :

TABLE 1 – Tableau récapitulatif du nombre de faits dans chaque snapshot

Snapshot	**Nombre de triplets**
Snapshot 0	318
Snapshot 1	247
Snapshot 2	262
Snapshot 3	51
Snapshot 4	242
Snapshot 5	177
Snapshot 6	< 20
Snapshot 7	51
Snapshot 8	204
Snapshot 9	< 20
Snapshot 10	< 20

Ici nous observons une **certaine disparité dans le nombre de faits** (ici représentés par des **triplets**) qui composent chaque sous graphe. Cette disparité n'est pas optimale et c'est d'ailleurs celle-ci qui nous poussera à chercher d'autres jeux de données sur lesquels entraîner nos nouveaux modèles. Cependant, dans un objectif de compréhension du code, les snapshots n° 0, 1 et 2 comportent suffisamment de données pour nous permettre une analyse objective des premiers éléments de code fournis. Les snapshots comportant moins de 20 faits n'ont pas été retenus pour notre analyse.

La nature de nos données ne nous permet pas de réaliser des analyses statistiques plus poussées sur celles-ci. En effet, il ne s'agit pas de données numériques mais de données textuelles retraçant différents faits portant sur les récentes publications scientifiques des membres du **LIRIS**.

De plus, la nature intrinsèque des graphes de connaissances implique l'existence de relations de dépendances entre les données, ce qui complexifie l'analyse de celles-ci.

Dans les sous-sections suivantes, nous retracerons l'ensemble des modifications réalisées sur le code de **LoGNet** qui nous ont permis d'exécuter celui-ci sur notre base de données.

4.1.3 Premières modifications du code originel

Adaptation du graphe pour la bibliothèque PyTorch

La première modification vise à adapter la base de données pour que les différents outils de la bibliothèque **PyTorch** puisse être opérationnels sur le graphe de connaissances issu de notre base.

Une première fonction allant dans ce sens est donc créée. Son objectif est donc de récupérer l'ensemble des entités et des relations du graphe, puis de leur attribuer des identifiant (ID). On créé ensuite un **edge_index**, qui correspond à la matrice d'adjacence de notre graphe, représentant les relations entre les différentes entités.

Afin d'économiser de la mémoire et de réduire le coût de calcul par la suite, cette matrice possède N lignes et 2 colonnes. Sur la première colonne sont représentées les entités sujets, et sur la seconde colonne les entités objets avec lesquelles elles sont reliées. Enfin un **edge_attribute** est créé représentant les différents types de relations. Le graphe est ainsi prêt pour **PyTorch**.

```python
def convert_itext2kg_to_pyg(triples):
    # 1. Calculs des mappings
    entities = sorted(list(set([s for s, p, o in triples]
                + [o for s, p, o in triples])))
    entity_to_id = {entity: i for i, entity in enumerate(entities)}

    relations = sorted(list(set([p for s, p, o in triples])))
    rel_to_id = {rel: i for i, rel in enumerate(relations)}

    # 2. Creation des tenseurs
    sources = [entity_to_id[s] for s, p, o in triples]
    targets = [entity_to_id[o] for s, p, o in triples]
    edge_types = [rel_to_id[p] for s, p, o in triples]

    edge_index = th.tensor([sources, targets], dtype=th.long)
    edge_attr = th.tensor(edge_types, dtype=th.long)

    num_nodes = len(entities)
    x = th.randn((num_nodes, 16))

    # 3. Creation du data
    data = Data(x=x, edge_index=edge_index, edge_attr=edge_attr)

    # 4. Attacher les dictionnaires d'entite et de relation
    data.entity_to_id = entity_to_id
    data.predicate_to_id = rel_to_id

    return data
```

Listing 1 – Fonction de conversion d'un graphe textuel vers PyTorch Geometric (PyG)

Extraction des triplets

L'analyse du graphe passe ensuite par l'extraction des faits sous forme de triplets. Cette étape est fondamentale dans la mesure où la méthode ne nous permet pas de raisonner directement sur le graphe. Dans le cas du snapshot 0 par exemple, on obtient ainsi un tableau **Numpy** de dimension 318 (le nombre de triplets) $\times$ 3 (les colonnes des triplets : **sujet, prédicat, objet**).

Construction du triple Line Graph

Il s'agit ici de la plus importante modification du code original. En effet, dans le code initial, la bibliothèque **DGL** prévoyait l'utilisation d'une fonction construisant directement un Line Graph à partir d'un graphe hétérogène.

Dans la bibliothèque **PyTorch**, il existe une fonction qui transforme un graphe en Line Graph, mais celle-ci ne prend en entrée qu'un graphe homogène, c'est à dire un graphe n'ayant qu'un seul type de relations. Cette fonction n'est donc pas adapté à des graphes de connaissances comme celui dont nous disposons. **Il a donc été décidé de reconstruire manuellement le triple Line Graph à partir du graphe de connaissances originel**.

Puisque les triplets sont déjà extraits, on définit une fonction qui prend en entrée ces triplets et qui va construire une matrice d'adjacence pour ces triplets. On donne la formule mathématique associée :

Définition : Soit $\mathcal{G}$ un graphe de connaissances.
Pour deux triplets $e_i = (s_i, p_i, o_i) \in \mathcal{G}$ et $e_j = (s_j, p_j, o_j) \in \mathcal{G}$, les éléments de la matrice d'adjacence $\mathcal{M}$ du **Line Graph** sont définis par :

$$\mathcal{M}_{ij} = \begin{cases} 1 & \text{si } (s_i = s_j \text{ ou } o_i = o_j \text{ ou } s_i = o_j \text{ ou } o_i = s_j) \text{ et } i \neq j \\ 0 & \text{sinon} \end{cases}$$

Il est maintenant aisé de construire une fonction qui construit un graphe compatible PyTorch. Elle convertit la matrice d'adjacence en tenseurs PyTorch avant de les utiliser pour construire l'**edge_index** de ce nouveau graphe. A présent, nous avons un triple Line Graph sur lequel nous pourrons raisonner ensuite.

Génération de faux triplets

Parce que le code initial ne nous permettait pas de comprendre comment et dans quel proportion les faits négatifs étaient construits, nous avons repris les éléments de l'article présentant la méthode pour construire ces triplets. **L'objectif de notre programme sera de détecter ces triplets, qui sont donc les anomalies de notre graphe**.

Dans un premier temps, et dans un objectif de rester fidèle à la description de l'article, nous choisissons de générer des faux triplets en permutant les entités entre elles. On écrit ainsi la fonction de code suivante :

```python
def generer_faux_triplets(
    triples: np.ndarray, all_entities: np.ndarray, all_relations: np.
        ndarray, ratio: float
) -> np.ndarray:
    num_triples = len(triples)
    num_corrupted = max(1, int(num_triples * ratio))
    indices = np.random.choice(num_triples, num_corrupted, replace=False)
    corrupted_triples = triples[indices].copy()

    for i in range(num_corrupted):
        col_to_corrupt = np.random.choice([0, 1, 2])
        if col_to_corrupt in (0, 2):
            corrupted_triples[i, col_to_corrupt] = np.random.choice(all_
                entities)
        else:
            corrupted_triples[i, 1] = np.random.choice(all_relations)

    return corrupted_triples
```

Listing 2 – Fonction de génération des faux triplets

Autrement dit : on choisit un nombre de triplets à corrompre qui est égal à un pourcentage du nombre de triplets du graphe (le ratio). Pour chacun de ces triplets choisis au hasard, nous décidons de sélectionner au hasard soit le sujet, soit la relation, soit l'objet. Puis, en fonction de la partie du triplet choisie, celle-ci sera remplacée par un autre objet similaire contenu dans les autres triplets du graphe.

Par exemple, si pour un triplet à corrompre, le sujet est choisi, celui-ci sera remplacé par un autre sujet contenu dans les autres triplets.

La conséquence immédiate est que les triplets générés sont bien souvent faux sémantiquement, avec des phrases qui n'ont aucun sens. Nous sommes donc dans la première définition de l'anomalie qui correspond à la génération de faux triplets (voir 4 : premiers travaux dans la détection d'anomalies dans les graphes de connaissances, p.23). On donne un exemple de faux triplets générés :

sujet	prédicat	objet
digital humanities	part_of	géodisco project
itineraries described in textual documents	combines	nlp approaches
6th acm sigspatial international workshop on geospatial humanities	held_in	toponym disambiguation in historical documents
article addressing gaps in nlp research on historical french and complex textual structures	addresses_gap_in	paper on using network analysis to identify qualitative neighbors for toponyms in eighteenth century french encyclopedias
ludovic moncla	is_author_of	labex imu
ludovic moncla	has_objective	j. nogueras iso
ludovic moncla	was_supervised_by	cartographier les odonymes de paris cités dans les romans du xixème siècle.
ludovic moncla	located_in	article co authored with katherine mcdonough and matje van de camp
ludovic moncla	affiliated_with	humanities research
ludovic moncla	has_end_date	computer science
soundcityve	identifies	city of lyon
department	part_of	geocoding for texts with fine grain toponyms: an experiment on a geoparsed hiking descriptions corpus
ludovic moncla	is_involved_with	sound landscape
ludovic moncla	co_authored	online
ludovic moncla	addresses_gap_in	workshop corpus in giscience

FIGURE 11 – Exemple de faux triplets pour le snapshot 0 avec un ratio de 0.1

L'étape d'ajout de faux triplets dans un graphe de connaissances s'appelle **la contamination**.

Modification de la fonction de perte

Dans le code initial, la fonction de perte utilisée était la Margin Ranking Loss. L'objectif de cette fonction est de calculer la perte en comparant pour chaque itération deux triplets, x_1 et x_2, par rapport à un label fixé y. La formule est la suivante :

$$Loss(x_1, x_2, y) = max(0; -y(x_1 - x_2) + marge)$$

Cette fonction nécessite que les deux triplets en entrée, x_1 et x_2 soient de nature opposée, c'est-à-dire, un triplet vrai et un triplet faux. Dès lors, une première contradiction apparaît : **dans le cadre de notre projet, la détection d'anomalies au sein des graphes de connaissances ne doit permettre d'identifier qu'un petit nombre d'anomalies dans un grand jeu de données.** Or la condition imposée par la fonction de perte ci-dessus exige qu'il y ait autant d'anomalies que de triplets vrais au sein de notre graphe de connaissances. Autrement dit, un ratio de 1 serait appliqué au moment de la création des faux triplets.

Il convient donc de changer cette fonction de perte pour en choisir une qui soit plus adaptée à notre objectif.

Parmi les différentes pistes explorées, nous choisissons d'utiliser la **BCE with Logits Loss function.** Cette fonction reprend le cadre de la fonction **BCE** (**B**inary **C**ross **E**ntropy) tout en corrigeant un défaut majeur de celle-ci quant aux données fournies en entrée. En effet, alors que la fonction BCE prenait en entrée des valeurs comprises entre 0 et 1, ce qui nécessitait au préalable l'application de la fonction sigmoïde aux différents scores calculés en amont pour chaque triplet, La nouvelle fonction prend en entrée des Logits, c'est à dire des scores positifs ou négatifs, pour lesquels la fonction sigmoïde sera appliquée directement au sein même de la fonction. Cette fonction est donc bien plus adaptée à notre cadre où la plausibilité des triplets est désignée par un score, qui peut être positif ou négatif.

La fonction **BCE with Logits Loss function** prend la forme suivante :

$$l_n = -w_n \left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n)) \right]$$

où w_n est un poids recalculé à chaque itération, y_n est le label du triplet n,
x_n le score de plausibilité du triplet n,
et $\sigma(x_n)$ désigne la fonction sigmoïde ($f(x) = \frac{1}{1+e^{-x}}$) appliquée à x_n.

4.1.4 Entraînement du modèle

Nous choisissons dans un objectif de reproduction de la méthode de l'article de créer des "masques" d'entraînement et de test pour évaluer les performances de notre modèle modifié.

Un masque est une manière de décomposer un jeu de données en deux parties en construisant des matrices qui représentent les triplets contenus dans chaque ensemble.

Pour illustrer cette stratification, les masques d'entraînement (M_{train}) et de test (M_{test}) peuvent être représentés sous la forme de vecteurs colonnes booléens de dimension $N \times 1$ (avec N le nombre de triplets), structurés en deux blocs distincts :

$$M_{\text{train}} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ \vdots \\ \hline 0 \\ 1 \\ 1 \\ \vdots \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} 1 \\ 1 \\ 0 \\ \vdots \end{matrix}}\right\} \text{Triplets positifs } (80\% \text{ de } 1) \\ \left.\vphantom{\begin{matrix} 0 \\ 1 \\ 1 \\ \vdots \end{matrix}}\right\} \text{Triplets négatifs } (80\% \text{ de } 1) \end{matrix} \qquad M_{\text{test}} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ \hline 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} 0 \\ 0 \\ 1 \\ \vdots \end{matrix}}\right\} \text{Triplets positifs } (20\% \text{ de } 1) \\ \left.\vphantom{\begin{matrix} 1 \\ 0 \\ 0 \\ \vdots \end{matrix}}\right\} \text{Triplets négatifs } (20\% \text{ de } 1) \end{matrix}$$

Par construction, ces deux masques sont complémentaires et mutuellement exclusifs : $M_{\text{train}} + M_{\text{test}} = \mathbf{1}$ (où $\mathbf{1}$ est le vecteur unitaire), garantissant qu'aucun triplet n'est utilisé à la fois pour l'apprentissage et pour l'évaluation.

Autrement dit, nous construisons deux matrices de zéros de taille $N \times 1$ où N est le nombre de triplets contenus dans le snapshot. Ces matrices correspondent respectivement au masque d'entraînement et de test. Pour la partie "entraînement" (**train mask**), on attribue dans la première matrice ensuite la valeur 1 aux indices des triplets qui constituent l'ensemble d'entraînement. La même démarche est appliquée pour construire l'ensemble de test.

Le découpage du graphe s'organise donc autour d'un masque d'entraînement qui comprend 80 % des triplets du graphe (contamination incluse) et un masque de test (**test mask**) qui comprend les 20 % restants.

Dans le cadre de la contamination du graphe, un ratio de 10 % d'anomalies est appliqué pour chacun des snapshots étudiés, ce qui représente pour le snapshot 0 une trentaine d'anomalies. Au vu du faible nombre d'anomalies générées, une stratification est appliquée pour que la proportion d'anomalies dans le train mask et le test mask soit similaire.

4.1.5 Outils de mesure utilisés

Afin de capter l'efficacité du modèle, nous choisissons d'évaluer celui-ci en traçant les **courbes ROC/AUC**.

La courbe **ROC** (de l'anglais **R**eceiver **O**perating **C**haracteristic, pour « caractéristique de fonctionnement du récepteur ») est une mesure de la performance d'un classificateur binaire.

Graphiquement, on représente souvent la mesure **ROC** sous la forme d'une courbe qui donne le taux de vrais positifs (fraction des réellement positifs qui sont effectivement détectés comme positifs) en fonction du taux de faux positifs (fraction des triplets réellement négatifs qui sont, à tort, détectés comme positifs). Elle est donc souvent associée à la mesure **AUC** (de l'anglais Area Under Curve) qui représente la probabilité que le modèle, s'il reçoit un exemple positif et un exemple négatif (choisis aléatoirement), classe l'exemple positif plus haut que l'exemple négatif.

Puis, pour comparer les scores entre les anomalies et les vrais triplets, on représente les boxplots des scores de plausibilité globale, locale, et finale afin de comprendre la contribution de chaque outil à la détection d'anomalies.

Des fichiers récapitulant les faux triplets générés pour chaque snapshot, ainsi que les anomalies détectées par le modèle sont créés sous la forme de fichiers CSV. Ils nous permettent d'avoir une meilleure visibilité quant au type d'anomalies détectées par le modèle.

4.1.6 Résultats et premières constatations

Les premiers résultats obtenus furent décevants. Les scores obtenus par le calcul de l'AUC montrent une difficulté du modèle à générer des règles efficaces pour détecter des anomalies sur les triplets de test. Le tableau ci-dessous présente un récapitulatif des scores obtenus pour chaque snapshot évalué. Les snapshots comportant moins de 20 triplets n'ont pas été évalués.

TABLE 2 – Tableau récapitulatif de l'AUC obtenue dans chaque snapshot

Snapshot	Nombre de triplets	AUC entraînement	AUC test
Snapshot 0	318	0.9461	0.4245
Snapshot 1	247	0.7697	0.4320
Snapshot 2	262	0.7146	0.2717
Snapshot 3	51	0.9906	0.4204
Snapshot 4	242	0.7058	0.4204
Snapshot 5	177	1.00	0.7593
Snapshot 6	< 20	–	–
Snapshot 7	51	1.00	1.00
Snapshot 8	204	0.8627	0.4512
Snapshot 9	< 20	–	–
Snapshot 10	< 20	–	–

Comme nous pouvons le voir, les snapshots qui comportent le plus de triplets présentent une AUC inférieure à 0.5, ce qui signifie actuellement que le modèle ne fait pas mieux que le hasard pour prédire si un triplet est normal ou non.

Lorsque nous nous concentrons sur le snapshot 0, qui est notre ensemble de référence dans ce jeu de données, on observe que les scores de plausibilité entre les triplets normaux et les anomalies sont confondus, ce qui confirme les difficultés du modèle à effectuer une sélection claire.

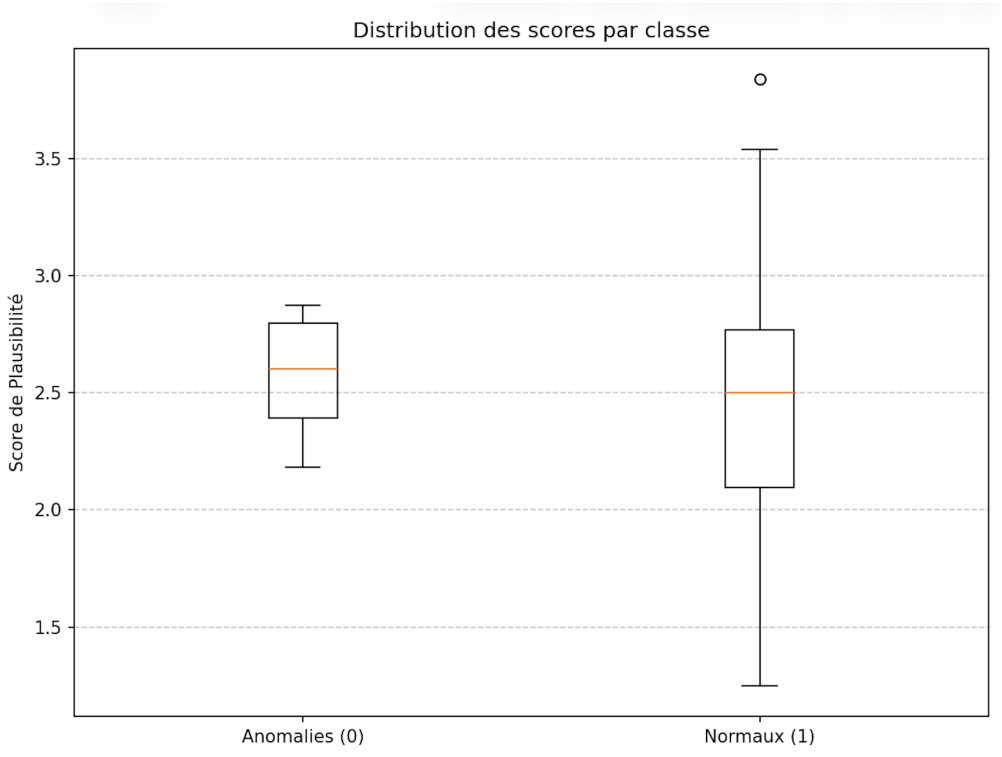


FIGURE 12 – Boxplot des scores finaux pour l'ensemble de test en fonction du type de triplet pour la snapshot 0

Les boxplots précédents nous montrent l'absence de séparation claire dans les scores finaux entre les triplets anomalies et les triplets normaux.

D'autre part, on constate une certaine instabilité dans les scores observés entre différents entraînements. Malgré le réglage des hyperparamètres de notre modèle, comme par exemple taux d'apprentissage, ou encore le poids de pénalisation de l'erreur, les résultats restent médiocres.

Cette phase de modification du code a été suivie d'une longue période d'analyse pour tenter de comprendre quelles sont les causes de l'inefficacité de notre modèle.

4.2 Analyse des premiers résultats et remise en cause de la méthode LoGNet

Au regard des mauvais résultats obtenus par notre modèle, nous avons tout d'abord cherché à comprendre la cause première de l'inefficacité de celui-ci. Nous avons identifié 4 points à analyser, sur lesquels nous reviendrons dans les prochaines sous sections. Ces points sont :

— La taille du jeu de données
— La construction du Line Graph
— La qualité des embbedings
— La génération des faux triplets

4.2.1 Les défauts présentés par le jeu de données initial

Si le jeu de données initial présentait l'avantage d'être compréhensible et construit par nos propres méthodes, il ne comportait malheureusement que trop peu de données, ce qui devient très vite un inconvénient, notamment dans les méthodes d'apprentissage supervisé comme celles que nous étudions.

En réalité, il est difficile de trouver le bon jeu de données qui soit adapté à notre expérience. Les bases de données s'avèrent souvent complexes et manquent parfois d'informations. Notamment, beaucoup de graphes de connaissances sont écrits avec des nombres et les triplets sont donc des chaînes d'indices (par exemple, un triplet peut s'écrire [14,11,2]). Sans dictionnaire pour interpréter ces indices, ces graphes deviennent peu lisibles. De plus, construire un dictionnaire sur un des graphes comprenant plusieurs milliers d'entités s'avère coûteux en temps et en calcul, car on souhaite in fine que les triplets soient cohérents. Par ailleurs, les travaux précédemment effectués par notre équipe, notamment sur la méthode **iText2KG**, prévoyaient la construction de graphes de connaissances à partir de corpus textuels. Ainsi, nos nouvelles méthodes doivent permettre de détecter ces anomalies textuelles dans les graphes. L'utilité des graphes d'indices ne comportant pas de dictionnaire est donc remise en cause.

4.2.2 La construction du Line Graph

Comme expliqué précédemment, la conversion du code original vers la bibliothèque PyTorch nous a poussé à réécrire certaines fonctions utilisées initialement et qui n'existaient pas dans

la nouvelle bibliothèque. Parmi ces fonctions, une des plus importantes de la méthode originale, qui concerne la construction du Line Graph.

Nous allons maintenant rappeler son objectif et la méthode de construction.
Un Line Graph est une structure simplifiée qui permet de convertir des graphes hétérogènes (possédant des diversités de relations) en des graphes homogènes (qui ne possèdent qu'un type de relations). Le Line Graph est donc un graphe dont les nœuds sont constitués des triplets de l'ancien graphe. Les arêtes sont ensuite tracées entre les nœuds qui partagent des entités communes.

Cependant, le vrai défi dans la construction de ces nouvelles structures réside dans la capacité à reconstituer les informations du graphe original dans ce nouveau graphe. Ces informations sont les embeddings des sujets, relations, objets, qui composent les triplets du graphe. S'il existe des fonctions qui permettent de construire le Line Graph dans les bibliothèques utilisées (PyTorch), l'inconvénient est que celles-ci ne prennent en entrée uniquement des graphes homogènes. Il faut donc au préalable convertir notre graphe hétérogène en graphe homogène avant même de construire le Line Graph, d'où la perte d'informations.

4.2.3 La qualité des embeddings

Conséquence du point précédent, la construction d'une structure intermédiaire (le Line Graph) délègue l'entièreté de la sémantique aux attributs des nœuds. L'efficacité du raisonnement global dépend donc intrinsèquement de la qualité de l'embedding initial des triplets. Or, la méthode d'encodage originale de LoGNet présente des limites architecturales.

Dans cette méthode, l'encodage local du triplet est réalisé par un réseau de neurones récurrent (un **BiGRU**). Historiquement, **les réseaux récurrents (RNN) sont conçus pour modéliser des dépendances à long terme sur des données séquentielles fluides (séries temporelles, phrases longues).** L'application d'un RNN sur un triplet de connaissances s'avère sous-optimale pour deux raisons :

— D'une part, la séquence traitée est extrêmement courte (strictement trois éléments : Sujet, Relation, Objet), ce qui ne permet pas d'exploiter la mémoire interne du RNN.
— D'autre part, un triplet ne relève pas d'une logique purement séquentielle, mais plutôt d'une cohérence sémantique globale : la relation lie le sujet et l'objet simultanément. Par conséquent, l'encodage séquentiel peine à capturer toute la subtilité des interactions sémantiques.

Ces défauts d'architecture, combinés à des pertes d'informations dans la construction du Line Graph, peuvent expliquer les défauts de détection d'anomalies de notre modèle.

4.2.4 La génération des faux triplets

La génération des faux triplets (voir 4.1.3) peut être remise en question dans le cadre de notre projet. Dans l'objectif de détecter les signaux faibles, nous devons construire un modèle capable de détecter des évènements anormaux. Cependant, le terme "anormal" ne veut pas forcément dire "faux". Ce terme peut également être associé à des évènements rares, qui

peuvent être sémantiquement corrects.

C'est à ce moment-là que notre projet va prendre une tout autre direction : **il ne s'agit plus de détecter des triplets faux générés en grand nombre dans un graphe de connaissances mais d'être capable de mettre en évidence l'évènement "rare", "anormal", présent en très petit nombre dans un grand jeu de données.**

4.3 De la remise en cause de la méthode initiale à la conception d'un modèle nouveau

Cette seconde partie de stage portant sur l'adaptation et la modification d'une méthode pour répondre à nos objectifs a mis en évidence des failles claires qui nous ont amenées à réfléchir à de nouvelles solutions.

Cette réflexion menée collectivement a abouti à la conception d'un nouveau modèle. Celui-ci a été développé à la suite de modifications successives effectuées sur notre premier modèle et ne fait pas l'objet d'un plan défini en amont. La première d'entre elles concernait la méthode d'embedding des triplets.

Bien que l'approche combinant une étude globale et locale ait été conservée dans notre méthode, la prise en compte de la sémantique des mots dans les embeddings des triplets nous a amené à supprimer le Line Graph, élément principal de la méthode originale.

Dans la prochaine section, nous allons présenter le modèle **BERT/R-GCN,** capable de détecter des anomalies complexes présentes en petite quantité dans un graphe de connaissances temporelles.

5 Présentation du modèle BERT/R-GCN

5.1 Prise en compte de la sémantique des mots avec Sentence-BERT

Le premier problème concernait l'embedding des triplets qui ne prenait pas en compte la sémantique des mots. Afin de calculer les scores locaux, **il a donc été décidé de remplacer le modèle GRU par Sentence-BERT pour calculer les embeddings des triplets.**

Sentence-BERT est un modèle de langage basé sur l'architecture Transformer. Contrairement aux approches statiques, il génère des représentations vectorielles (*embeddings*) contextualisées, où la valeur mathématique de chaque mot est dynamiquement ajustée en fonction de l'intégralité de la séquence dans laquelle il s'inscrit.

On présente l'utilisation de ce modèle dans notre méthode.

Soit $\mathcal{G} = \{\mathcal{E}, \mathcal{R}\}$ un graphe de connaissances, avec $\mathcal{E}$ l'ensemble des entités et $\mathcal{R}$ l'ensemble des relations.

On extrait tout d'abord l'ensemble des triplets $[s, p, o]$ constituant le graphe.

Puis, dans ces triplets seront traités séparément les sujets, les prédicats et les objets. Pour chacune de ces catégories, on réalisera l'embedding des éléments qui les composent grâce à **Sentence-BERT**.

Remarque : Il serait envisageable de traiter la phrase complète afin de capturer les nuances contextuelles de chaque terme. Toutefois, **BERT** dispose de bases sémantiques solides permettant d'isoler le sens intrinsèque des mots.

L'évaluation sémantique du triplet complet sera prise en charge par la suite via un réseau de neurones spécifique, avant d'y intégrer la structure globale du graphe. Cette logique se traduit par le code ci-dessous :

```
subjects = kg_triples[:, 0]
predicates = kg_triples[:, 1]
objects = kg_triples[:, 2]

eS_base = self.entities_emb(subjects)
eR_base = self.relations_emb(predicates)
eO_base = self.entities_emb(objects)
```

Listing 3 – Forward : Extraction des embeddings de base

Remarque : on ne réalise pas l'embedding dans le forward. Puisque celui-ci est fixe, on le réalise en amont afin d'économiser du temps de calcul et d'améliorer le raisonnement du modèle.

Par la suite, on calcule les différences entre les embeddings des sujets et des objets ainsi que les produits de ces mêmes embeddings. Ces éléments vont permettre d'ajouter de l'information pour les calculs des scores qui vont suivre. Enfin, on concatène les embeddings pour avoir un vecteur unique pour chaque triplet.

```
diff_so = eS_base - eO_base
prod_so = eS_base * eO_base

x_local = th.cat([eS_base, eR_base, eO_base, diff_so,prod_so], dim=-1)
```

Listing 4 – Forward : Concaténation des embeddings et calcul de différences et de produits

5.2 Calcul du score de plausibilité locale

C'est ici que la non prise en compte des phrases contenues dans les triplets va être corrigée. Dans **LoGNet**, le score local était calculé par **TransE**. Cette formule a dans un premier temps été modifiée, pour permettre d'avoir des scores de plausibilité plus faibles pour les anomalies par rapport aux vrais triplets. La formule était donc :

$$\text{local score} = -\left\|\mathbf{e}_s^{\text{base}} + \mathbf{e}_r^{\text{base}} - \mathbf{e}_o^{\text{base}}\right\|_1$$

Ainsi, plus le triplet est faux, plus la distance est grande et donc plus le score est bas puisque les scores sont négatifs. Cependant, nous avons constaté que le modèle avait du mal à reconnaître le faux du vrai pour des triplets de la forme $[o, p, s]$ (c'est-à-dire l'inverse du triplet

vrai $[s, p, o]$).

C'est pourquoi il a été envisagé de déléguer la tache de calcul du score local à un **MLP** (**M**ulti **L**ayer **P**erceptron), qui lui résout partiellement ce problème. Ainsi, nous construisons un réseau neuronal composé de 4 couches (1 couche d'entrée, 2 couches intermédiaires, 1 couche de sortie).

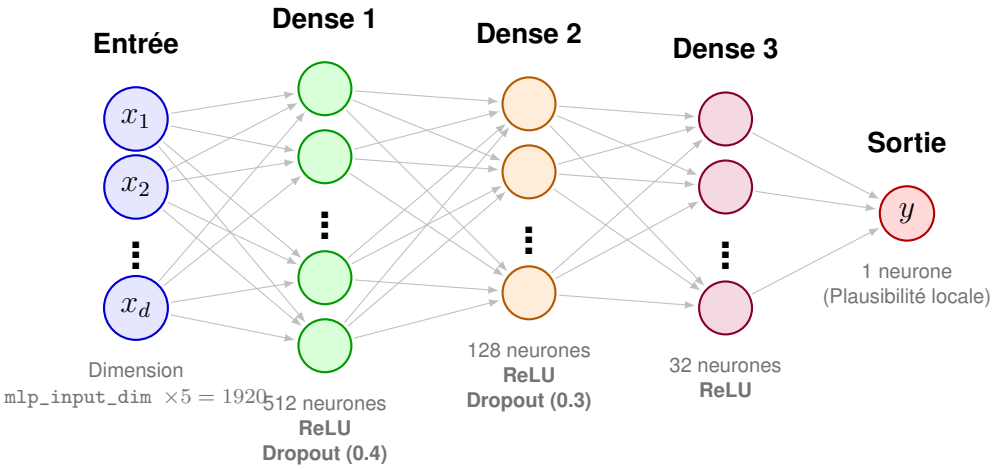


FIGURE 13 – Architecture détaillée du module `mlp_local`.

Comme illustré par la Figure 13, ce réseau de neurones prend en entrée un vecteur de dimension 1920. Cette dimension correspond à la concaténation des plongements (embeddings) initiaux du sujet, de la relation et de l'objet générés par Sentence-BERT (chacun de taille 384), auxquels s'ajoutent leur différence et leur produit scalaire ($5 \times 384 = 1920$ caractéristiques).

L'architecture détaillée se décompose ainsi :

— Couche dense 1 : **Réduit la dimensionnalité de l'espace d'entrée de 1920 à 512 neurones**. Elle est suivie d'une fonction d'activation **ReLU** pour introduire de la non-linéarité et d'une régularisation par **Dropout** (0.4) afin de prévenir le sur-apprentissage en désactivant aléatoirement 40 % des neurones lors de la phase d'entraînement.

— Couche dense 2 : **Compresse l'information de 512 à 128 neurones**. L'activation se fait de nouveau par **ReLU**, suivie d'un **Dropout** (0.3).

— Couche dense 3 : **Affine la représentation spatiale en passant de 128 à 32 neurones** (activation ReLU).

— Couche de sortie : **Constituée d'un unique neurone**, elle retourne une valeur scalaire brute (**logit**) qui représente le score de plausibilité locale du triplet.

Cette succession de couches denses permet de modéliser les interactions complexes et non linéaires entre les entités et la relation. Contrairement à une simple fonction de distance géométrique, ce MLP est capable d'apprendre l'asymétrie sémantique d'un fait. Il parvient ainsi

à faire la distinction structurelle entre un triplet valide $[s, p, o]$ et sa version inversée absurde $[o, p, s]$, corrigeant ainsi la faille identifiée dans l'approche précédente.

5.3 Suppression du Line Graph et utilisation du R-GCN

5.3.1 Suppression du Line Graph

Le modèle **LoGNet** suivait la logique suivante : ne pouvant capter les différents types de relations après calcul du score local, la méthode incluait la construction d'un Line Graph qui simplifie la structure originale en la réduisant à un graphe avec seulement un seul type de relation. Cependant cette méthode présentait des failles dont les principales sont :

— La démultiplication du nombre d'arêtes, qui peut parfois rendre le graphe plus complexe.
— La nécessité de reporter toute l'information du graphe original dans le Line Graph pour que le modèle puisse effectuer un raisonnement cohérent.

Puisque **Sentence-BERT** est capable de comprendre les mots et leur signification, on peut conclure que ce dernier est capable de capter les différents types de relations. Partant de ce constat, la construction du Line Graph n'est plus nécessaire.

5.3.2 Présentation du R-GCN

Reprenant les derniers éléments envoyés sur les GCN et le GAT (ce dernier étant d'ailleurs le modèle utilisé pour raisonner sur le Line Graph), un premier constat est que ces différents réseaux convolutionnels ne peuvent pas capter les différents types de relations.

Cependant, [SKB+18] propose une méthode où les GCN peuvent prendre en compte la diversité des relations dans un graphe de connaissances. Ces réseaux utilisant les méthodes de message passing sont appelés **R-GCN**.

Au cœur du R-GCN se trouve la règle de propagation qui met à jour l'état caché (la représentation) de chaque nœud en agrégeant les informations de ses voisins. La formule de mise à jour pour un nœud v_i à la couche $l+1$ est :

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right)$$

$h_i^{(l)}$: État caché du nœud i à la couche l.
$\mathcal{R}$: Ensemble des types de relations.
$\mathcal{N}_i^r$: Ensemble des indices des voisins du nœud i via la relation r.
$W_r^{(l)}$: Matrice de poids spécifique à la relation r et à la couche l.
$W_0^{(l)}$: Matrice de poids pour la connexion réflexive (self-connection), permettant au nœud de conserver sa propre information de la couche précédente.
$c_{i,r}$: Constante de normalisation (souvent choisie comme $c_{i,r} = |\mathcal{N}_i^r|$).
σ : Fonction d'activation non-linéaire (ex. ReLU).

On constate dans cette dernière formule que les différents types de relations sont pris en compte notamment avec $W_r^{(l)}$ qui est spécifique à chaque relation $r \in \mathcal{R}$. Toutefois la construction de telles matrices pour des relations très spécifiques et rarement présentes dans le graphe

risque d'augmenter considérablement le nombre de paramètres et ainsi de créer du sur-apprentissage (overfitting). C'est pourquoi des mécanismes de régularisation sont mis en place.

1. Décomposition de base (Basis decomposition) :

Les poids sont une combinaison linéaire d'un nombre restreint de matrices de base partagées.

$$W_r^{(l)} = \sum_{b=1}^{B} a_{rb}^{(l)} V_b^{(l)}$$

Avec $V_b^{(l)}$: Matrices de transformations de base.
$a_{rb}^{(l)}$: Coefficients spécifiques à la relation r.

2. Décomposition par blocs (Block-diagonal decomposition) :

Cette deuxième décomposition impose une contrainte de parcimonie (sparsity) en définissant $W_r^{(l)}$ comme une somme directe sur un ensemble de matrices de faible dimension.

$$W_r^{(l)} = \bigoplus_{b=1}^{B} Q_{br}^{(l)}$$

Cette structure regroupe les variables latentes de manière à ce qu'elles soient plus fortement couplées à l'intérieur d'un groupe qu'entre les groupes.

Ces réseaux sont utilisés pour la classification d'entités et la prédiction de liens. Concernant la prédiction d'entités (le cas qui nous intéresse dans notre projet), Le modèle empile plusieurs couches **R-GCN** et applique une fonction softmax sur le dernier état caché. L'apprentissage se fait en minimisant la perte d'entropie croisée (cross-entropy) sur les nœuds étiquetés :

$$\mathcal{L} = -\sum_{i \in \mathcal{Y}} \sum_{k=1}^{K} t_{ik} \ln h_{ik}^{(L)}$$

$\mathcal{Y}$: Ensemble des indices des nœuds ayant une étiquette.
t_{ik} : Étiquette de vérité terrain (ground truth).
$h_{ik}^{(L)}$: Sortie du réseau pour la classe k.

5.3.3 Utilisation du R-GCN dans notre modèle

L'objectif du R-GCN est d'attribuer un score global aux triplets. Pour éviter d'augmenter la complexité calculatoire, on va compresser les dimensions des vecteurs d'embeddings que le réseau peut prendre en entrée. Ainsi, **BERT** prend en entrée des vecteurs de taille 384, que nous allons compresser à 64.

```
all_nodes = th.arange(self.entity_count + 1, device=kg_triples.device)
    x_ent_base = self.ent_compress(self.entities_emb(all_nodes))
    eR_global = self.rel_compress(self.relations_emb(predicates))
```

Listing 5 – compression des embeddings

Enfin, on utilise applique le R-GCN à notre graphe par les commandes suivantes :

```
x_rgcn = self.rgcn(x_ent_base, pyg_data.edge_index, pyg_data.edge_attr
    )
x_rgcn = self.dropout(x_rgcn)

eS_global = x_rgcn[subjects]
eO_global = x_rgcn[objects]
```

Listing 6 – utilisation du R-GCN dans le graphe

Le **R-GCN** prend donc en entrée 3 paramètres :

1) Les vecteurs d'embeddings compressés de l'ensemble des entités et relations du graphe,
2) le graphe de connaissances original avant contamination (pyg_data.edge_index, qui correspond à la matrice d'adjacence du graphe de connaissances),
3) les différentes relations du graphe (pyg_data.edge_attr).

Remarque : Ici, on n'extrait pas le vecteur des relations avec le R-GCN. C'est normal puisque l'objectif initial de ce type de modèles est d'actualiser la position des nœuds dans un graphe en fonction des différentes relations qui les relient. Les relations sont donc vues comme des liens "fixes" sur lesquelles les matrices de poids seront construites.

5.4 Calcul du score de plausibilité globale

Après extraction des sujets et objets des triplets dans la matrice générée par le R-GCN, nous pouvons calculer ce qui sera le score de plausibilité globale de chaque triplet.

La norme **TransE** modifiée (voir 5.2 : Calcul du score de plausibilité locale) a été retenue. On a donc :

$$\text{global score} = -\left\|\mathbf{e}_s^{\text{global}} + \mathbf{e}_r^{\text{global}} - \mathbf{e}_o^{\text{global}}\right\|_1$$

5.5 Calcul du score de plausibilité finale

Le score de plausibilité finale se définit comme la combinaison convexe du score de plausibilité locale et du score de plausibilité globale. Il est exprimé suivant :

$$\text{plausibility score} = \alpha \times \text{local score} + (1-\alpha) \times \text{global score}$$

avec $\alpha \in [0, 1]$

Remarque : avec $\alpha = 0.5$, on obtient : **plausibility score** $= 0.5 \times$ **local score** $+ 0.5 \times$ **global score** et donc les deux scores ont la même importance dans la plausibilité finale.

Dans une telle configuration, avec $\alpha = 0.5$, **il y a une séparation claire entre les scores des triplets positifs et négatifs.**

En effet, le MLP calculant le score de plausibilité locale, plus le triplet est vrai, plus le score sera élevé. Pour un triplet faux, le score local va tendre vers 0.
En ce qui concerne la plausibilité globale, tous les scores calculés seront négatifs. Cependant, pour un triplet vrai, son score va tendre vers 0, signifiant que la distance entre les vecteurs du sujet et de l'objet à travers la relation les reliant est faible. Cette distance augmente à mesure que le triplet devient faux, faisant tendre le score global vers $-\infty$. Autrement dit, pour un triplet $x = [s, r, o]$ nous avons :

$\lim_{x \to \text{vrai}}$ local score $= +\infty$; $\lim_{x \to \text{faux}}$ local score $= 0^+$

$\lim_{x \to \text{vrai}}$ global score $= 0^-$; $\lim_{x \to \text{faux}}$ global score $= -\infty$

donc

$$\lim_{x \to \text{vrai}} \text{plausibility score} = +\infty$$

et

$$\lim_{x \to \text{faux}} \text{plausibility score} = -\infty$$

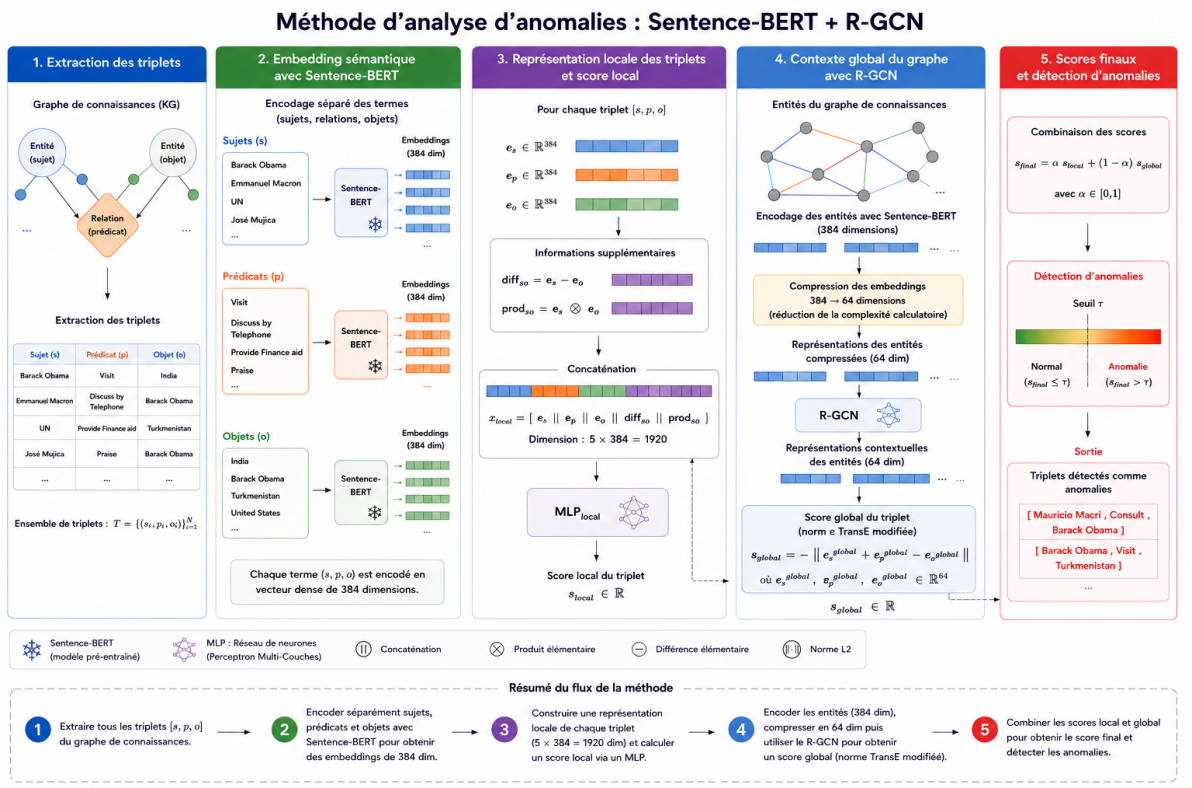


FIGURE 14 – Schéma récapitulatif du modèle BERT/R-GCN

6 Entraînement et évaluation du modèle

6.1 Bases de données utilisées

6.1.1 Présentation de la base de données ICEWS18

Ce modèle a d'abord été testé sur les snapshots utilisés pour le modèle LoGNet modifié. Puis, afin d'avoir une base de données plus importante en termes de taille, c'est sur **ICEWS18** que nous avons testé notre modèle.

ICEWS18 est un TKG (Temporal Knowledge Graph) qui décrit un ensemble de faits géopolitiques. Cette base de données se compose d'un ensemble d'entraînement (train_base), d'un ensemble de validation et d'un ensemble de test (test_base). Au regard de la taille du jeu de données d'entraînement, nous choisissons de tester notre modèle uniquement sur ce jeu de données. Ce dernier rassemble **23033 entités et 256 relations** et compte **373018 évènements**. Ces éléments se présentant au départ sous la forme d'index ont été convertis en triplets textuels après récupération des dictionnaires associés pour les entités et les relations.

6.1.2 Etude descriptive

L'étude descriptive de notre base de données fut une étape difficile. En effet, les données étant textuelles et présentées sous la forme de graphe, nous étions face à un cas de figure jusqu'alors jamais rencontré dans le cadre de notre formation. Par conséquent, appliquer les méthodes vues au cours de cette année pour analyser des données qualitatives et/ou quantitatives s'avère peu efficace pour analyser nos données. Ces déductions nous ont amené à étudier la structure du graphe plutôt que les éléments qui le composent pour mieux comprendre la structure de nos données. Pour réaliser cette tâche, l'utilisation de bibliothèques telles que **Pandas** ou encore **Networkx** a été déterminante.

Nous pouvons désormais présenter les éléments suivants :

Métrique	**Valeur**
Statistiques globales et sémantiques	
Nombre total d'entités ($\|\mathcal{E}\|$)	23 033
Nombre de relations uniques ($\|\mathcal{R}\|$)	256
Nombre total d'événements (triplets)	373 018
Statistiques topologiques	
Densité du graphe	0.000703
Degré moyen d'une entité	32.39 connexions
Statistiques temporelles	
Période couverte (Index temporels)	0 à 239 (240 *snapshots*)
Moyenne d'événements par *snapshot*	1 554

TABLE 3 – Caractéristiques globales et topologiques du jeu de données ICEWS18.

Dans le tableau ci-dessus, la densité du graphe mesure si le graphe a beaucoup d'arêtes ou peu. Un graphe dense (dense graph) est un graphe dans lequel le nombre d'arêtes (ou d'arcs) est proche du nombre maximal. Un graphe creux (sparse graph) a au contraire peu d'arêtes. La densité d'obtient en effectuant le calcul suivant :

$$\text{Densité} = \frac{\mid E \mid}{\mid V \mid \times(\mid V \mid -1)}$$

Il est également important de préciser ce qu'est le dégré moyen d'une entité. Elle correspond au nombre de relations que possède chaque entité du graphe. Elle se calcule de la manière suivante :

$$\text{Degré moyen} = \frac{2 \times \text{Nombre total d'arêtes (événements)}}{\text{Nombre total de nœuds (entités)}}$$

Top 5 des Relations	**Fréquence**	**Top 5 des Entités (*Hubs*)**	**Implication**
Make statement	17.2 %	United States	4.4 %
Consult	10.6 %	Citizen (India)	4.4 %
Express intent to meet or negotiate	5.6 %	Russia	3.5 %
Make an appeal or request	5.5 %	India	3.5 %
Arrest, detain, or charge...	4.5 %	Police (India)	3.0 %

TABLE 4 – Distribution des relations et des entités illustrant l'effet de longue traîne et la topologie centralisée du graphe ICEWS18.

L'exploration des données du corpus ICEWS18 (résumée dans le tableau 3) confirme la complexité inhérente aux graphes géopolitiques mondiaux. Avec une densité mesurée à 0.000703, le graphe est qualifié de très clairsemé (sparse). Par ailleurs, le tableau 4 met en évidence un net déséquilibre dans la distribution des données. D'une part, la topologie du réseau est centralisée autour de quelques hubs majeurs (comme les États-Unis ou l'Inde), typique d'une distribution en loi de puissance. D'autre part, la sémantique est sujette à un fort effet de longue traîne : les deux relations les plus courantes concentrent à elles seules près de 28% des événements, ce qui représente un défi d'apprentissage supplémentaire pour le modèle lors de la détection d'anomalies sur des relations rares.

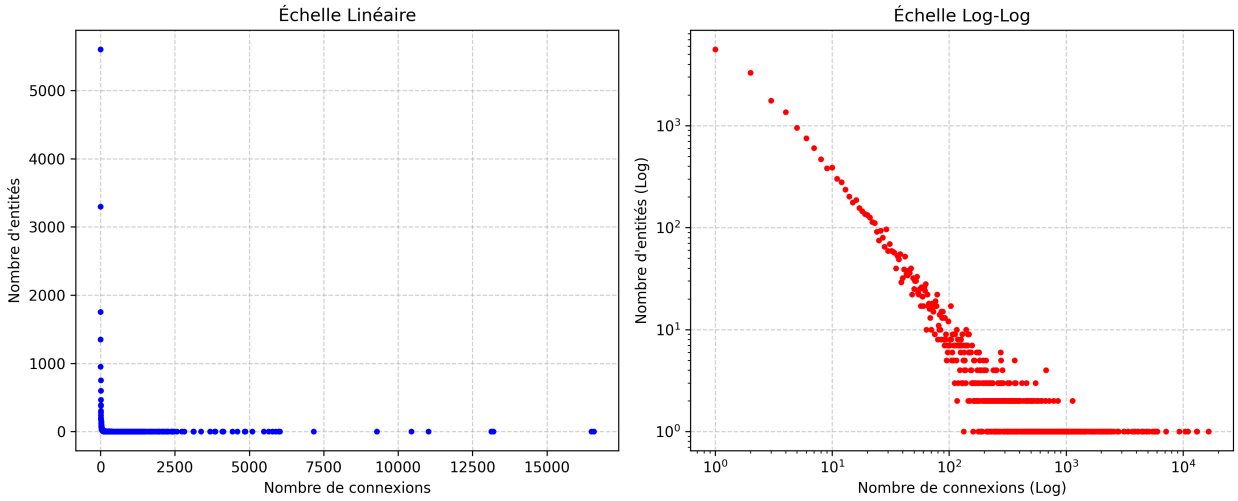


FIGURE 15 – Graphique de distribution des degrés (échelle linéaire et échelle log-log)

Afin de visualiser la topologie du réseau ICEWS18, nous avons tracé la distribution des degrés des entités (Figure 15). Le graphique en échelle linéaire (à gauche) illustre parfaitement l'effet de longue traîne : une écrasante majorité d'entités ne possède qu'une poignée de connexions, tandis qu'une infime minorité de Hubs accapare des milliers d'arêtes. Le passage en échelle logarithmique (à droite) transforme cette courbe en une droite décroissante, ce qui est la signature mathématique caractéristique des distributions à queue lourde (proches d'une loi de puissance). Cette hétérogénéité structurelle confirme la nécessité d'utiliser un modèle de type **R-GCN** pour propager efficacement le contexte sémantique des Hubs vers les entités isolées.

6.2 Génération des faux triplets (anomalies)

La tâche de génération des faux triplets a été confiée à un **LLM**, en l'occurence ici à **GPT 4.1**. L'objectif ici, contrairement à ce qui était appliqué dans LoGNet, n'est pas de générer des triplets sémantiquement faux en échangeant les entités des triplets du graphe (par exemple, pour 2 triplets ["Paris, "Capitale de", "France"] et ["Mbappé", "est joueur de", "Real Madrid"], on crée un troisième triplet faux ["Paris", "capitale de", "Mbappé"]).

L'objectif ici est de créer un triplet sémantiquement vrai (qui a du sens), mais qui porte sur une thématique différente de celle abordée dans le graphe initial. Par exemple, si notre graphe évoque la géopolitique, notre modèle devra être capable d'identifier un triplet portant sur la thématique du golf. Nous donnons donc ces premières instructions au LLM en page suivante.

```
    completion = client.chat.completions.create(
        model="openai/gpt-4.1",
        messages=[
            {"role": "user", "content": f"""
             Génère {nb_triplets} triplets sous la forme [Sujet, Relation,
                 Objet] en respectant STRICTEMENT ces règles :
             1. Utilise au moins une entité parmi cette liste pour chaque
                triplet : {data_entities}.

             2. Les thématiques doivent concerner des sujets au hasard (
                sport, astronomie, cuisine...) complètement différents de
                la géopolitique.

             3. Les triplets peuvent porter sur des faits vrais ou
                absurdes.

             4. N'écris AUCUN texte avant ou après. Pas de 'Voici les
                triplets' ni de numéros.

             5. Format exact : [Sujet,Relation,Objet]

             6. Utilise l'anglais pour les relations avec des underscores
                (ex: is_planet_of)."""}
        ]
    )
```

Listing 7 – première consigne donné au LLM pour la génération de faux triplets

Quelques précisions sont toutefois nécessaires quant aux consignes données :

- Dans la première consigne, la demande d'utilisation d'au moins une entité du graphe initial pour chaque triplet évite que soit construit un graphe d'anomalies disjointes du premier, qui rendrait trop facile la consigne de détection d'anomalies.
- Dans la 3^{eme} consigne, les triplets peuvent porter sur des faits vrais ou absurdes ce qui signifie que le LLM a le droit de se tromper dans les faits générés, tout en restant sémantiquement vrais.
- Les dernières consignes concernent la mise en forme des triplets car nous souhaitons pouvoir les convertir directement en tableau Numpy et les convertir en tenseurs par la suite.

Les faux triplets, incluant des entités nouvelles, sont ensuite indexés, puis ajoutés dans le graphe original. La consigne devient donc facile pour le modèle : tout ce qui ne porte pas sur la géopolitique devient anomalie. Les scores obtenus laissaient présager que le modèle avait trouvé une faille à nos instructions, associant tous les nouveaux triplets à des anomalies.

Nous décidons alors de créer des hard-négatives en envoyant cette nouvelle consigne :

```
        completion = client.chat.completions.create(

         model="openai/gpt-4.1",
         messages=[
             {"role": "user", "content": f"""
              Génère {nb_triplets} triplets d'anomalies sous la forme [
                 Sujet, Relation, Objet] en respectant STRICTEMENT ces rè
                 gles :

              1. Utilise au moins une entité parmi cette liste pour chaque
                 triplet : {data}.

              2. LE THEME DOIT RESTER STRICTEMENT GEOPOLITIQUE (guerre,
                 diplomatie, traités, etc).

              3. Cependant, l'événement décrit doit être historiquement,
                 géographiquement ou logiquement absurde.
                 (Ex: Le Vatican déclare la guerre nucléaire à Greenpeace,
                    ou un pays
                 s'envahit lui-même).

              4. N'écris AUCUN texte avant ou après.

              5. Format exact : [Sujet,Relation,Objet]

              6. Utilise l'anglais avec des underscores pour la relation (
                 ex: declare_nuclear_war_on)."""}

         ]

    )
```

Ainsi, en générant un fait en rapport avec le contexte du graphe, pour détecter une anomalie, BERT ne peut plus tricher sur la rareté de la relation dans le graphe. En effet, BERT pouvait aisément isoler un triplet sur le football des autres triplets portant sur la géopolitique juste en analysant la relation et les entités constituant le triplet en question. Désormais, c'est donc au R-GCN d'intervenir en analysant les nœuds à travers les différentes relations pour détecter ce qui est anormal.

Remarque : Une consigne encore plus exigeante pourrait être envisagée : elle viserait à créer des anomalies en utilisant uniquement les relations du graphe et en partageant toujours au moins une entité commune avec le graphe original.

Là où actuellement les triplets générés partagent au moins une entité avec le reste du graphe, ils partageraient désormais une entité et une relation avec le graphe, tout en restant sémantiquement vrais.

6.3 Modification de la fonction de perte

Dans le code initial, la fonction de perte utilisée était la Margin Ranking Loss. L'objectif de cette fonction est de calculer la perte en comparant à chaque itération deux triplets, x_1 et x_2, par rapport à un label fixé y. La formule est la suivante :

$$Loss(x_1, x_2, y) = max(0; -y(x_1 - x_2) + marge)$$

Cette fonction nécessite que les deux triplets en entrée, x_1 et x_2 soient de nature opposée, c'est-à-dire, un triplet vrai et un triplet faux. Dès lors, une première contradiction apparaît : **dans le cadre de notre projet, la détection d'anomalies au sein des graphes de connaissances ne doit permettre d'identifier qu'un petit nombre d'anomalies dans un grand jeu de données.** Or la condition imposée par la fonction de perte ci-dessus exige qu'il y ait un grand nombre d'anomalies au sein de notre graphe de connaissances, pour pouvoir effectuer des comparaisons cohérentes.

Il convient donc de changer cette fonction de perte pour en choisir une qui soit plus adaptée à notre objectif.

Parmi les différentes pistes explorées, nous choisissons d'utiliser la **BCE with Logits Loss function.** Cette fonction reprend le cadre de la fonction **BCE** (**B**inary **C**ross **E**ntropy) tout en corrigeant un défaut majeur de celle-ci quant aux données fournies en entrée. En effet, alors que la fonction BCE prenait en entrée des valeurs comprises entre 0 et 1, ce qui nécessitait au préalable l'application de la fonction sigmoïde aux différents scores calculés en amont pour chaque triplet, La nouvelle fonction prend en entrée des Logits, c'est à dire des scores positifs ou négatifs, pour lesquels la fonction sigmoïde sera appliquée directement au sein même de la fonction. Cette fonction est donc bien plus adaptée à notre cadre où la plausibilité des triplets est désignée par un score, qui peut être positif ou négatif.

La fonction **BCE with Logits Loss function** prend la forme suivante :

$$l_n = -w_n \left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n)) \right]$$

où w_n est un poids recalculé à chaque itération, y_n est le label du triplet n,
x_n le score de plausibilité du triplet n,
et $\sigma(x_n)$ désigne la fonction sigmoïde $(f(x) = \frac{1}{1+e^{-x}})$ appliquée à x_n.

6.4 Création du pipeline pour l'entraînement

Reprenant les méthodes utilisées dans [W+26], un graphe de connaissances temporelles peut-être vu comme un ensemble de sous-graphes d'évènements pris à différentes périodes t. A défaut de prendre pour le moment le caractère temporel du graphe (voir perspectives), on peut néanmoins décomposer le TKG en plusieurs sous graphes et faire raisonner notre modèle sur ceux-ci.

6.4.1 Préparation des données

Une première fonction, **prepare_snapshot_data** est construite. Cette fonction prend en entrée les triplets du graphe de connaissances et les anomalies générées par le LLM. On nomme les triplets positifs comme étant les triplets du graphe orignal, avant contamination. Les triplets négatifs, eux, correspondent aux anomalies générées.

A présent, un découpage peut être effectué sur les triplets du graphe. Un premier découpage est posé sur les triplets positifs, que l'on découpe en un ensemble d'entraînement et de test.

Puis, la même chose est appliquée aux triplets négatifs. Dans les deux cas, le découpage se présente sous la forme 80-20, c'est à dire que 80 % des triplets servent d'entraînement et 20 % servent pour le test.

La raison de ce double découpage s'explique par le fait qu'on ne souhaite donner en entraînement au R-GCN que les triplets **POSITIFS** du masque d'entraînement pour le calcul du score global. Or dans un graphe contaminé, si on ne fait qu'un seul découpage et que les triplets positifs sont donnés au R-GCN, le risque est que ce dernier modèle apprenne sur les triplets positifs de test.

On peut donc à présent reconstruire le graphe PyTorch qui servira de base pour le R-GCN. On obtient le code suivant :

```python
def prepare_snapshot_data(triples_np: np.ndarray, anomalies_ids_np: np.ndarray, num_entities: int, num_relations: int):

    # 1. SÉPARATION VRAIS/FAUX
    triples_neg_np = anomalies_ids_np
    num_pos = len(triples_np)
    num_neg = len(triples_neg_np)

    # 2. SPLIT TRAIN / TEST SUR LES POSITIFS
    indices_pos = np.arange(num_pos)
    train_idx_pos, test_idx_pos = train_test_split(
        indices_pos, test_size=0.2, random_state=RANDOM_STATE
    )
    # 3. CRÉATION DU GRAPHE (pyg_data) UNIQUEMENT AVEC LE TRAIN
    # C'EST LA CORRECTION MAJEURE : Le R-GCN ne triche plus
    triples_train_pos = triples_np[train_idx_pos]
    sources = triples_train_pos[:, 0]
    relations = triples_train_pos[:, 1]
    targets = triples_train_pos[:, 2]

    edge_index = th.tensor(np.stack([sources, targets], axis=0), dtype=th.long)
    edge_attr = th.tensor(relations, dtype=th.long)
    pyg_data = Data(edge_index=edge_index, edge_attr=edge_attr, num_nodes=num_entities)
    # 4. PRÉPARATION DES TENSEURS POUR L'ÉVALUATION (TransE)
    # On assemble les positifs et les négatifs
    all_triples_np = np.concatenate([triples_np, triples_neg_np])
    all_triples_tensor = th.tensor(all_triples_np, dtype=th.long)

    labels_tensor = th.tensor([1.0] * num_pos + [0.0] * num_neg, dtype=th.float32)

    # 5. MASQUES D'ENTRAINEMENT ET DE TEST
    # Le train contient les positifs de train + 80% des négatifs
    # Le test contient les positifs de test + 20% des négatifs
    indices_neg = np.arange(num_neg) + num_pos
    train_idx_neg, test_idx_neg = train_test_split(
        indices_neg, test_size=0.2, random_state=RANDOM_STATE
    )

    train_mask = th.zeros(len(all_triples_tensor), dtype=th.bool)
    train_mask[train_idx_pos] = True
    train_mask[train_idx_neg] = True
    test_mask = th.zeros(len(all_triples_tensor), dtype=th.bool)
    test_mask[test_idx_pos] = True
    test_mask[test_idx_neg] = True

    return {
        "pyg_data": pyg_data.to(DEVICE),
        "all_triples_tensor": all_triples_tensor.to(DEVICE),
        "all_triples_np": all_triples_np,
        "labels_tensor": labels_tensor.to(DEVICE),
        "train_mask": train_mask.to(DEVICE),
        "test_mask": test_mask.to(DEVICE),
        }
```

On remarque donc que le graphe qui servira de base pour le R-GCN n'est constitué que des triplets positifs du train mask. Par ailleurs, une attention particulière est portée sur le fait que les types de triplets (vrais et anomalies) respectent les mêmes proportions dans le train mask et le test mask.

En effet : Etant donné un graphe $\mathcal{G}$ on note x la proportion de triplets positifs et y la proportion de triplets négatifs, avec $y = \alpha x$ où $\alpha \in [0, 1]$. Le graphe complet comporte donc $x + y$ triplets.

Le découpage impose que les triplets, quel que soit leur type, soient séparés en 2 parties (80-20). On obtient :

$$x_{train} = 0.8x \text{ et } x_{test} = 0.2x$$

$$y_{train} = 0.8y \text{ et } y_{test} = 0.2y$$

Le masque d'entraînement dans notre code est la réunion des triplets d'entraînement positifs et négatifs : $train\ mask = x_{train} + y_{train} = 0.8x + 0.8y = 0.8(x + y)$

Par un raisonnement similaire appliqué à l'ensemble de test on obtient que $test\ mask = x_{test} + y_{test} = 0.2x + 0.2y = 0.2(x + y)$

Puisque $y = \alpha x$, **on déduit que quelque soit le masque utilisé, la proportion d'anomalies reste la même**.

Cette séparation des données suit la même logique que celle appliquée dans le modèle initial LoGNet (voir 4.1.4)

6.4.2 Choix des hyperparamètres

Pour que notre modèle fonctionne correctement, nous choisissons des hyperparamètres qui vont dicter le rythme et la manière dont notre modèle va apprendre. Afin de garantir la stabilité de l'apprentissage et la reproductibilité de nos expériences, le modèle a été paramétré selon les hyperparamètres suivants. Chacun de ces choix a été guidé par des observations empiriques visant à équilibrer la précision du modèle et son coût de calcul.

WEIGHT DECAY (10^{-2}) : Terme de régularisation L_2 appliqué par l'algorithme d'optimisation. Il pénalise les poids trop importants au sein des couches du réseau, ce qui permet de limiter drastiquement le sur-apprentissage (overfitting) et d'améliorer la capacité de généralisation du modèle sur les nouvelles données.

NOMBRE D'EPOCHS (150) : Nombre total d'itérations complètes sur l'ensemble du jeu de données d'entraînement. Fixé à 150, il définit la limite maximale du processus d'apprentissage dans le cas où le critère d'arrêt prématuré ne serait pas déclenché.

LR (5×10^{-4}) : Le taux d'apprentissage (**Learning Rate**). Il détermine l'amplitude des ajustements effectués sur les poids du réseau lors de la descente de gradient. Une valeur de 5×10^{-4} a été choisie pour assurer une convergence stable et progressive, évitant ainsi que le modèle

n'oscille de manière excessive autour du minimum de la fonction de perte.

WEIGHT PLAUSIBILITY (0.5) : Paramètre d'équilibrage central de notre architecture (noté α). Il contrôle la pondération entre le score de plausibilité locale (généré par le module MLP) et le score de plausibilité globale (généré par le R-GCN) lors du calcul du score final. Fixé à 0.5, il accorde une importance strictement symétrique aux composantes locales et globales pour l'identification des anomalies.

COMPRESSED DIM (64) : Dimension de l'espace de projection des plongements (embeddings). Ce paramètre indique que les vecteurs sémantiques initiaux (par exemple, de taille 384 en sortie de Sentence-BERT) sont compressés via une couche linéaire en vecteurs de dimension 64 avant d'alimenter le R-GCN. Cette compression permet de réduire significativement la complexité spatiale et temporelle tout en préservant l'information topologique essentielle.

PATIENCE (15) : Paramètre régissant le mécanisme d'arrêt prématuré (Early Stopping). Il indique que l'entraînement sera automatiquement interrompu si la fonction de perte évaluée sur l'ensemble de test ne montre aucune amélioration pendant 15 époques consécutives. Cela prévient le sur-apprentissage sur les dernières époques.

RANDOM STATE (42) : Graine aléatoire (seed) globale du programme. Elle fige le comportement des générateurs de nombres pseudo-aléatoires pour l'initialisation des poids du réseau, la génération des faux triplets et la répartition probabiliste des masques d'entraînement et de test. Son utilisation garantit la stricte reproductibilité des résultats présentés dans cette étude.

MIN TRIPLES (0) : Seuil de filtrage structurel. Il définit le nombre minimum d'arêtes (triplets) requis pour qu'un sous-graphe (snapshot temporel) soit analysé. Fixé à 0, il instruit le pipeline de traiter l'intégralité des snapshots historiques, même les plus clairsemés.

6.4.3 Métriques utilisées

Dans cette sous-partie nous allons revenir sur ce qui nous permet d'évaluer la capacité du graphe à réaliser des bonnes prédictions.

Courbes ROC/AUC

Comme défini précédemment lors de l'étude du modèle initial (cf. Section 4.1.5), la courbe **ROC** et son aire sous la courbe (**AUC**) demeurent nos métriques de référence pour évaluer les performances de prédiction.

Dans le cadre de notre nouvelle architecture, le choix de l'**AUC** est d'autant plus crucial qu'elle est particulièrement robuste face aux jeux de données présentant un fort déséquilibre de classes (ici, seulement 6 anomalies noyées parmi des centaines de triplets normaux dans l'ensemble de test). Elle permet d'évaluer la capacité intrinsèque de notre modèle hybride à attribuer un score de plausibilité plus faible à une anomalie qu'à un fait géopolitique avéré.

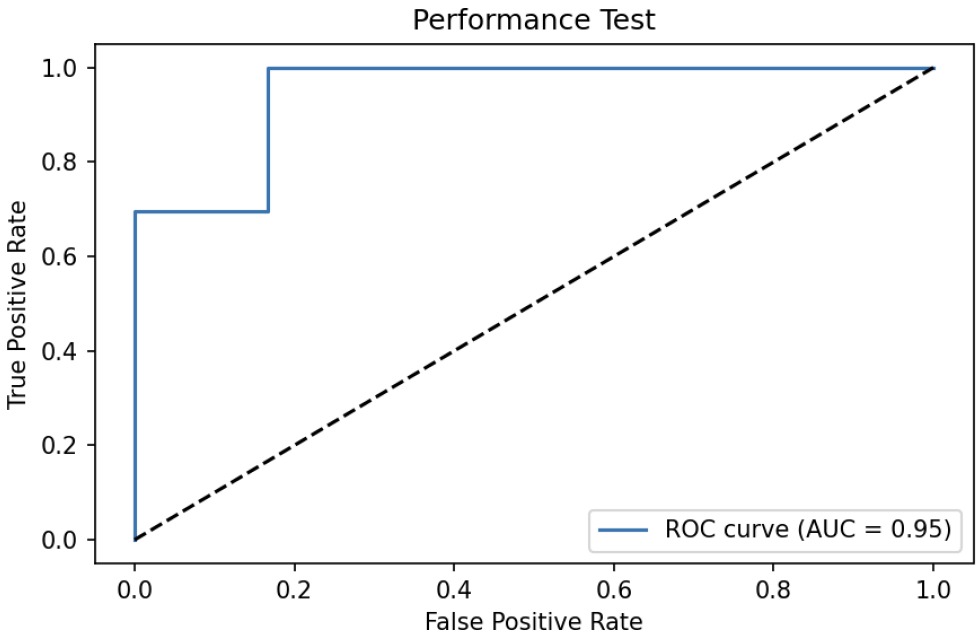


FIGURE 16 – Exemple de Courbe ROC avec mesure de l'AUC sur un des snapshots étudiés

Courbes de perte test vs train

La courbe de perte permet notamment de mettre en évidence l'overfitting du modèle. Pour plus de lisibilité, on compare sur le même graphique la perte pour l'ensemble d'entraînement et de test.

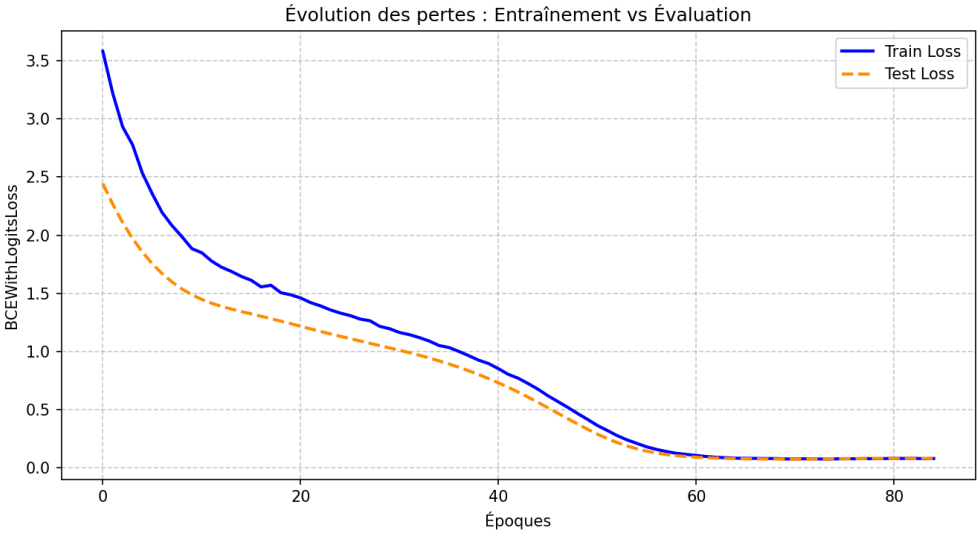


FIGURE 17 – Exemple de Courbes de perte Train vs Test sur un des snapshots étudiés

Ici on voit que les courbes de perte se superposent, ce qui confirme que notre modèle à une très bonne capacité de généralisation sur l'ensemble de test.

Boxplots des scores locaux, globaux et finaux

Une méthode qui nous permet de capturer rapidement la répartition des triplets est de tracer les boxplots des scores à chaque étape.

Si le modèle a une bonne capacité de détection, on s'attend à avoir des boîtes très éloignées l'une de l'autre entre les triplets normaux et anormaux.

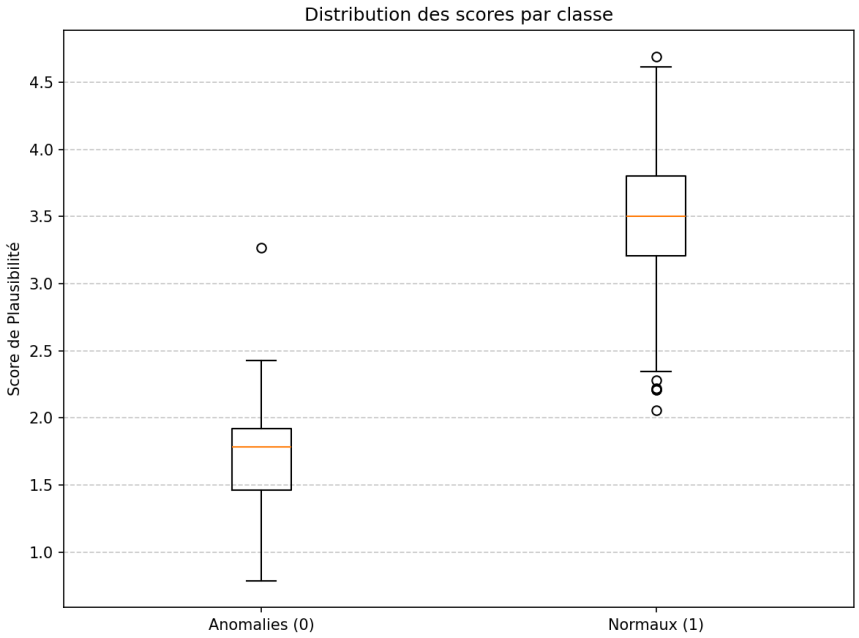


FIGURE 18 – Exemple de Boxplots des scores finaux de plausibilité sur un des snapshots étudiés

On peut constater ici que les scores sont bien distincts entre les triplets positifs et négatifs. Si la médiane des triplets positifs se situe autour de 3.5, celle des triplets négatifs est d'environ 1.8.

Remarque : ici tous les scores finaux sont positifs. On aurait pu s'attendre à ce que les scores des faux triplets soient négatifs. Cela est dû à la grande différence de scoring des scores locaux, qui sont bien plus élevés que les scores globaux.

Création de fichiers CSV d'affichage des anomalies

Afin de mieux observer les triplets considérés comme anomalies, on crée les fichiers CSV suivants pour chaque snapshot :

- Un premier fichier qui renvoie tous les scores obtenus (local, global, final) pour tous les triplets sur graphe.
- Un deuxième tableau qui renvoie les anomalies injectées dans le snapshot.
- Un dernier fichier qui renvoie les anomalies détectées. Pour ce dernier tableau nous sommes obligés de choisir un seuil qui servira de référence et sur lequel le modèle

pourra s'appuyer pour retourner les anomalies. Afin de déterminer le seuil de classification optimal τ^*, nous cherchons le point de la courbe ROC qui se rapproche le plus du classifieur théorique parfait. Dans l'espace ROC, ce classifieur idéal est situé aux coordonnées $(0, 1)$, ce qui correspond à un taux de faux positifs (FPR) nul et un taux de vrais positifs (TPR) de 100%. Pour chaque seuil de classification τ évalué par le modèle, la distance euclidienne $d(\tau)$ entre le point de fonctionnement sur la courbe ROC $(\text{FPR}(\tau), \text{TPR}(\tau))$ et le point idéal $(0, 1)$ est calculée selon la formule :

$$d(\tau) = \sqrt{\text{FPR}(\tau)^2 + (1 - \text{TPR}(\tau))^2}$$

Le seuil optimal retenu τ^* est celui qui minimise cette distance euclidienne :

$$\tau^* = \underset{\tau}{\text{argmin}} \left(\sqrt{\text{FPR}(\tau)^2 + (1 - \text{TPR}(\tau))^2} \right)$$

On a donc la fonction suivante dans notre code :

```python
def best_roc_threshold(y_true: np.ndarray, y_scores: np.ndarray)
    -> tuple[float, float]:
    fpr, tpr, thresholds = roc_curve(y_true, y_scores)
    roc_auc = auc(fpr, tpr)
    dist = np.sqrt(fpr**2 + (1 - tpr) ** 2)
    ix = np.argmin(dist)
    return float(thresholds[ix]), float(roc_auc)
```

6.5 Résultats et interprétation

L'évaluation de notre architecture hybride **BERT/R-GCN** révèle une amélioration drastique des performances prédictives par rapport à la méthode initiale basée sur LoGNet. Pour valider notre approche tout en maîtrisant les temps de calcul liés à l'inférence par apprentissage profond, nos expérimentations se sont concentrées sur un sous-échantillon représentatif du jeu de données ICEWS18. En effet, l'exécution du pipeline complet (Sentence-BERT et R-GCN) nécessite une puissance de calcul importante : le traitement de ces premiers graphes requiert environ 1 heure 15 minutes sur une machine équipée d'un GPU. Ce sous-échantillon permet toutefois d'établir une preuve de concept (Proof of Concept) statistiquement robuste, dans la mesure où il regroupe une diversité d'entités et de relations représentative de la densité globale du réseau géopolitique.

Protocole de contamination et robustesse au déséquilibre

Contrairement à une approche par ratio fixe, nous avons fait le choix méthodologique d'injecter un nombre constant d'anomalies (exactement 30 triplets négatifs générés par le LLM) dans chaque *snapshot*. Par conséquent, le taux de contamination varie de $4,3\%$ (pour le Snapshot 0) à seulement $2,1\%$ (pour le Snapshot 3). Après application de notre masque de stratification (80/20), l'ensemble de test ne contient à chaque itération que 6 anomalies isolées au milieu de centaines de faits avérés. Ce protocole simule des conditions d'observation réalistes et évalue la véritable capacité du modèle à détecter des signaux faibles dans un contexte de fort déséquilibre de classes.

TABLE 5 – Évaluation de l'aire sous la courbe (AUC) pour les *snapshots* de l'échantillon de la base de données ICEWS18.

Snapshot	**Taille du graphe (Triplets)**	**AUC Entraînement**	**AUC Test**
Snapshot 0	665	1.0000	0.9987
Snapshot 1	1123	0.9890	1.0000
Snapshot 2	1320	0.9469	0.9893
Snapshot 3	1398	0.9568	0.9446
Snapshot 4	1216	0.9621	0.9904
Snapshot 5	853	1.0000	1.0000
Snapshot 6	745	0.9999	0.9933
Snapshot 7	1308	0.9792	0.9517
Snapshot 8	1831	0.9373	0.9596
Snapshot 9	1895	0.9621	0.9710

Analyse des performances prédictives

L'analyse des performances prédictives dans le tableau 7 met en exergue l'excellente acuité de détection de l'architecture proposée. En effet, l'aire sous la courbe (AUC) évaluée sur l'ensemble de test se maintient systématiquement au-dessus du seuil de 0.94, allant jusqu'à atteindre un score parfait sur certains graphes instantanés (snapshot 1 et 5). Ces résultats valident empiriquement la pertinence de l'approche hybride : la synergie entre l'indépendance sémantique locale, modélisée par Sentence-BERT, et l'intégration topologique globale, assurée par le R-GCN, s'avère particulièrement discriminante pour distinguer les faits avérés des signaux anormaux.

Outre cette précision, le modèle fait preuve d'une forte capacité de généralisation. L'écart quasi nul observé entre l'AUC d'entraînement et celle de test atteste d'un apprentissage stable. Cette stabilité métrique confirme le calibrage adéquat des mécanismes de régularisation mis en œuvre, tels que le taux d'abandon (Dropout) et la pénalisation des poids (Weight Decay). Le modèle évite ainsi l'écueil du sur-apprentissage et parvient à classer avec succès des triplets demeurés invisibles durant sa phase d'entraînement.

Enfin, la robustesse du modèle s'illustre par son indépendance face aux variations d'échelle et de densité du réseau. Bien que le volume d'arêtes varie de manière significative — allant du simple au triple entre les snapshots 0 et 9 — et que la dilution des anomalies soit extrême, le pouvoir discriminant de l'architecture reste stable. La légère inflexion de la métrique observée sur le Snapshot 3 (avec une AUC de 0.9446) s'interprète naturellement : une densification de la structure du graphe, intrinsèquement corrélée à une augmentation du bruit topologique, complexifie marginalement l'identification des signaux faibles sans pour autant dégrader la fiabilité globale du pipeline.

7 Conclusion et Perspectives

7.1 Critiques sur la méthode BERT/R-GCN

Ces nouveaux travaux, dont le point de départ initial était une volonté de prendre en compte la sémantique des triplets, nous ont finalement amené à repenser l'architecture de notre modèle en sacrifiant son outil majeur, le **Line Graph**.

Cependant, la nouvelle approche combinant Sentence-BERT (pour la détection d'anomalies locales) et un R-GCN (pour l'analyse de la structure relationnelle) affiche des performances nettement supérieures. Cette architecture hybride parvient à identifier efficacement les anomalies, même lorsqu'elles sont marginales.

Toutefois, cette amélioration s'accompagne de contraintes de scalabilité qu'il convient de souligner. La principale limite concerne la complexité spatiale et temporelle de l'architecture. Le passage à l'échelle sur l'intégralité d'un corpus tel qu'ICEWS18 (qui comporte 240 snapshots) s'avère particulièrement lourd computationnellement. Ce goulet d'étranglement est imputable en grande partie au R-GCN : la construction systématique de matrices de poids spécifiques pour chaque nouveau type de relation démultiplie le temps de calcul et l'empreinte mémoire, et ce, malgré les techniques de décomposition et d'optimisation mathématique évoquées précédemment (voir 5.3.2).

D'autre part, dans l'objectif de détecter des anomalies sur les graphes de connaissances temporelles, on peut remarquer qu'ici notre méthode ne s'intéresse qu'a la sémantique des triplets (plausibilité locale) et leur position dans le graphe (plausibilité globale). Il serait également utile de prendre en compte le contexte historique des triplets dans un graphe pour détecter des anomalies entre différents snapshots.

7.2 Utilité de cette méthode dans les travaux futurs

La méthode **BERT/R-GCN**, en isolant les évènements structurellement et sémantiquement atypiques, constitue la première brique indispensable à l'identification de signaux faibles. En effet, dans des domaines tels que la géopolitique ou la finance, une anomalie n'est pas nécessairement une erreur de donnée, mais souvent le signe précurseur d'un évènement majeur (un signal faible). Bien que des évaluations comparatives (benchmarks) plus exhaustives soient nécessaires pour la positionner face à l'état de l'art, notre architecture hybride offre d'ores et déjà un socle de détection robuste, interprétable et hautement discriminant.

7.3 La prise en compte de l'historique des snapshots pour la détection d'anomalies dans les TKG

La courte durée de ce stage a malheureusement empêché de poursuivre nos travaux. Toutefois, nous avions déjà imaginé une suite à notre modèle pour prendre en compte la dimension temporelle.

En effet, il était envisagé de reprendre un réseau récurrent, comme le **LSTM**, pour réaliser non pas l'embedding des triplets (comme ce fut le cas dans la méthode LogNet) mais pour prendre

la dimension historique des graphes de connaissances. Ainsi, nous aurions pu, en se basant sur les précédents snapshots (aux dates $t-1$, $t-2$ par exemple) réaliser des prédictions pour le snapshot en date t. Nous aurions ensuite comparé nos prédictions aux véritables événements contenus dans le graphe à cette période pour obtenir un score de plausibilité historique (historical plausibility score). Ce dernier score serait venu s'ajouter dans le calcul du score final de plausibilité et aurait apporté une vraie évolution quant à la détection d'anomalies sur les graphes de connaissances temporelles.

7.4 Conclusion globale

Cette période de stage au sein du **LIRIS** a constitué une étape charnière de mon parcours, en m'offrant la chance de découvrir de l'intérieur le monde de la recherche appliquée. Plus qu'une simple mise en pratique de ma formation académique, cette expérience m'a véritablement enseigné l'exigence de la démarche scientifique : avancer par itérations, accepter que les modèles initiaux soient remis en cause, et surtout, m'enrichir des échanges constants avec l'équipe de recherche.

Sur le plan technique, j'ai tiré une immense satisfaction de ma montée en compétences sur des architectures aussi complexes et stimulantes que les modèles de langage (LLMs) et les réseaux de neurones sur graphes (GNNs). Aborder ces sujets en profondeur a été une découverte passionnante.

Au-delà des résultats obtenus lors de ce projet, ce stage m'a surtout fait prendre conscience de l'immensité des défis liés à l'explicabilité de l'Intelligence Artificielle. Ce stage a réellement été enrichissant et a confirmé ma volonté de continuer à étudier dans le champ de l'IA, qui je pense, va s'inscrire comme un élément incontournable de la science des données dans les années à venir.

Références

[G+23] Y. Gu et al. Don't generate, discriminate : Proposal for grounding language models to real-world environments. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

[HBC+21] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, Roberto Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Computing Surveys*, 54(4) :37 :1–37 :37, 2021.

[L+23] Jaejun Lee et al. INGRAM : Inductive knowledge graph embedding via relation graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[L+24] L. Luo et al. Reasoning On Graphs (ROG) : Faithful and interpretable large language model reasoning. In *International Conference on Learning Representations (ICLR)*, 2024.

[L+26] L. Luo et al. G-Reasoner : Foundation models for unified reasoning over graph structured knowledge. In *International Conference on Learning Representations (ICLR)*, 2026.

[LMB+24] Yassir Lairgi, Ludovic Moncla, Khalid Benabdeslem, Rémy Cazabet, and Pierre Cléau. iText2KG : Incremental knowledge graphs construction using large language models. In *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*, 2024.

[LMB+26] Yassir Lairgi, Ludovic Moncla, Khalid Benabdeslem, Rémy Cazabet, and Pierre Cléau. ATOM : AdapTive and OptiMized dynamic temporal knowledge graph construction using LLMs. In *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*, 2026.

[M+24] J. Ma et al. Debate on Graph (DOG) : A flexible and reliable reasoning framework for large language models. In *AAAI Conference on Artificial Intelligence*, 2024.

[Pir22] G. Pirro. LoGNet : Local and global triple embedding network. In *International Semantic Web Conference (ISWC)*, 2022.

[S+24] J. Sun et al. Think-On-Graph (TOG) : Deep and responsible reasoning of large language models on knowledge graphs. In *International Conference on Learning Representations (ICLR)*, 2024.

[SKB+18] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web : 15th International Conference, ESWC 2018, Proceedings*, pages 593–607. Springer, 2018.

[W+26] C. Wang et al. Temporal knowledge graph reasoning using global and recent history information. *Scientific Reports*, 16, 2026.

[WSL+24] Jiapu Wang, Kai Sun, Linhao Luo, Wei Wei, Yongli Hu, Alan Wee-Chung Liew, Shirui Pan, and Baocai Yin. Large language models-guided dynamic adaptation for temporal knowledge graph reasoning. In *Advances in Neural Information Processing Systems*, 2024.

Annexes

*Afin de garantir la concision et la lisibilité du présent document, les figures et résultats présentés dans cette annexe se concentrent sur le **Snapshot 0**, choisi à titre d'exemple représentatif. Les neuf autres snapshots évalués dans le cadre de cette étude (voir Section 6.5) affichent des comportements d'apprentissage, des métriques de performance et des séparations de classes strictement analogues, confirmant la stabilité et la robustesse de l'architecture hybride BERT/R-GCN face à la variation temporelle du graphe.*

A Évaluation des performances (Exemple sur le Snapshot 0)

Cette section détaille la courbe ROC/AUC ainsi que l'évolution des pertes (apprentissage versus évaluation) pour le snapshot initial, attestant de l'excellente capacité de généralisation et de la stabilité de l'apprentissage.

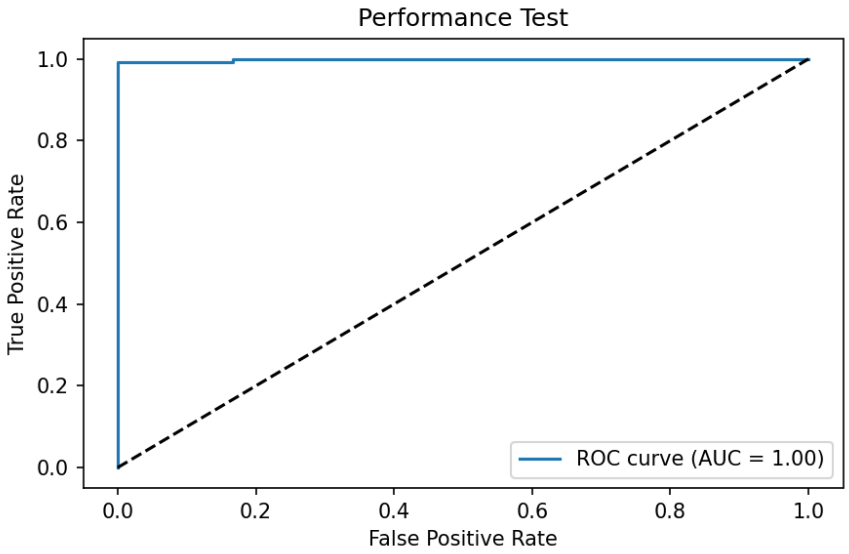


FIGURE 19 – Courbe ROC et aire sous la courbe (AUC) pour l'ensemble de test du snapshot 0.

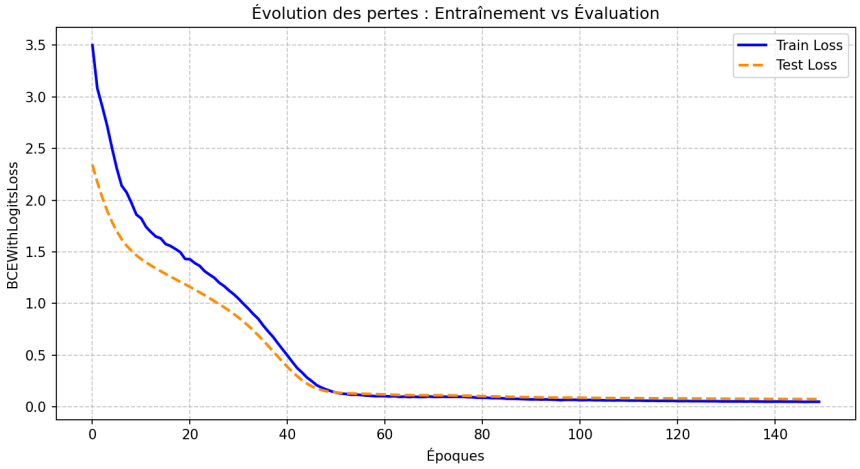
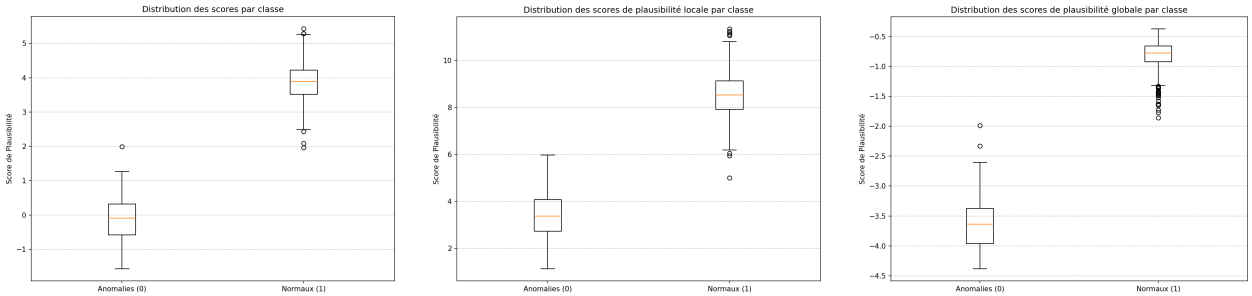


FIGURE 20 – Évolution comparée des pertes (Train vs Test) sur le snapshot 0.

B Distribution des scores de plausibilité

Les diagrammes de dispersion (*boxplots*) ci-dessous illustrent la séparation nette obtenue entre les scores de plausibilité des triplets normaux et ceux des anomalies injectées. Ce grand écart entre les médianes se retrouve de manière constante sur l'intégralité des snapshots analysés.



Plausibilité finale Plausibilité locale Plausibilité globale

FIGURE 21 – Distribution comparée des scores de plausibilité (final, local, global) pour le snapshot 0.

La figure 21 illustre parfaitement la capacité du modèle BERT/R-GCN à discriminer les anomalies, en leur attribuant un score plus faible. Sur ce snapshot cette discrimination semble plus marquée lors du calcul du score de plausibilité globale. Cela confirme la capacité du R-GCN à raisonner sur la structure pour identifier les éléments anormaux.

Nous constatons ainsi une cohérence entre les scores locaux, globaux et finaux. Le score médian de plausibilité finale des triplets normaux est nettement positif, tandis que celui des anomalies est négatif. Les précédentes affirmations dans le calcul des scores (5.5 : calcul du score de plausibilité finale) sont donc vérifiées ici.

C Exemples qualitatifs de signaux faibles et anomalies détectés

La figure ci-dessous présente un extrait des triplets géopolitiques anormaux générés par le modèle de langage (*hard-negatives*) pour le snapshot 0. Ces anomalies contextuelles ont été correctement identifiées par notre architecture grâce à un score de plausibilité inférieur au seuil optimal τ^*. Une efficacité de détection similaire a été validée empiriquement sur les autres sous-graphes temporels.

Triplet	Label	Plausibilite_Locale	Plausibilite_Globale	Score_Final	Detection_Correcte
Islamic Preacher (Afghanistan) | sign_surrender_to | Mars	0.0	2.2985713	-3.4211502	-0.5612894	True
Media Personnel (Afghanistan) | draft_constitution_for | Atlantis	0.0	3.3887577	-3.3731318	0.007812977	True
Kim Jong-Un | sign_peace_treaty_with | Antarctica	0.0	3.1577828	-2.3285565	0.41461313	True
United Nations | bombard | Finland	0.0	4.7342467	-3.2337022	0.7502723	True
United Nations | purchase | Mount_Everest	0.0	5.7075315	-3.1615727	1.2729794	True
Stefan Löfven | Make statement | Sweden	1.0	5.005599	-1.0805326	1.9625332	False
Police (Philippines) | establish_monarchy_in | United_States	0.0	5.9688573	-1.9906663	1.9890954	True

FIGURE 22 – Anomalies détectées sur l'ensemble de test du snapshot 0

Sur ce dernier tableau 22, on remarque que toutes les anomalies présentes dans l'ensemble de test (6 anomalies) ont été détectées. Seule 1 évènement réel a été classé à tort comme anomalie sur un total de 139 triplets, ce qui représente une erreur marginale et explique l'AUC de $0,9987$ sur ce snapshot.

D Comparaisons avec le premier modèle

Nous avons également testé le modèle LoGNet sur la base de données ICEWS18 afin de vérifier les performances de ce modèle sur une base de données plus importante. Afin que l'expérience soit la plus juste possible, nous avons choisi de générer les anomalies par LLM comme dans le cadre de la méthode BERT/R-GCN. Cependant, en raison des différences fondamentales entre les deux modèles, il est nécessaire d'adapter la méthode de génération des anomalies. L'objectif est d'empêcher LoGNet d'exploiter un biais d'apprentissage : il classerait systématiquement comme anomalie toute nouvelle entité générée par le LLM, sous le seul prétexte qu'elle est absente de ses données d'entraînement.

```python
def generer_anomalies_llm(nb_triplets, echantillon_entites, echantillon_relations) -> np.ndarray:
client = OpenAI(base_url="https://openrouter.ai/api/v1", api_key=cle)

completion = client.chat.completions.create(
    model="openai/gpt-4o",
    messages=[
        {"role": "user", "content": f"""
         Génère {nb_triplets} triplets d'anomalies sous la forme [Sujet, Relation, Objet].
         RÈGLES STRICTES :
         1. Le Sujet et l'Objet DOIVENT être piochés UNIQUEMENT dans : {echantillon_entites}.
         2. La Relation DOIT être piochée UNIQUEMENT dans : {echantillon_relations}.
         3. N'invente AUCUN nom.
         4. Crée un événement géopolitique HAUTEMENT PLAUSIBLE, mais FACTUELLEMENT FAUX (qui n'a jamais eu lieu).
         5. N'utilise pas l'absurde. Les combinaisons doivent sembler réelles.
         6. Format exact attendu : [Sujet,Relation,Objet]"""}
    ]
)
```

Les résultats obtenus sur les 10 premiers snapshots, même si ils sont bien meilleurs, diffèrent beaucoup d'un snapshot à l'autre, prouvant les difficultés de notre premier modèle à obtenir une certaine stabilité. On peut représenter les résultats dans le tableau suivant.

TABLE 6 – Évaluation de l'aire sous la courbe (AUC) pour les *snapshots* de l'échantillon de la base de données ICEWS18 avec la méthode LoGNet modifiée.

Snapshot	**Taille du graphe (Triplets)**	**AUC Entraînement**	**AUC Test**
Snapshot 0	665	0.9997	0.9989
Snapshot 1	1123	0.9905	0.8819
Snapshot 2	1320	0.9905	0.9416
Snapshot 3	1398	0.9710	0.9066
Snapshot 4	1216	0.9645	0.8566
Snapshot 5	853	0.9962	0.9168
Snapshot 6	745	0.9590	0.7363
Snapshot 7	1308	0.9839	0.8566
Snapshot 8	1831	0.9319	0.8054
Snapshot 9	1895	0.8858	0.6209

TABLE 7 – Évaluation de l'aire sous la courbe (AUC) pour les *snapshots* de l'échantillon de la base de données ICEWS18 avec la méthode BERT/R-GCN.

Snapshot	**Taille du graphe (Triplets)**	**AUC Entraînement**	**AUC Test**
Snapshot 0	665	1.0000	0.9987
Snapshot 1	1123	0.9890	1.0000
Snapshot 2	1320	0.9469	0.9893
Snapshot 3	1398	0.9568	0.9446
Snapshot 4	1216	0.9621	0.9904
Snapshot 5	853	1.0000	1.0000
Snapshot 6	745	0.9999	0.9933
Snapshot 7	1308	0.9792	0.9517
Snapshot 8	1831	0.9373	0.9596
Snapshot 9	1895	0.9621	0.9710

Cette expérience permet de montrer que le premier modèle que nous avons tenté de reproduire permet de fournir de meilleurs résultats sur des bases de données plus importantes, et vient confirmer la supériorité de la méthode BERT/R-GCN pour la détection d'anomalies.

E Analyse de la complexité temporelle du modèle BERT/R-GCN

L'évaluation de la viabilité d'un modèle d'apprentissage sur graphes nécessite une analyse de sa complexité temporelle lors de la phase de propagation avant (*forward pass*). Notre architecture hybride traitant simultanément la sémantique locale et la topologie globale, il convient de décomposer ce calcul.

Posons les variables suivantes définissant les dimensions du système à un instant t :

- N : Nombre total d'entités (nœuds) dans le dictionnaire global.
- E : Nombre de relations (arêtes) actives dans le graphe d'entraînement.
- B : Taille du lot évalué (nombre de triplets traités simultanément, B correspondant ici à l'intégralité du *snapshot*).
- d_b : Dimension des plongements initiaux issus de Sentence-BERT ($d_b = 384$).
- d_c : Dimension compressée pour le traitement topologique ($d_c = 64$).
- d_h : Dimension maximale de la couche cachée du réseau perceptron multicouche (MLP) local ($d_h = 512$).

1. Complexité de la voie locale (Sémantique)

La composante locale évalue la cohérence intrinsèque des triplets via un MLP. Ce traitement vectorisé est indépendant de la taille globale du graphe (N ou E). La complexité est dominée par la projection de l'entrée concaténée (de dimension proportionnelle à d_b) vers la couche cachée d_h. La complexité temporelle locale s'élève à :

$$\mathcal{O}(B \cdot d_b \cdot d_h)$$

2. Complexité de la voie globale (Topologique)

Le traitement de la structure relationnelle repose sur un réseau de convolution sur graphes relationnels (R-GCN). Cette étape s'effectue en deux temps :

1. **Compression des nœuds :** L'ensemble des N entités est projeté de l'espace sémantique dense (d_b) vers un espace réduit (d_c) afin d'alléger le calcul tensoriel. Coût : $\mathcal{O}(N \cdot d_b \cdot d_c)$.
2. **Propagation des messages (Message Passing) :** Le R-GCN met à jour les représentations en agrégeant le voisinage pour chaque type de relation. Coût de l'agrégation : $\mathcal{O}(E \cdot d_c^2 + N \cdot d_c^2)$.

3. Complexité globale et justification architecturale

En combinant l'évaluation globale (R-GCN) et le calcul du score global (de coût $\mathcal{O}(B \cdot d_c)$), la complexité temporelle asymétrique globale du modèle se définit par :

$$\mathcal{O}\Big(\underbrace{B \cdot d_b \cdot d_h}_{\text{Sentence-BERT+MLP}} + \underbrace{N \cdot d_b \cdot d_c}_{\text{Compression}} + \underbrace{E \cdot d_c^2 + N \cdot d_c^2}_{\text{R-GCN}} \Big)$$

Cette formulation mathématique justifie notre choix d'architecture à deux voies. L'intégration d'une couche de compression réduisant d_b à d_c empêche le terme $\mathcal{O}(E \cdot d_b^2)$ du R-GCN de saturer la mémoire de calcul.

Piste d'optimisation : Actuellement, le terme lié à la compression dépend de N (le dictionnaire global, soit plus de 23 000 entités pour ICEWS18). Dans le cadre d'un passage à l'échelle industrielle, l'extraction dynamique d'un sous-graphe induit (*induced subgraph*) pour chaque *snapshot* permettrait de restreindre N aux seules entités actives à l'instant t, réduisant ainsi drastiquement l'empreinte spatio-temporelle du modèle.