SAAD DAHLEB UNIVERSITY OF BLIDA 1

Faculty of Science

Department of Computer Science

MASTER'S THESIS

In Computer Science

Specialization: Intelligent Systems Engineering

THESIS TOPIC :

Deep Graph Clustering in the context of Linked Data

By: **Miss. Amrouche Nassiba** and **Miss. Djeghdjegh Sara**

Defended on –, June, 2025, in front of the jury members:

Dr. M. Fareh	**Supervisor**	University of Blida 1
Dr. M. Mezzi	**Co-Supervisor**	University of Blida 1
Dr. –	**President**	University of Blida 1
Dr. –	**Examinator**	University of Blida 1

Acknowledgments

First of all, we would like to thank ALLAH, who gave us the strength, ability, and above all, the patience to accomplish this work.
We would like to express our sincere gratitude to Mrs. Fareh , our superviser for her unwavering support, trust, and invaluable guidance throughout this journey.
Our heartfelt thanks also go to Mrs. Mezzi, our co-supervisor, for his availability, insightful advice, and continuous support, which have greatly contributed to the success of this project.
It has been an honor to work with your team and to benefit from your expertise.
We extend our deepest gratitude to our beloved parents, who have always been there for us.
You have made countless sacrifices, sparing neither health nor effort, to support your children.
Words cannot fully express our appreciation.
You protected us, provided an education that any child would dream of, and offered a remarkable example of hard work and perseverance. We are forever indebted to the values you instilled in us values we carry with pride.
Thank you from the bottom of our hearts. We love you unconditionally.
We would also like to warmly thank our friends, who stood by us throughout this journey. Your constant encouragement, thoughtful advice, and moral support helped us overcome many challenges. Whether through discussions, moments of laughter, or simply being there, your presence has been both uplifting and motivating.
Finally, to all those we could not mention by name, we extend our sincere thanks once again. To each and every one of you—thank you!

Noussaiba and Sara

Dedication

To my beloved mother,
Your unconditional love, endless tenderness, and unwavering support have been the light guiding me through every step of my life. Without you, I would not be the person I am today. Every challenge I faced, every achievement I reached, was only possible because of your presence, your strength, and your faith in me.
I dedicate this success to you it is the fruit of your sacrifices, your prayers, and your boundless love.
From the bottom of my heart, thank you for everything. I love you.

To my dear father,
With all my love, deep appreciation, and endless gratitude, I thank you for everything you have given me, your sacrifices, and your constant belief in my potential.
You gave me everything you could, just to see me learn, grow, and succeed.You are one of the greatest reasons behind who I am today and all that I have accomplished.
I am proud to be your child, and I carry your values with me every step of the way.
Thank you for being my foundation, my guide, and my strength. I love you.

I would also like to thank my dear **brothers and my sister** for their love, support, and belief in me throughout this journey.

My heartfelt thanks to my dear **maternal cousins**, for their love, affection, and constant encouragement that always meant so much to me.

To my childhood friend **Sarah** , thank you for being there since the beginning and for always believing in me.

To my friends **Ikram** and **Yasmine** , your support, kindness, and presence made the hard moments easier and the good ones even better.

And above all, my deepest gratitude goes to my teammate and binome, **Sara**. It was truly a pleasure working with you, your collaboration, and friendship made this experience not only successful but unforgettable.

Thank you for always being there for me.

Noussaiba

Dedication

To my beloved mother and dear father,
You are my pillars of strength, the source of my courage and endless love. Your sacrifices, unwavering support, and deep faith in me have been the foundation of every step I've taken. This achievement is a reflection of your endless care and love. It is as much yours as it is mine.

After you, my deepest gratitude extends to the ones who have shared this journey of life with me since the beginning.

To my sister and brother,
Thank you for your constant encouragement and comforting presence. Your words, laughter, and love have helped me stay grounded and reminded me to smile, even during the most challenging times.

Along the way, many others have contributed to the person I have become. I am especially grateful for the role that education has played in shaping me.

To all the teachers who have guided me throughout my life,
From my earliest days in school to this very stage, each of you has shaped the person I am today. Your lessons, dedication, and belief in me have left a lasting impact, and I carry your influence with gratitude and pride.

In the context of this academic journey, I owe a special thanks to the mentors who supported me directly during this project.

To my supervisors, Dr. Fareh and Dr. Mezzi,
Your kindness, guidance, and patience have been a true blessing throughout this journey. I am deeply thankful for your valuable insights, continuous support, and the trust you placed in me. Your presence and mentorship made all the difference.

Finally, I wish to thank the person who stood by my side throughout the development of this project.

To my project partner, Noussaiba,
Thank you for being more than just a teammate. You were a true companion through it all. I will always cherish our collaboration, our shared efforts, and the genuine friendship that grew along the way.

Sara

Abstract

Knowledge graphs from the Semantic Web, often represented in RDF format, contain heterogeneous data that make their exploitation complex. Clustering these graphs enables the identification of latent semantic structures and facilitates better organization of information.
In this context, this work aims to design a deep clustering solution for heterogeneous graphs tailored to Linked Data. We propose an approach that combines semantic encoding of RDF (Resource Description Framework) graphs, representation learning using the Graph Transformer (HGT) model, and advanced deep graph clustering techniques (contrastive learning, reconstructive learning).
A variational Graph autoencoder is applied to the node attributes to enhance their quality and remove noise before passing them to the HGT in order to generate better representations.
These embeddings are then used to apply various clustering strategies (partitioning, hierarchical, spectral, and Deep Embedded Clustering).
The evaluation focuses on the quality of the clusters and their exploitation in recommendation tasks. A graphical interface has also been developed to make the system accessible.
This work highlights an innovative and comprehensive approach, combining methodological rigor with a strong focus on applicability, thereby opening promising perspectives for the analysis of complex semantic graphs.

Keywords:
Semantic Web, Knowledge Graph, Resource Description Framework (RDF), Clustering, Deep Graph Clustering, Heterogeneous Graph Transformer (HGT), Variational Autoencoder, Linked Data, Recommendation, Contrastive Learning, Reconstructive Learning.

Résumé

Les graphes de connaissances issus du Web sémantique, souvent représentés au format Cadre de description de ressources (RDF), contiennent des données hétérogènes rendant leur exploitation complexe. Le regroupement de ces graphes permet d'identifier des structures sémantiques latentes et de mieux organiser l'information.

Dans ce contexte, ce travail vise à concevoir une solution de clustering profond de graphes hétérogènes adaptée aux données liées. Nous avons proposé une approche combinant l'encodage sémantique des graphes RDF, l'extraction de représentations à l'aide du modèle Transformeur de graphe hétérogène(HGT), et des techniques avancées de Regroupement profond de graphes (apprentissage contrastif,apprentissage reconstructif).

Un Auto-Encodeur variationnel pour Graphes est appliqué au attribut pour améliorer leurs qualité et enlever le bruit avant de les passés au HGT pour produire des meilleurs représentations. Afin d'appliquer différentes stratégies de regroupement (partitionnement, hiérarchique, spectral, et Regroupement profond intégré dans l'espace latent(DEC)) sur ces représentations.

L'évaluation porte sur la qualité des clusters ainsi que sur leur exploitation dans des tâches de recommandation. Une interface graphique a également été développée afin de rendre le système accessible.

Ce travail met en lumière une approche innovante et complète, alliant rigueur méthodologique et souci d'applicabilité, ouvrant ainsi des perspectives prometteuses pour l'analyse de graphes sémantiques complexes.

Mots Clés :

Web sémantique, Graphe de Connaissances,Cadre de description de ressources (RDF), Regroupement, Regroupement profond de graphes, Transformeur de graphe hétérogène (HGT), Autoencodeur variationnel, Données liées, Recommandation, Apprentissage contrastif, Apprentissage reconstructif.

ملخص

تُعدّ رسوم المعرفة المستخرجة من الويب الدلالي، والتي غالبًا ما تُمثّل بصيغة إطار توصيف الموارد (RDF)، من بين المصادر التي تحتوي على بيانات ضخمة وغير متجانسة، مما يجعل استغلالها مهمة معقدة. يتيح تجميع هذه الرسوم (التجميع) إمكانية تحديد البُنى الدلالية الكامنة وتحسين تنظيم المعلومات.

في هذا السياق، يهدف هذا العمل إلى تصميم حل للتجميع العميق (التجميع العميق) للرسوم البيانية غير المتجانسة، والمُوجه خصيصًا نحو البيانات المرتبطة (البيانات المرتبطة). نقترح منهجية تجمع بين الترميز الدلالي لرسوم RDF، واستخلاص التمثيلات باستخدام نموذج محول الرسوم البيانية غير المتجانسة (HGT)، بالإضافة إلى تقنيات متقدمة في التجميع العميق للرسوم مثل التعلم التباييني (التعلم التباييني) والتعلم الإعادي (التعلم الإعادي).

تم تطبيق المشفّر التوليدي التباييني (المشفّر التوليدي التباييني) على سمات العقد بهدف تحسين جودتها وإزالة الضجيج قبل تمريرها إلى نموذج محول الرسوم البيانية غير المتجانسة لإنتاج تمثيلات أكثر دقة.

تم بعد ذلك استخدام هذه التمثيلات لتطبيق استراتيجيات مختلفة في التجميع، مثل التجميع التجزيئي (التجميع التجزيئي)، التجميع الهرمي (التجميع الهرمي)، التجميع الطيفي (التجميع الطيفي)، والتجميع العميق المدمج (التجميع العميق المدمج).

تركز عملية التقييم على جودة العناقيد الناتجة واستغلالها في مهام التوصية (التوصية). كما تم تطوير واجهة رسومية لتسهيل الوصول إلى النظام واستخدامه.

يسلط هذا العمل الضوء على مقاربة مبتكرة وشاملة، تجمع بين الصرامة المنهجية والاهتمام بالتطبيق العملي، مما يفتح آفاقًا واعدة لتحليل الرسوم البيانية الدلالية المعقدة.

الكلمات المفتاحية:

الويب الدلالي، رسوم المعرفة، RDF، التجميع، التجميع العميق للرسوم البيانية، HGT، المشفّر التباييني التوليدي، البيانات المرتبطة، التوصية، التعلم التباييني، التعلم الإعادي.

Contents

List of Figures

List of Tables

List of Acronyms

API Application Programming Interface. 29

ARI Adjusted Rand Index. 21

DGL Deep Graph Library. 28

DOI Digital Object Identifier. 30

GCN Graph Convolutional Network. 20

GNN Graph Neural Network. 11

HGT Heterogeneous Graph Transformer. , 81

HTTP Hypertext Transfer Protocol. 29

KGs Knowledge Graphs. 2

KL Kullback-Leibler. 16

NMI Normalized Mutual Information. 21

OWL Web Ontology Language. 7

RDF Resource Description Framework.

SPARQL SPARQL Protocol and RDF Query Language. 6

URI Uniform Resource Identifier. 6

URL Uniform Resource Locator. 30

VGAE Variational Graph AutoEncoder. 28, 81

XML Extensible Markup Language. 6

General Introduction

1. Context

The Semantic Web and Linked Data have introduced a new paradigm for representing and interconnecting data through graph-based structures. By modeling information as RDF triples, knowledge graphs enable the integration of heterogeneous data sources enriched with semantics. These graphs are increasingly used in various domains such as digital libraries, biomedical informatics, and recommender systems. As the complexity of structured semantic data increase, deep graph clustering emerges as a powerful approach to effectively analyze and uncover meaningful semantic and structural patterns within knowledge graphs.

2. Problem statement

With the advent of the Semantic Web and Linked Data, the representation of data as interconnected objects in a graph has proven to be a powerful and attractive paradigm. The graph model allows for the description of complex patterns of relationships among heterogeneous entities such as individuals, places, publications, and abstract concepts. However, efficiently processing and analyzing such rich, and diverse, remains a significant challenge, particularly in the task of discovering coherent groups or clusters of related entities.

Clustering in this context must go beyond analyzing the individual attributes of data items to also take into account their semantic and structural dependencies. The relationships between entities, encoded as edges in the graph and described through RDF triples, carry critical meaning that must be preserved to enable meaningful clustering. Traditional clustering approaches often fail in this regard, as they typically ignore the underlying graph structure and the semantics encoded in ontologies.

The core problem addressed by this thesis is the clustering of heterogeneous, semantically rich, and structurally connected data, such as that found in Linked Data and Knowledge Graphs. This type of data is often aggregated from multiple sources and represented using different vocabularies and formats, making data integration and consistent representation difficult.

Furthermore, incompleteness, noise, and heterogeneity are common characteristics of real-world

KGs, which further complicate the process of learning reliable entity representations.
Another key challenge is the lack of ground-truth labels, which makes unsupervised clustering difficult to validate.

3. Work objectives

These challenges necessitate a solution that integrates semantic embedding techniques with deep learning architectures specifically designed for graph data, to effectively capture both the semantic richness and the topological complexity of knowledge graphs, ultimately enabling more accurate and meaningful clustering within Linked Data environments. To address these challenges, we propose in this project deep graph clustering as a viable and promising solution. Unlike traditional clustering techniques, deep graph clustering leverages the intrinsic graph structure by preserving both local and global interaction patterns among entities, resulting in clusters that are not only structurally coherent but also semantically valid.
The objectives of our project are to:

- Explore embedding techniques that simultaneously integrate structural features and semantic information.
- Project seeks to introduce semantics into the clustering process, thereby improving the accuracy and relevance of the resulting clusters.
- Propose a detailed system pipeline adapted to the Semantic Web.
- Use the solution developed in this thesis to construct a recommendation system in the bibliographic domain.

4. Thesis organization

The thesis is structured as follows:

- Chapter 1: Deep graph clustering for Linked data
  Introduces the concepts of semantic web, linked data, and graph-based learning, and describes the motivation behind clustering in this context. Subsequently a summary of comprehensive review of existing deep clustering algorithms, with a specific focus on their relevance to linked and graph-structured data.
- Chapter 2: Deep Graph Clustering System Design
  In this chapter, we begin by examining the characteristics of our solution. Then, we describe the overall system architecture in detail, explaining its main components.

- Chapter 3: Implementation and Test
  This chapter presents the implementation of the proposed solution. We describe the various tools used for the development of our system, then discuss in detail the complete testing procedure, analyze the results of the clustering as well as their interpretation, and finally, we present the interface and functionalities of the developed application.

  This thesis will conclude with a general conclusion that summarizes the main points discussed and offers some perspectives for improvement of our work.

Chapter 1

Deep graph Clustering for Linked Data

1.1 Introduction

The World Wide Web has significantly transformed communication, information dissemination, and business, evolving towards the Semantic Web to enhance accessibility and usability.
Since 2006, the emergence of intelligent web systems has enabled enriched and contextualized knowledge representation through linked data. In this context, graph data structures have gained prominence as they model complex relationships between entities, facilitating tasks like semantic integration and ontology alignment.
Deep graph clustering extends these capabilities by leveraging advanced representation learning to group entities based on both structural and semantic properties. This approach not only improves clustering accuracy but also enables scalable analysis of linked data, crucial for applications in knowledge discovery and the semantic web[19].
This chapter is structured to provide a comprehensive foundation for understanding the role of deep graph clustering in the context of linked data.
First, we introduce the World Wide Web. Then we focus on the Semantic Web, exploring its architecture. After that, we present the linked Data. In the second part, we introduce the deep graph clustering methods, exploring learning paradigms.
Then we compare some related works, followed by a discussion and analysis that synthesizes the key insights of the chapter.

1.2 World Wide Web

The Web, or World Wide Web, is a public hypertext system operating on the Internet, allowing users to access pages (web pages) hosted on various sites via a browser. It relies on hyperlinks between pages, creating a structure similar to a spider's web. The Web was initially developed by Tim Berners-Lee at CERN [4] in 1989.
Building upon the foundations of the World Wide Web, the Semantic Web represents its next

evolutionary step. While the traditional Web focuses on linking documents, the Semantic Web aims to link data in a structured and meaningful way.

1.3 Semantic Web

The Semantic Web is an evolving extension of the World Wide Web in which the semantics of information and services on the web are defined, making it possible for the web to understand and satisfy the requests of people and machines to use the Web Content. It derives from W3C director Tim Berner - Lee vision of the Web as a universal medium for data, information and knowledge exchange[3].

1.3.1 Semantic Web Architecture

The architecture of the Semantic Web, represented by the "Semantic Web Stack" or "Semantic Web layer cake" , is often illustrated as a pyramid, where each layer builds on the capabilities of the layer below it. This stack evolves continuously as new technologies and standards emerge. Below is an explanation of the key layers in this architecture:

- **Uniform Resource Identifier (URI)/IRI (Uniform Resource Identifier /International-ized Resource Identifier) :** This layer provides unique identifiers for resources on the web. URI is used for identifying web resources, while IRI extends this capability to support a wider range of characters, allowing resources to be identified in various languages.

- **Extensible Markup Language (XML) (Extensible Markup Language) :** XML is a flexible, extensible language used to define and structure data. While it does not inherently communicate the meaning of the data, XML forms the base for structuring information that higher layers can build on.

- **Resource Description Framework :** RDF is a framework for describing resources on the web in a machine-readable format. It enables data to be structured as triples (subject, predicate, and object), creating a graph model of data relationships. RDF is crucial in making data interoperable and understandable by machines.

- **Resource Description Framework Schema :** RDFS extends RDF by providing basic elements for the description of ontologies or data schemas. It allows for defining relationships between resources and specifying the types of resources, enhancing data semantics.

- **SPARQL Protocol and RDF Query Language (SPARQL) "SPARQL Protocol and RDF Query Language" :** SPARQL is a powerful query language for retrieving and manipulating data stored in RDF format. It enables querying across multiple RDF datasets, making it the primary tool for extracting meaningful data from the Semantic Web.

- **Web Ontology Language (OWL) "Web Ontology Language" :** OWL is a language used for representing rich and complex knowledge about things, groups of things, and the relationships between them. It enhances the expressiveness of RDF/RDFS, allowing for the creation of ontologies with detailed constraints and relationships.

- **Unifying Logic :** This layer provides the logical foundations for ensuring interoperability between different systems that use Semantic Web standards. It helps combine data from diverse sources and ensures consistency in reasoning.

- **Proof :** The Proof layer ensures that the data and the derived conclusions are trustworthy. It involves validating that the logical steps and reasoning processes are correct and reliable, which is critical for maintaining data security and accuracy.

- **Crypto "Cryptography" :** This layer addresses the need for security in the Semantic Web, providing mechanisms for ensuring data integrity and privacy through encryption techniques.

- **Trust :** The Trust layer ensures the credibility of information by verifying the source, security, and authenticity of the data. It plays a crucial role in making the Semantic Web safe for data exchange and collaboration.

This architecture provides a hierarchical framework for how data is structured, queried, reasoned, and secured in the Semantic Web, allowing both humans and machines to understand and interact with web-based knowledge more effectively[5].

This architecure is represented in Figure 1.1 :

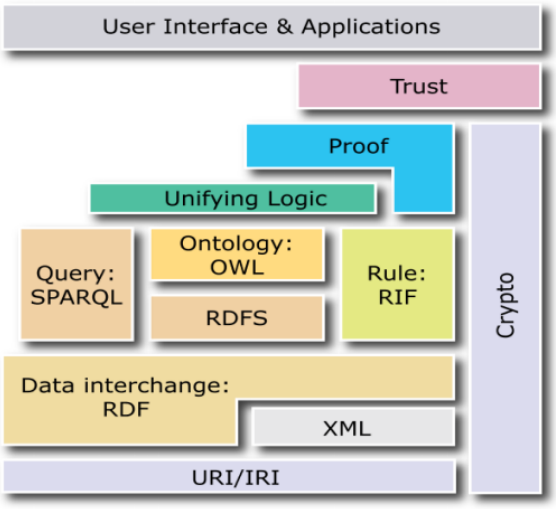

Figure 1.1: Semantic web layer cake [1]

1.3.2 Applications

Semantic Web is applied in many fields and has many uses, including in healthcare, e-commerce, and scientific research.

- Healthcare : Semantic Web can be used to integrate data from various sources so that physicians can provide more accurate diagnoses, and treatments. some examples include :
  - SNOMED CT[1] : is a clinical terminology with wide clinical expression covering the entire clinical spectrum, and supports clinicians in enhancing the accuracy and consistency of data entered by them.
- In e-commerce: The Semantic Web improves recommendation precision by using user information and metadata to provide personalized recommendations based on browsing history and affinity. among the examples are :
  - Amazon: Amazon (Amazon.com) is the world's largest online retailer and a prominent cloud service provider.
- In scientific research : the Semantic Web can be used to integrate data from different experiments and studies, making it easier to identify patterns and draw conclusions. among the examples are :
  - Dbpedia : is a crowd-sourced community effort to extract structured content from the information created in various Wikimedia projects. The data is served as Linked Data, which is revolutionizing the way applications interact with the Web.
- In social networking we find : Facebook, Intagram, LinkedIn, Twitter and others, which are websites and apps that allow users and organizations to connect, communicate, share information and form relationships.

1.3.3 Benefits of The Semantic Web

The Semantic Web offers a range of benefits that enhance data accessibility, integration, and accuracy [13]:

- Data Integration: By adding metadata to web pages, computers can understand how different pieces of data relate to each other. This can be particularly useful in scientific research, where data from different sources must be integrated to make meaningful conclusions.
- Language Independence: Information can be accessed and understood across multiple languages, breaking down linguistic barriers.

[1] `https://www.snomed.org/`

- Enhanced Data Accuracy: The Semantic Web allows for more precise data representation, leading to relevant and accurate information.
- Data-Driven Processes: Seamlessly linking data to processes and procedures facilitates automation and informed decision-making.

1.4 Linked data

Linked Data is all about interlinking data on the Web from different sources in a meaningful way. Instead of connecting documents as usual web pages, it connects actual entities such as people, places, or objects using structured information. This makes systems understand and work together with each other's data, even in different locations or organizations [2].
Such unification of data from throughout the Web is a stepping stone towards what is known as the Web of Data, a Web in which data, and not documents, is the base unit of information exchange. Under this vision, the Web evolves from a document-based platform to an decentralized, semantic, machine-understandable network of data.
Linked Data supports this vision through the provision of:

- Use of URIs to point to entities,
- Common data models like RDF and vocabularies (ontologies) for expressing meanings.[6]

Figure 1.2 depicts an example of linked data graph :

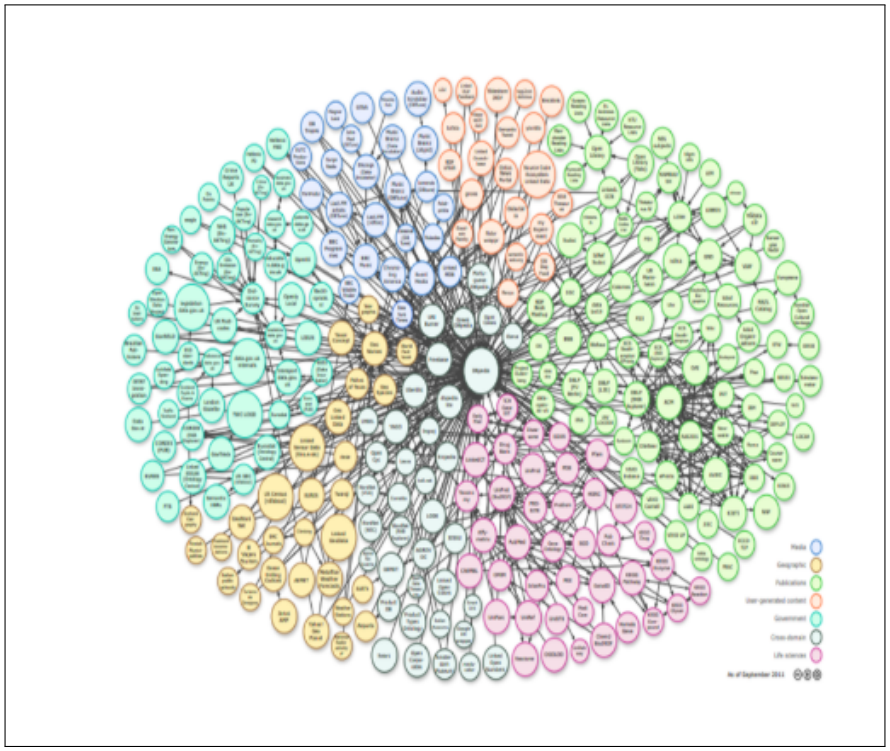

Figure 1.2: Example of linked data graph [2]

1.4.1 Linked data Datasets

Linked data datasets provide structured, interlinked, and machine-readable data that enable advanced reasoning and semantic querying over large-scale knowledge graphs. Below are some of the most widely used datasets in the Linked Data and Semantic Web communities:

- Dbpedia : DBpedia is a crowd-sourced initiative that extracts structured content from various Wikimedia projects, creating an open knowledge graph (OKG) accessible on the Web. This data is provided as Linked Data, transforming how applications interact with online information.

- Wikidata : Wikidata is a free and open knowledge base that can be accessed and edited by both humans and machines. It serves as the central repository for structured data used by various Wikimedia projects, including Wikipedia, Wikivoyage, Wiktionary, and Wikisource.

- YAGO (Yet Another Great Ontology) : is a large knowledge base with general knowledge about people, cities, countries, movies, and organizations. there is YAGO3, YAGO4 .

- Neo4j : High-speed graph database with unbounded scale, security, and data integrity for mission-critical intelligent applications.

1.4.2 Benefits of linked data

The use of Linked Data offers a number of significant benefits[14]:

- Uniformity: For consistency and interoperability, all interconnected content utilizes globally unique URIs based on the RDF triple model (subject–predicate–object).

- De-referencability: Similar to URLs, URIs can be utilized to find and retrieve entity descriptions directly from the Internet.

- Integrability: Since all data shares the same structure, combining datasets is straightforward. Deeper semantic integration is possible through ontology alignment and schema matching.

Faced with the increasing complexity of graphs derived from Linked Data, traditional processing methods are reaching their limits, thus paving the way for more powerful approaches such as deep graph clustering, capable of exploiting both the topological structure and semantic attributes to reveal hidden communities within graphs.

1.5 Deep graph clustering methods

Deep graph clustering is a method that groups data by taking advantage of both node attributes and their relationships within a graph, using deep learning mechanisms to discover complex and useful representations [19].
It relies on graph data models that represent data as nodes (entities) and edges (relations), facilitating the storage, processing, and analysis of highly interconnected data. These models are extremely effective at modeling the structure of complex systems such as social networks, and knowledge graphs [12].
Among the many analysis tasks performed on graph-structured data, clustering plays a fundamental role in revealing unknown patterns and communities within the graph structure.
Clustering, or cluster analysis, is the process of segregating data into distinct classes such that objects with similar attributes or features are grouped together. It is considered an unsupervised learning technique that attempts to find useful patterns in unlabeled data.
Graph clustering is a specialized form of clustering for graph-structured data, in which the goal is to cluster nodes or subgraphs based on structural or attribute similarity [12].

1.5.1 Notation and Definition of deep graph clustering

It is a fundamental and challenging task to separate the nodes into different groups in an unsupervised manner. Benefiting from the strong graph representation capability of deep Graph Neural Network (GNN), which are a family of neural architectures that learn node representations by recursively aggregating and transforming information from neighboring nodes and their features. This mechanism captures both the graph topology and node attribute information, making GNNs particularly effective for representation learning in graphs [11].

Consider an attribute graph $G_A = (V, E, X)$ as the example input of deep graph clustering, where:

- $V = \{v_1, v_2, \ldots, v_N\}$ is the node set of N nodes belonging to K classes,
- E is a set of edges, and
- $X \in \mathbb{R}^{N \times D}$ is the node attribute matrix.

Additionally, in matrix form, G_A is represented by:

- the adjacency matrix $A \in \mathbb{R}^{N \times N}$, and
- the attribute matrix $X \in \mathbb{R}^{N \times D}$.

As shown in Figure 1.3, the target of deep graph clustering is to encode nodes using neural networks and assign them into different clusters. Specifically, the self-supervised neural network

F first encodes the nodes in the attribute graph G_A as follows:

$$Z = F(G_A) = F(X, A).$$

Here, Z represents the latent representations or *embeddings* of the nodes, where each row z_i corresponds to a low-dimensional representation of node v_i. These embeddings capture the structural and attribute-based context of each node within the graph.

The self-supervised neural network F is trained with pre-text tasks, such as reconstruction, contrast, etc. After encoding, the clustering method C groups the nodes into several disjoint clusters as follows:

$$\Phi = C(Z, K),$$

where K denotes the number of clusters and $\Phi \in \mathbb{R}^{N \times K}$ is the clustering assignment matrix.

1.5.2 Learning paradigm

The demonstration of the general pipeline of deep graph clustering. As shown in Figure 1.3, the encoding neural network F is trained in a self-supervised manner to embed the nodes into the latent space. After encoding, the clustering method C separates the node embeddings Z into several disjoint clusters.

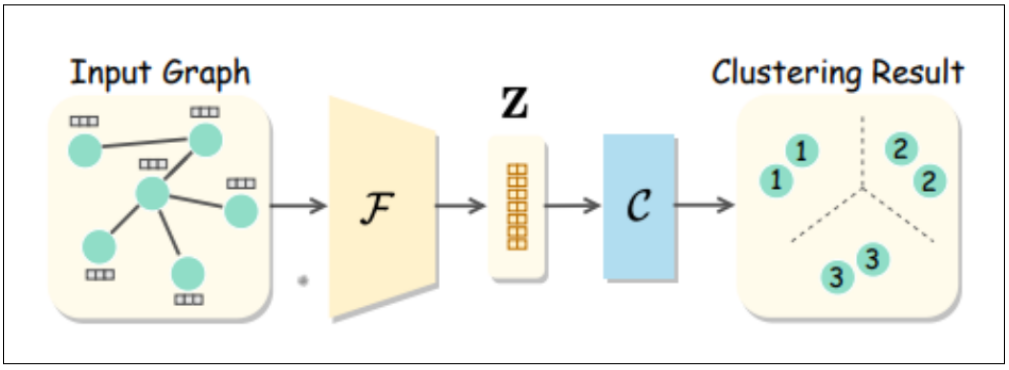

Figure 1.3: Deep graph clustering general pipeline[11]

Based on the learning paradigm, the surveyed methods fall into four categories: reconstructive methods, adversarial methods, contrastive methods, and hybrid methods. Taking the graph as input, the learning paradigms of deep graph clustering methods are described as follows:

1. **Reconstructive Methods :** The core idea of the reconstructive methods is forcing the network to encode the features in the graph and then recovering (part of) the graph information from the learned embeddings. Thus, the reconstructive methods focus on the intra-data information in the graph and adopt the node attributes and graph structure as the self-supervision signals.

2. **Adversarial Methods :** The adversarial deep graph clustering methods aim to improve the quality of features by creating a zero-sum game between the generator and the discriminator. Specifically, the discriminator is trained to recognize whether learned features are from the real data distribution, while the generator aims to generate confusing embeddings to cheat the discriminator. In these settings, the generation and discrimination operations provide effective selfsupervision signals, guiding the network to learn high-quality embeddings.

3. **Contrastive Methods :** The critical idea in contrastive deep graph clustering methods is to improve the discriminativeness of features by pulling together the positive samples while pushing away the negative ones. Thus, contrastive methods focus on the inter-data information and construct the self-supervision signals via meaningful relationships between samples.

4. **Hybrid Methods :** In recent years, some papers have demonstrated the effectiveness of combining different learning paradigms. In addition, researchers also verified the effectiveness of the combination of reconstructive and contrastive learning paradigms [11].

1.5.3 Clustering method

In deep graph clustering, the aim is to group nodes into clusters without the need for labeled data. There are two main approaches: traditional clustering methods and neural-based clustering methods.

The table 1.1 presents a comparative table highlighting traditional clustering methods versus neural clustering methods:

Aspect	**Traditional Methods (e.g., K-means)**	**Neural Clustering Methods**
Approach	Group nodes based on static embeddings (numerical representations).	Integrate clustering directly into neural networks for adaptive learning.
Flexibility	Limited: Embeddings are not adjusted during clustering.	High: Node representations and clustering are adjusted simultaneously.
Efficiency	Quick for small datasets but less effective for complex graphs.	More precise and efficient, especially for complex datasets.
Scalability	Struggles with large graphs; does not process data in batches.	Better scalability using batch processing techniques.
Suitability for Graphs	Limited adaptability to graph structure.	Tailored for graph data, leveraging deep learning's representation power.

Table 1.1: Comparison of Traditional Clustering Methods versus Neural Clustering Methods

1.5.4 Challenge and Opportunity

Deep graph clustering, while promising, faces several inherent challenges.Below are some of the key issues encountered in practice [11]:

Graph Data Quality: Many techniques assume the graph data is clean and accurate, but in reality, real-world graphs often contain noise or errors, with incomplete or wrong information about the nodes and connections.

Unknown Number of Clusters: A lot of methods expect us to know exactly how many clusters we need from the get-go. But in real life, that's rarely the case—we often don't have that information to begin with, which makes everything a lot trickier.

Discriminative Ability: To create distinct groups, it's essential to have clear, unique representations of the nodes. A common issue is when everything collapses into a single cluster, which undermines the usefulness of the analysis.

1.5.5 Applications of Deep Graph Clustering

The versatility of deep graph clustering has led to its adoption across a wide range of domains. The following list highlights some notable application areas[11]:

Natural Language Processing: Useful in document clustering, topic mining, and speech separation.

Computer Vision: Applied in face analysis, saliency detection, and object recognition.

Bioinformatics: Employed for metagenomic binning, molecular structure analysis, and single-cell RNA sequencing.

Social Network Analysis: Enables community detection, anomaly detection, and the study of social structures.

Recommendation Systems: Facilitates user grouping, extraction of user interests, and personalized recommendations.

Medical Science: Used in disease analysis, processing and analyzing medical big data, and medical image analysis.

1.6 Related Works

Deep graph clustering techniques are increasingly applied to graph-structured data, where entities and relationships form complex networks.
These techniques leverage the interplay between graph topology and node attributes to improve tasks such as entity clustering, relationship discovery, and semantic enrichment of knowledge

graphs.
In this section, we present related works in deep graph clustering, highlighting their methodologies and performance:

1. **L. Sun et al. (2024): Lorentz Structural Entropy Neural Network for Deep Graph Clustering (Lsenet) [16]**

   - **Problem and Objectives**

     LSEnet tries to address the graph clustering problem without prior knowledge of the number of clusters, in contrast to current approaches.
     It introduces the Differentiable Structural Information framework, which integrates deep learning with graph information theory. The model considers node attributes and graph structure, and applies hyperbolic geometry to encode complex interactions better. This adaptable approach improves the efficiency and usefulness of graph clustering.

   - **Key Innovations**

     LSEnet constructs an ideal partition tree revealing cluster organization and embeds the nodes into hyperboloid space by convolutions on manifolds. This results in efficient unsupervised clustering without knowing the number of groups a priori.

2. **B, Wu et al. (2024): Synergistic Deep Graph Clustering Network (SynC) [18]**

   - **Problem and Objectives**

     SynC takes up the task of the limitations of current deep graph clustering approaches, including their inability to incorporate learning of representations and enrichment of graph structure effectively. This may lead to collapse of representations, particularly in sparsely homogenuous graphs or classes with imbalanced classes. To address this, SynC proposes a self-supervised autoencoder, which optimizes the embeddings by taking into account structural information. Its objectives are: to improve the quality of these node representations, to achieve state-of-the-art clustering performance, to create a synergy between learning and structure, and to ensure computational efficiency and model simplicity.

   - **Key Innovations**

     SynC combines representation learning and structural adjustment by sharing weights, which improves the consistency of embeddings while maintaining low model complexity.

3. **Liang, H. & Gao, J. (2021) : Wasserstein Adversarially Regularized Graph Autoencoder (WARGA) [9]**

   - **Problem and Objectives**

     WARGA propose a way to improve node representation embedding learning in graphs through the use of Wasserstein distance as regularization rather than Kullback-Leibler (KL) divergence, which performs worse when distributions intersect. This enables them to achieve more aligned and stable embeddings, which are useful for link prediction and node clustering, while adding an adversarial learning element to enhance the quality of learned representations.

   - **Key Innovations**

     By regularizing embeddings employing the Wasserstein distance, WARGA improves graph representation learning and improves the model's resilience and performance for link prediction and clustering.

.

4. **Yu, Li et al. (2024): Contrastive deep nonnegative matrix factorization for community detection(CDNMF) [8]**

   - **Problem and Objectives**

     Classical methods like Non-negative Matrix Factorization (NMF) often fail to capture the complexity of real-world networks because they focus primarily on network topology and neglect node attributes.
     CDNMF aims to improve this framework by deepening it and integrating contrastive learning to utilize both the topology and attributes of nodes, in order to improve the detection of community relationships and make community detection more accurate and efficient.

   - **Key Innovations**

     CDNMF enhances community detection by combining deep and contrastive learning, integrating node topology and attributes, and uses unbiased negative sampling to better learn community relationships, thereby outperforming current methods.

5. **M, Shi et al. (2020) : Multi-Class Imbalanced Graph Convolutional Network Learning (DR-GCN)[15]**

   - **Problem and Objectives**

The work addresses the challenge of learning from graphs with imbalanced class distributions, where certain classes of nodes are underrepresented. The main objective is to improve the representation of nodes in a semantic space while considering these imbalances, in order to optimize model performance on often neglected minority classes.

- **Key Innovations**

  The key innovations of DR-GCN include the introduction of a conditional adversarial training process that enhances the separation of node representations across different classes, as well as a distribution alignment mechanism that balances learning between majority and minority classes.

6. Chu, Wang et al. (2019): Deep Attentional Embedded Graph Clustering (DAEGC)[17]

- **Problem and Objectives**

  The work addresses the challenge of clustering attributed graphs by effectively integrating both structural and content information.

  The main objective is to develop a unified framework that simultaneously learns graph embeddings and performs clustering, overcoming the limitations of traditional two-step methods where the learned embeddings may not be optimal for clustering tasks.

- **Key Innovations**

  The key innovations of DAEGC include the use of a graph attentional autoencoder that captures the importance of neighboring nodes, enhancing the quality of latent representations.

  Additionally, a self-training module is introduced to generate soft labels from confident clustering assignments, guiding the optimization process.

7. Y, Lui et al. (2021): Deep Graph Clustering via Dual Correlation Reduction (DCRN)[10]

- **Problem and Objectives**

  DCRN solves the issue of representation collapse for deep graph clustering, which decreases the discriminative power of nodes. It aims to enhance clustering on attributed graphs by alleviating information redundancy at both the sample and feature levels, via self-supervised learning methods.

- **Key Innovations**

  The DCRN model introduces a dual correlation reduction mechanism to eliminate

redundancy, a Siamese framework to contrast representations between views of the graph, and long-distance propagation regularization to mitigate over-smoothing in GCNs.

8. **Q, Li et al. (2024) :Incorporating Higher-order Structural Information for Graph Clustering (HeroGCN)[7]**

   - **Problem and Objectives**
     The work addresses the challenge of effectively capturing higher-order structural information in graph-structured data for clustering tasks. Existing methods often overlook the importance of this information, leading to suboptimal clustering performance.

     The main objective is to develop a model that integrates both graph structure and node attributes, thereby enhancing clustering outcomes by leveraging the rich relationships present in the data.

   - **Key Innovations**
     HeroGCN enhances clustering by combining higher-order structure and node attributes. It integrates mutual infomax, modularity supervision, an AGCN layer, and adaptive fusion for better representations.

1.6.1 Comparison of the works :

Based on the works presented, we performed a comparison using the following tables.
The first comparison is presented in Table 1.2 and focuses on the following criteria:

- **Dataset :** the collection of data used to evaluate or train the model.

- **Learning Paradigm** Refers to the approach used to train the model, which could be:
  - Contrastive
  - Reconstructive
  - Adversarial
  - Hybrid

- **Clustering Method** The algorithm or technique used to group nodes or entities into clusters. Examples include:
  - Neural Clustering
  - Traditional Clustering

Work	Dataset	Learning paradigm	Clustering Methods
L, Sun et al. (2024): LSENET	-Cora -Citeseer -Amazon-photo -Computer Dataset -Football -Kerate -AMAP	Contrastive	Neural clustering
B, Wu et al. (2024) : SynC	-ACM -Citeseer -DBLP -AMAP -CORA -UAT -Wisconsin -Texas	Reconstructive	Neural clustering
Liang, H. And Gao, J. (2021) : WARGA	-Cora -Citeseer -PubMed	Reconstructive + Adversarial	Traditional clustering(kmeans)
Yu, Li et al. (2024) : CDNMF	-Cora -Citeseer -PubMed	Contrastive	Neural clustering
M, Shi et al. (2020): DR-GCN	-Cora -Citeseer -PubMed -DBLP	Adversarial	Traditional clustering(kmeans)
hu, Wang et al. (2019) : DAEGC	-Cora -Citeseer -PubMed	Reconstructive	Neural clustering
Y, Liu et al. (2021) : DCRN	-ACM -Citeseer -PubMed -DBLP -AMAP -CARFULL	Contrastive	Traditional clustering (Kmeans)

Q, Li et al. (2024) : HeroGCN	-ACM -Citeseer -DBLP -USPS -HHAR	Reconstructive	Neural clustering

Table 1.2: Comparison of related works in terms of Datasets, Learning paradigm , and Clustering method

The second part of the comparison is presented in Table 1.3 and focuses on the following criteria:

- **Graph Type**
  Indicates the nature of the graph used in the study, such as:
  - Attribute Graph
  - Heterogeneous Graph
  - Dynamic Graph
  - Pure Structure Graph
- **Network Architecture**
  Describes the structural design of the model used for processing the graph. Examples include:
  - Graph Neural Network (GNN):
    - Graph Convolutional Network (GCN): A neural network that processes graph-structured data by aggregating information from neighboring nodes. It is used for tasks such as node classification, link prediction, and clustering.
    - Transform Input Graph Auto-Encoder (TIGAE): An architecture that enhances graph convolutional representations using a linear transformation with bias, improving node embeddings, and mitigating representation collapse.
  - Deep Nonnegative Matrix Factorization (DNMF): A deep learning approach that extends traditional NMF to learn hierarchical representations from graph data.
  - Autoencoders: A type of neural network trained to encode input data into a lower-dimensional representation and reconstruct it, useful for feature learning and anomaly detection.
  - Autoencoders with attention mechanism: An extension of autoencoders that integrates attention mechanisms to focus on important features, improving representation learning.
  - Siamese Network: A neural network architecture with two identical subnetworks that learn to compare input pairs, commonly used for similarity learning and verification tasks.
- **Metrics results:** The measures used to evaluate model performance. Common metrics include:

- **Accuracy (ACC):**Proportion of correctly clustered nodes.

- **Normalized Mutual Information (NMI):** Evaluates the similarity between predicted and ground truth clusters.

- **Adjusted Rand Index (ARI):** Measures the agreement between the clustering results and the true partitioning.

- **F1-Score:** Balances precision and recall in evaluating clustering outcomes.

For the metric values mentioned in the table, we chose the Citeseer dataset.

- **Limitations :** The constraints or weaknesses of each work.

Work	Graph type	Network Architecture	Metrics results	Limitation
L. Sun et al. (2024): LSENET	Attribute graph	GCN	NMI : 49.35±0.20 ARI : 46.91±1.12	-Dependence on Node Attributes. -Computational Complexity. -Limitation to Static Graphs.
B, Wu et al. (2024) : SynC	Attribute graph	TIGAE	ACC : 71.77±0.27 NMI : 46.37±0.42 ARI : 48.09±0.45 F1 : 65.72±0.36	-Computational Complexity. -Representation Collapse.
Liang, H. And Gao, J. (2021) : WARGA	Attribute graph	GCN	ACC : 56.2 ± 0.03 NMI : 30.1 ± 0.02 ARI : 28.5 ± 0.02	-Dependence on Hyperparameter Settings. -Performance on Large Datasets.
Yu, Li et al. (2024) : CDNMF	Heterogeneous graph	DNMF	ACC : 47.56 NMI : 25.59	-Computational Resources. -Sensitivity to Hyperparameters.

M, Shi et al. (2020) : DR-GCN	Attribute graph	GCN	ACC 64.19 NMI 38.24 ARI 38.24 F-score 56.48	-Dependence on the quality of input data. -Its effectiveness on sparse or complex graphs remains uncertain.
Chu, Wang et al. (2019): DAEGC	Attribute graph	Autoencoders with attention mechanism	ACC : 67.1 NMI : 26.6 ARI : 27.8 F-score : 65.9	-High computational complexity on very large graphs. -Reliance on the quality and quantity of input data.
Y, Lui et al. (2021) : DCRN	Attribute graph	Siamese Network	ACC 70.88±0.19 NMI 45.92±0.35 ARI 47.73±0.29 F-score 65.79±0.2	-Sensitivity to hyper-parameters. -Potential scalability issues with very large graphs. -Dependence on the quality of input data.
Q, Li et al. (2024) : HeroGCN	Attribute graph	GCN + Autoencoders	ACC 70.33 NMI 43.05 ARI 45.57 F-score 62.41	-Sensitivity to hyperparame-ters. -Dependence on data quality.

Table 1.3: Comparison of different works in terms of Graphe type, Network Architecture, Metrics results, and Limitations

1.6.2 Analysis and Discussion

Most studies focus on attribute graphs, except for CDNMF, which utilizes a heterogeneous graph, thus confirming the importance of nodes with exploitable characteristics.

GCNs dominate the architectures, although some works employ an autoencoder, and there are works that combine an Autoencoder with other architectures , such as DAEGC which introduces an Autoencoder with attention mechanism and HeroGCN which uses a GCN with an Autoencoder. The integration of Autoencoders in some works highlights a trend towards learning latent representations to enhance clustering.

Among the different learning paradigms in deep graph clustering, contrastive and reconstructive methods are the most widely used. Reconstructive methods are frequently applied alone or with other learning paradigms such as adversarial and contrastive learning. Adversarial learning, while not as widely adopted, is present in models like WARGA and DR-GCN.

Neural clustering is where it's headed, but the appearance of traditional methods in WARGA, DCRN, and DR-GCN suggests that traditional methods may still hold value.

A more in-depth analysis of existing approaches reveals several common pitfalls. High sensitivity to hyperparameter values such as WARGA, HeroGCN, or DCRN, or a strong dependence on the quality of the input data is many models' Achilles' heel. Others suffer from scalability issues, with costly computational complexity on large graphs such as DAEGC or CDNMF. Representation collapse is a similar issue to SynC that is already known, as well as the exclusive reliance on node attributes in certain models like LSENET, which limits their performance on highly dynamic or sparsely labeled graphs.

1.7 Conclusion

This chapter delved into the foundational concepts and advancements in the Semantic Web, highlighting its architecture and potential to enhance the World Wide Web. We explored the Linked Data, focusing on their role in structuring, sharing, and contextualizing knowledge to improve interoperability and accessibility.
The discussion then transitioned to deep graph clustering methods, introducing graph data models and clustering analysis, and methodologies.
By reviewing and comparing related works, we identified significant advancements and limitations, offering insights into the potential of deep graph clustering for linked data. This analysis lays a robust groundwork for future exploration and application in subsequent chapters.
In the following chapter, we will present our deep graph clustering approach.

Chapter 2

Deep Graph Clustering System Design

2.1 Introduction

The Deep Graph Clustering techniques discussed in the previous chapter have advanced the state of the art for processing structured data. Yet, many continue to be subject to limitations when dealing with complicated and varied graph structures, such as those typically found within Linked Data and the Semantic Web.
To address these challenges, we propose our work that enhance representation learning and clustering robustness by integrating more adaptable learning paradigms.
In this chapter, we present the overall objectives of our research work, give an overview of the system pipeline, describe the critical components and algorithms used.

2.2 Improvements over the related works

Our proposed model addresses some key enhancements to the primary limitations of existing deep graph clustering methods applied on knowledge graphs. In contrast to the majority of prior efforts that only deal with attribute graphs, our framework naturally supports heterogeneous graphs, leveraging the richness of multi-typed nodes and relationships, an aspect still unvisited apart from models.

Apart from this, unlike the traditional GCN-based models, which are plagued by representation collapse and rigidity, we involve a combination model of Heterogeneous Graph Transformers and a variational autoencoder to enable more expressive and richer embeddings.
This combination allows for more powerful, expressive, and more flexible embeddings suitable for real-world heterogeneous graphs.
By combining heterogeneous modeling, powerful representation learning, our approach overcomes most of the structural and algorithmic restrictions of the related works studied.

2.3 Overview of the Proposed Pipeline

Our approach is based on Deep Graph Clustering as the core foundation, into which we integrate several modern components to effectively capture the structural and semantic complexities of the graph.

We first get the node embeddings through a Variational Graph Autoencoder, which improves the quality and expressiveness of the encoded node features in a compact, low-dimensional space.

Then, we further refine the embeddings through a Heterogeneous Graph Transformer, trained with a contrastive loss function.

Once the last embeddings produced by the HGT are done, various clustering algorithms are applied. The system is made up of a number of interrelated steps, each of which is essential to turning unstructured graph data into meaningful clusters.

To illustrate and clarify the entire process, we have designed an overall diagram of our system, which is shown in Figure 2.1, summarizing the key steps of our approach. This diagram provides a visual representation of the methodical progression, from data preparation to the final clustering results on the heterogeneous graph. It thus offers a detailed overview of our system.

Our work is composed of four main phases. Each phase consists of several steps, as shown in the diagram :

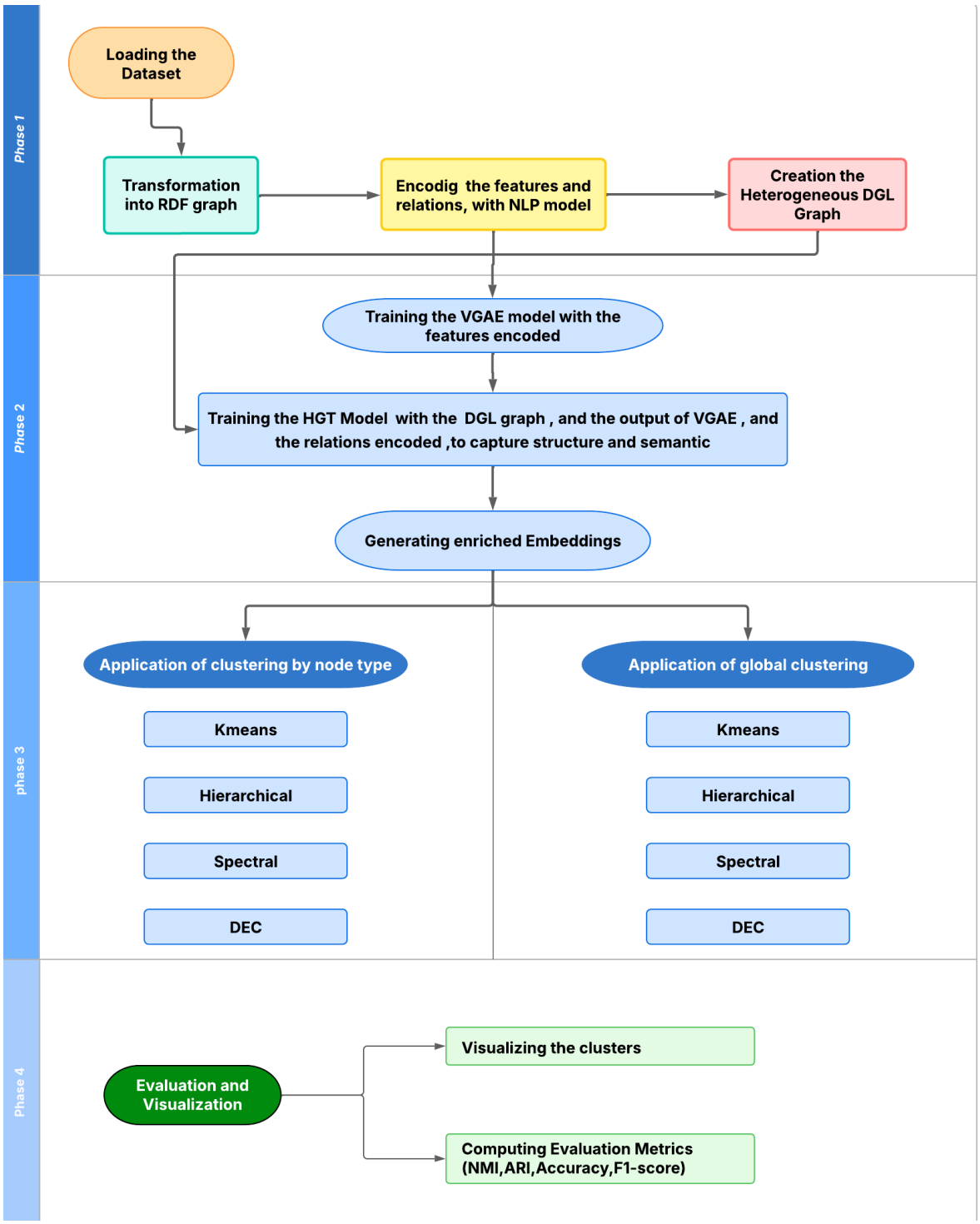

Figure 2.1: Pipeline of our system

Phase 1: Data preparation

First, We began the process by importing the dataset. But our interest was not in utilizing the whole dataset. Rather, we picked and extracted a certain subset of useful nodes, including publication, author, venue, and domain, along with their fundamental attributes.
Then we performed a step of data cleaning and organization.
Thereafter, we applied semantic encoding techniques to improve the representation of the inherent meaning contained in the data.
Lastly, we converted the resultant cleaned and enriched data into a heterogeneous DGL graph in line with the models employed.
Thus, this phase includes four main steps:

- Step 1: Loading the dataset.
- Step 2: Transformation of the data set to RDF, and structuring in RDF graph.
- Step 3: Encoding node attributes and relationships.
- Step 4: Creation of the heterogeneous DGL graph.

Phase 2: Extracting embeddings

After performing data preprocessing and cleaning, we applied a VGAE to the node attributes (features) in order to reduce the dimensionality of the data and eliminate noise, thereby improving the quality of the feature embeddings.
Next, we used the Heterogeneous Graph Transformer model to enrich the embeddings by incorporating the graph structure, semantically encoded relations, and the DGL heterogeneous graph.
This phase consists of two main steps:

- Step 1: Applying the VGAE to the features to enhance their quality
- Step 2: Training the HGT model to extract the final enriched embeddings

Phase 3: Clustering

The primary goal of this module is to categorize associated data into groups with similar characteristics, in an unsupervised setting where no ground truth are available. The objective is to identify meaningful clusters.
For this, we used two approaches: node-type-based clustering and global clustering. For each method, we applied several clustering techniques.

Phase 4: Evaluation and visualization

After applying one algorithm from each clustering category (K-means ,Hierarchical , , etc.) to the embeddings, we now reach a crucial phase aimed at evaluating and validating the quality of the resulting clusters.

2.4 Description of the Dataset

The dataset used in this work is derived from DBLP (Digital Bibliography Library Project), a comprehensive and widely used bibliographic database in the field of computer science.
DBLP provides structured metadata on scientific publications, covering a broad range of topics in computer science and related disciplines.
It contains different types of concepts such as authors, publications, venues (conferences or journals), and research topics.
This dataset offers a high-quality source of information for analyzing scholarly communication networks, author collaborations, venue impact, and research trends.
The raw data from DBLP is available in XML format, which provides a structured hierarchy that can be parsed and transformed into other formats such as RDF or graph structures for advanced processing and machine learning tasks.
DBLP is maintained by the University of Trier, Germany, and is freely accessible for academic use.[1]
In the following, we will describe the details of each phase of our system.

2.5 Phase 1 : Data preparation

The preliminary step of the phase aims at preparing the data before generating the embeddings.

2.5.1 Step 1: Loading the dataset

The dataset is built by issuing a query to the official DBLP API[2]. For each venue (conference or journal) in each scientific domain, an HTTP request is issued to the API with specific parameters such as the venue name and the number of publications to retrieve. The response, in XML format, is read to extract useful information.
To ensure relevance and domain-specific coverage, we a list of venue names corresponding to different research domains (e.g., , databases, computer vision), based on their scientific focus. This mapping allowed targeted queries and the construction of a semantically structured dataset.
The following algorithm shows the process of loading the data from API:

[1] https://dblp.org
[2] https://dblp.org/search/publ/api

Algorithm 1 Build dataset from DBLP API by research domain

Input: A dictionary `domain_venues` mapping research domains to a list of venue names
Output: A structured list of publications with metadata

foreach *(domain, venues) in* `domain_venues` **do**
    foreach *venue in venues* **do**
        Set `params` ← {"q": "venue:{venue}", "h": articles_per_domain, "format": "xml"}
        `response` ← HTTP GET on `https://dblp.org/search/publ/api` with `params`
        Extract relevant metadata fields `// e.g., title, year, authors, venue, DOI`

2.5.2 Step 2: Transformation and structuring of the RDF data

The extracted XML has been transformed into RDF format, allowing for semantic modeling and graph-based reasoning. A unique URI is given to each entity (such as an author or publication), enabling flexible resource linking.

This step ensures the structural consistency of the graph and improves the quality of the representations learned subsequently.

A. Structure of the RDF Graph

The following fundamental entities are included in the RDF graph created from DBLP:

1. **Publications:** Representing a documented work, such as an article or paper, that has been formally released to the public, typically through a journal, conference, or academic platform.

    - Title: The title of the publication.
    - Date: The year in which the publication was released.
    - Doi: Digital Object Identifier (DOI), providing a permanent and unique reference to the publication.
    - URL: A direct web link to access the publication online.
    - Number of authors: The total number of authors involved in the publication. Calculated by counting the Creator relationships associated with a post.
    - isPartOf: Indicates the venue (e.g., journal or conference) where the publication appeared.
    - Creator: Refers to the authors who contributed to writing the publication.
    - HasCitationCount: The number of times this publication has been cited in other scientific works.
      This value is obtained automatically using an asynchronous citation aggregator, which queries several online services, including:

- CrossRef.
- OpenCitations.
- OpenAlex.
- Semantic Scholar.

The system queries these APIs in parallel, and then selects the maximum value from the valid responses.

```
<rdf:Description rdf:about="http://example.org/publication/f3121919157d7c640263a4bb85eb6398">
  <rdf:type rdf:resource="http://example.org/Publication"/>
  <dc:title>mutation coverage is not strongly correlated with mutation coverage</dc:title>
  <dc:date rdf:datatype="http://www.w3.org/2001/XMLSchema#gYear">2024</dc:date>
  <schema1:doi>10.1145/3644032.3644442</schema1:doi>
  <schema1:url>https://doi.org/10.1145/3644032.3644442</schema1:url>
  <dc:isPartOf rdf:resource="http://example.org/venue/a025d3cc913c91d1960aca5eb6ea9567"/>
  <dc:creator rdf:resource="http://example.org/author/9ef4f679dd0942a0b7ed61aad0faccf2"/>
  <dc:creator rdf:resource="http://example.org/author/1332bc72a00977d6de375ac5d90a7feb"/>
  <dc:creator rdf:resource="http://example.org/author/ab6a22fec730b8a4a45077ec07bb6489"/>
  <cit:hasCitationCount rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">0</cit:hasCitationCount>
  <feat:num_authors rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">3</feat:num_authors>
</rdf:Description>
```

Figure 2.2: Example of publication ressource

2. **Authors:** is a person who has created or contributed to a publication, typically through writing or research.

   - Name: The full name of the author.
   - Num publications: The total number of publications authored by this person. Determined by counting publications linked via the Creator property.
   - Num venues: The number of different venues in which the author has published. Identified by collecting the unique values of isPartOf for the author's publications.
   - Domain dominant: The primary research domain associated with this author, based on their publications.
     Automatically determined as the most frequent domain among its publications (HasDomain).
   - Active years: The number of years during which the author has been actively publishing.
     Calculated as max(years) - min(years) + 1.
   - HasPublishedIn: Indicates a venue (e.g., journal or conference) where the author has published.
   - HasDomain: The research domain in which the author has worked (e.g., Knowledge Graphs).

Figure 2.3: Example of Author Ressource

3. **Venues:** Representing the event or journal where a publication appears.

   - Label: The name or label of the venue.
   - VenueType: The type of the venue, such as "conference" or "journal".
     Determined by a heuristic based on the venue name (e.g., contains "journal" or "trans.").
   - Num publications: The total number of publications that have appeared in this venue.
     Counted from publications linked by isPartOf.
   - Popularity: A computed measure of how popular or influential the venue is, often based on publication or citation metrics.
     Calculated as the relative proportion of publications in this venue compared to the total (nb_pubs / total_pubs).
   - PublishesDomain: The main research domain that this venue publishes in.

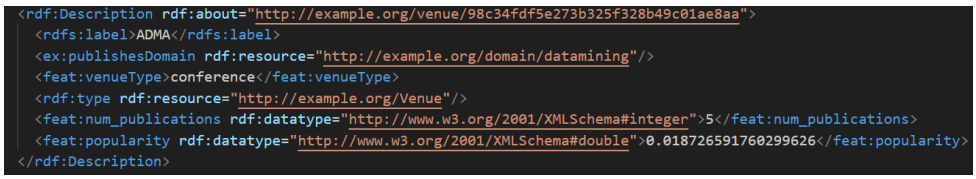

Figure 2.4: Example of venue ressource

4. **Domains:** Representing high-level scientific domains.

   - Label: The name of the domain.
   - Num venues: The number of venues that publish work in this domain.
   - Num publications: The total number of publications classified under this domain.

Figure 2.5: Example of domain ressource

The table 2.1 summarize the attributes and relations of each node type :

Node Type	RDF Type	Attributes	RDF Relations
Author	`foaf:Person`	- `foaf:name` - `feat:num_publications` - `feat:num_venues` - `feat:active_years` - `feat:domain_dominant`	- `ex:hasPublishedIn` → Venue - `ex:hasDomain` → Domain
Publication	`ex:Publication`	- `dc:title` - `dc:date` - `schema:url` - `cit:hasCitationCount` - `feat:num_authors`	- `dc:creator` → Author - `dc:isPartOf` → Venue
Venue	`ex:Venue`	- `rdfs:label` - `feat:num_publications` - `feat:popularity`	- `ex:publishesDomain` → Domain
Domain	`ex:Domain`	- `rdfs:label` - `feat:num_venues` - `feat:num_publications`	- Linked to other entities via their relations

Table 2.1: Description of Node Types in the RDF Graph

The following algorithm shows the process of transforming and creation of RDF graph :

Algorithm 2 Build RDF Graph Structure with Enriched Node Attributes

Input: Raw metadata for publications, authors, venues, domains
Output: RDF graph g with enriched typed nodes and literals

Function `BuildRDFGraph(`*metadata*`)` **:**
  Initialize RDF graph g
  foreach *publication* p **do**
    Add $(p, \texttt{rdf:type}, \texttt{ex:Publication})$ to g
    Add $(p, \texttt{dc:title}, \texttt{Literal(title)})$ to g
    Add $(p, \texttt{dc:date}, \texttt{Literal(year)})$ to g
    if *DOI exists* **then**
      Add $(p, \texttt{schema:doi}, \texttt{Literal(doi)})$
    if *URL exists* **then**
      Add $(p, \texttt{schema:url}, \texttt{Literal(url)})$
    Compute: num_authors
    Add $(p, \texttt{feat:num\_authors}, \texttt{Literal(value)})$ to g
  foreach *author* a *of* p **do**
    Add $(a, \texttt{rdf:type}, \texttt{foaf:Person})$ to g
    Add $(a, \texttt{foaf:name}, \texttt{Literal(name)})$ to g
    Add $(p, \texttt{dc:creator}, a)$ to g
    Compute: num_publications, num_venues, domain_dominant, active_years
    Add these as literals with properties under `feat:` namespace
    Add $(a, \texttt{ex:hasPublishedIn}, v)$ for all venues v
    Add $(a, \texttt{ex:hasDomain}, d)$ for dominant domain d
  foreach *venue* v **do**
    Compute: type, num_publications, popularity
    Add these as `feat:` properties
    Add $(v, \texttt{rdfs:label}, \texttt{Literal(name)})$ to g
    Add $(v, \texttt{rdf:type}, \texttt{ex:Journal or ex:Conference})$ to g
    Add $(p, \texttt{dc:isPartOf}, v)$ to g
    Add $(v, \texttt{ex:publishesDomain}, \texttt{domain})$ to g
  foreach *domain* d **do**
    Compute: num_venues, num_publications
    Add $(d, \texttt{feat:num\_venues}, \texttt{Literal})$
    Add $(d, \texttt{feat:num\_publications}, \texttt{Literal})$
  return g

B. Description of namespaces Used

In our RDF graph, several namespaces were used to ensure clarity, reuse, and semantic interoperability:

- **EX (http://example.org/)**:A namespace for all the domain specific objects such as Publication, Author, Venue, and Domain. It defines the basic schema of our dataset.

- **DC (http://purl.org/dc/elements/1.1/)** : Dublin Core vocabulary provides a set of standard properties for describing resources, including documents. This vocabulary is used to describe title, date, creator, and isPartOf.
- **SCHEMA (http://schema.org/)**: Provides additional metadata properties like URL and DOI, ensuring compatibility with web-based standards.
- **FOAF (http://xmlns.com/foaf/0.1/)**: Enables consistent description of people (Person) and their names.
- **FEAT (http://example.org/features/)**: A unique namespace for calculated features (e.g., `num_authors`, `popularity`, `venueType`) that enables graph enrichment and further analysis.
- **CIT (http://example.org/citation/)**: A namespace for citation data that supports bibliometric enrichment (e.g., hasCitationCount).

C. Data cleaning

After the enrichment phase, a final cleaning is performed to remove orphaned entities not linked to publications. Three types of nodes are concerned:

- Orphaned Authors : every node of type `foaf:Person` that does not appear in any `dc:creator` relationship is deleted.
- Orphaned Venues : every node of type `ex:Venue` not used in a `dc:isPartOf` relationship is deleted.
- Orphaned Domains : every `ex:Domain` that is not linked to any active venue (venue having at least one publication) via `ex:publishesDomain` relationship is deleted.
- Special Character Cleanup : all literal values (titles, names, abstracts, etc.) are cleaned to remove or replace special characters (e.g., `$, %, #, &, _`) using standard Unicode normalization and ASCII transliteration when necessary.
- Publication Filtering : only valid publication nodes are retained. A publication is considered valid if it has at least:
  - one title (`dc:title`) with non-empty content.
  - one author (`dc:creator`),
  - one venue (`dc:isPartOf`),
- Duplicate Removal via ID : all entities (authors, publications, venues, domains) are assigned a unique identifier (e.g., URI or hash-based ID), and duplicates are removed based on these IDs to ensure there are no redundant nodes.

The following algorithme shows the RDF cleaning algorithm:

Algorithm 3 RDF Graph Cleaning Algorithm

Input: RDF graph
Output: Cleaned RDF graph

```
/* Step 1: Remove orphaned authors */
foreach person ∈ foaf:Person do
    if person not in any dc:creator relationship then
        graph.remove(person);
/* Step 2: Remove orphaned venues */
foreach venue ∈ ex:Venue do
    if venue not in any dc:isPartOf relationship then
        graph.remove(venue);
/* Step 3: Remove orphaned domains */
foreach domain ∈ ex:Domain do
    if domain not linked via ex:publishesDomain to active venue then
        graph.remove(domain);
/* Step 4: Clean literal values from special characters */
foreach literal ∈ graph.getAllLiterals() do
    cleaned ← NormalizeAndTransliterate(literal);
    literal.setValue(cleaned);
/* Step 5: Filter invalid publications */
foreach pub ∈ dc:Publication do
    if pub lacks dc:creator, dc:isPartOf, or empty dc:title then
        graph.remove(pub);
/* Step 6: Remove duplicates by unique ID */
seenIDs ← {}
foreach entity ∈ graph.getAllEntities() do
    id ← entity.getUniqueIdentifier();
    if id ∈ seenIDs then
        graph.remove(entity);
    else
        seenIDs.add(id);
return graph
```

2.5.3 Step 3: Encoding attributs and relations

After preparing the RDF graph, we pass to encoding the node features and relations with semantically manner, to do that we used two NLP models to encode the textual attributes:

- **SentenceTransformer (MiniLM-L6-v2)** Light transformer model fine-tuned for generating semantically rich sentence embeddings. Employed for encoding the text attributes like person names, venue tags, domain names, and relation predicates.

- **SPECTER (Scientific Paper Embeddings)** A transformer-based model pre-trained on

scientific papers to generate publication title context embeddings. Used for encoding the text features of publications (e.g., paper titles).

For numeric attributes, we first apply a **logarithmic** transformation using the **log1p** function[3] to reduce large differences in values. This is helpful because some data, like citation counts or publication numbers, can vary a lot. After that, we scale all values between 0 and 1 using **Min-Max scaling**, which makes them easier to compare and use in machine learning algorithms. The following algorithm illustate the process of node features encoding:

Algorithm 4 Node Feature Encoding

Input: RDF graph
Output: Feature embeddings

```
/* Step 1:  Load models and tools                                    */
```

Load Specter model and tokenizer
Load MiniLM sentence embedding model
Initialize MinMax scaler for numeric features
Initialize empty dictionary node_embeddings

```
/* Step 2:  Iterate over RDF graph nodes                             */
```

foreach *node in RDF graph* **do**
    Initialize empty list embeddings
    if *node has text description* **then**
        Encode text with MiniLM → text_embedding
        Append text_embedding to embeddings
    if *node is a publication and has a title* **then**
        Encode title with Specter → title_embedding
        Append title_embedding to embeddings
    if *node has numeric features* **then**
        Apply log transform and MinMax scaling
        Encode as vector → numeric_embedding
        Append numeric_embedding to embeddings
    Concatenate all embeddings into a single vector
    Store in node_embeddings[node_id] = final_embedding
return *node_embeddings*

2.5.4 Step 4: Creation of the heterogeneous DGL graph

The next stage is to convert the data into a heterogeneous graph that is appropriate for machine learning, especially deep graph learning models, after cleaning the RDF graph and identifying the pertinent items and relations.

The Deep Graph Library (DGL), which offers a productive and adaptable interface for working with multi-type and multi-relational graphs, is used to carry out this change.

[3] `log1p(x)` computes $\log(1 + x)$, which is numerically more stable for small values of `x` close to 0. It also avoids issues with taking the logarithm of 0. This transformation helps reduce the skew in data with large magnitude differences.

A. Why DGL? DGL is specifically designed for representing and training deep graph models.It supports heterogeneous graphs, which is crucial in our case. And efficiently store features (attributes) for each node.

B. Transformation Steps:

1. **RDF Type Detection:** Based on the rdf data, we create a maping for each node type.
2. **Edge Construction:** Only RDF relations that connect two typed entities are kept after analysis. The standard triplets of the form (sourcetype, relationtype, and targettype) are used to organize edges.
3. **Local Indexing:** As required by DGL, every URI is assigned a unique local integer identifier.
4. **Graph Creation:** The dgl.heterograph() function creates a heterogeneous graph using the node and edge dictionaries.
5. **Feature Assignment:** Each node is assigned a precomputed embedding, derived from features that were previously encoded, based on the node's URI.

The followed algorithm contain the process of construction the heterogeneous graph:

Algorithm 5 Construction of the Heterogeneous Graph with DGL

Input: RDF graph, feature embeddings
Output: Heterogeneous graph DGL

```
/* Step 1: Initialization                                    */
```

Load RDF graph
Load node feature embeddings
Define RDF namespaces and entity type mapping
Initialize empty dictionaries for node types and edge lists

```
/* Step 2: Assign node types                                 */
```

foreach $node \in$ *RDF graph* **do**
    if $node$ *has an RDF type* **then**
        Assign corresponding node type

```
/* Step 3: Construct edge lists                              */
```

foreach *triple* $(s, p, o) \in$ *RDF graph* **do**
    if s *and* o *have known types* **then**
        Map (s, p, o) to edge: type $(type(s), relation, type(o))$
        Add (s, o) to edge list for that relation

```
/* Step 4: Build the DGL graph                               */
```

Create heterogeneous graph with nodes, edges, and types using DGL
Assign precomputed embeddings as node features
return *Heterogeneous DGL graph*

This is a sample of our DGL heterogeneous graph figured in figure 2.6:

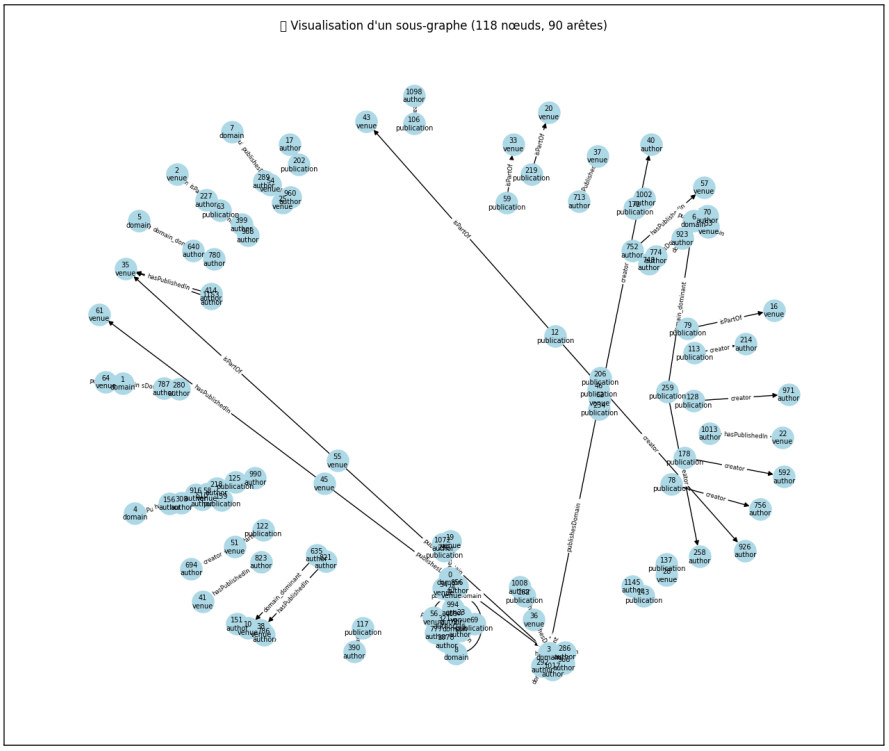

Figure 2.6: Sample of DGL heterogeneous graph

2.6 Phase 2 : Extracting embeddings

To obtain high-quality node representations, we adopted a two-stage embedding pipeline combining a Variational Graph Autoencoder and a Heterogeneous Graph Transformer. The VGAE serves as a feature preprocessing step that encodes the initial node attributes,which are often noisy or high-dimensional,into compact and regular latent vectors. These embeddings are then used as input to the HGT model, which further integrates the heterogeneous structure of the graph.

This design allows us to decouple the semantic compression of node features (via VGAE) from the structural modeling (via HGT), improving robustness and overall representation quality.

2.6.1 Feature preprocessing via a variational graph autoencoder

The Variational Graph Autoencoder is an unsupervised learning framework designed to generate low-dimensional representations of graph-structured data. It extends the standard Graph Autoencoder (GAE) by incorporating a probabilistic approach to the encoding process.

While a standard autoencoder learns to compress input features into deterministic vectors, the VGAE models each node's latent representation as a probability distribution, typically a

Gaussian. This allows the model to capture uncertainty and variability in the node embeddings, leading to more robust and generalizable representations.

In our approach, the VGAE is used as a preprocessing module before the HGT model. It transforms noisy or high-dimensional features into more compact and regular latent vectors that better reflect the underlying semantics of the graph. Each node type in the heterogeneous graph is processed independently using a dedicated VGAE model.

2.6.1.1 VGAE architecture development

To preprocess the raw node features, we implemented a Variational Graph Autoencoder for each node type in the heterogeneous graph. The VGAE follows a modular encoder-decoder architecture with a probabilistic latent space, allowing it to learn compact and regularized node embeddings.

Figure 2.7 represent the architecture of VGAE model :

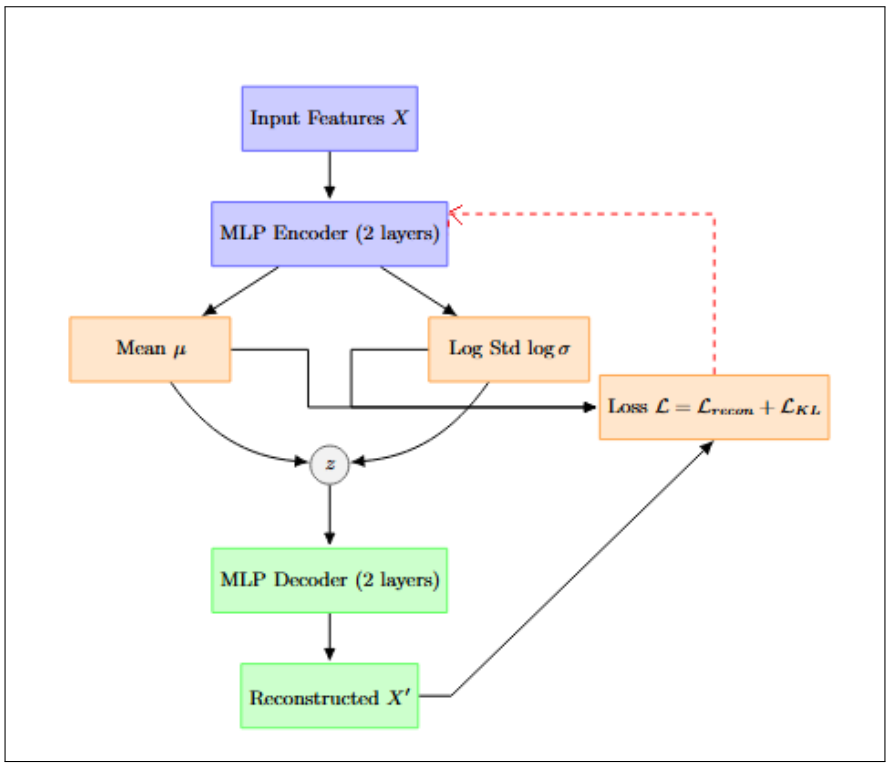

Figure 2.7: VGAE architecture

The architecture consists of:

- A two-layer multilayer perceptron (MLP) encoder with dropout and ReLU activations, which transforms the input features into a hidden representation.
- Two parallel linear layers that output the mean vector μ and the log-standard deviation vector $\log \sigma$ of the latent Gaussian distribution.

- A reparameterization mechanism that samples latent vectors $z \sim \mathcal{N}(\mu, \sigma^2)$, enabling backpropagation through stochastic sampling.
- A symmetric MLP decoder that attempts to reconstruct the original input features from the sampled vector z.

The VGAE is trained to minimize a combined objective:

- A reconstruction loss (Mean Squared Error) :
  used to measure the difference between the input features and their reconstruction. that ensures the decoder accurately reproduces the input features.
- A Kullback-Leibler (KL) divergence term, that encourages the learned latent distribution to stay close to a standard normal distribution $\mathcal{N}(0, I)$. It is computed as:

$$L_{KL} = -\frac{1}{2}\mathbb{E}\left[1 + 2\log\sigma - \mu^2 - e^{2\log\sigma}\right]$$

Once trained, the latent embeddings $Z \in \mathbb{R}^{N\times d}$ are obtained from the mean vectors μ and normalized to ensure consistency across the embedding space.

This preprocessing step allows the downstream HGT model to receive cleaner and more meaningful node features, improving the overall representation quality and reducing the impact of noise or sparsity in the raw data.

2.6.1.2 Model Configuration and Training

Before starting the training of the VGAE model, it is crucial to define the model parameters. To this end, we carefully defined all necessary parameters to ensure the model is properly configured to learn effectively from the input data.

1. Activation Functions and Regularization To enable efficient and robust learning in neural networks, two key components are essential: activation functions and regularization techniques. Activation functions introduce non-linearity into the model, allowing it to learn complex patterns. Regularization techniques, on the other hand, help prevent overfitting and enhance the model's generalization ability.

- **Activation Function : ReLU(Rectified Linear Unit)**
  is a widely used non-linear function in neural networks. It preserves only the positive values by setting all negative inputs to zero. This helps accelerate convergence and mitigates the vanishing gradient problem. It is defined as:

$$f(x) = \max(0, x)$$

  In our model, the ReLU function is applied after each linear layer, except the last decoder layer.

- **Regularization : Dropout**
  Regularization is a technique used to prevent overfitting. In our architecture, we used the Dropout technique, which randomly disables a portion of the neurons during training. This forces the model to rely less on specific connections and promotes generalization.
  We applied a dropout rate of 0.2 after each hidden layer. This means that 20% of neurons are ignored during each training iteration.

Algorithm 6 shows the training pipeline of the Variational Graph Autoencoder, where each node type in the heterogeneous graph is encoded independently into a latent Gaussian space using a two-layer MLP, followed by reparameterization and reconstruction, with a combined loss that includes a reconstruction term and a KL divergence regularization

Algorithm 6 VGAE for Node Feature Preprocessing

Input: Node features X_n for node type n
Output: Normalized latent embeddings $Z_n \in \mathbb{R}^{N \times d}$

$H^{(1)} \leftarrow \text{Dropout}(\text{ReLU}(\text{MLP}_1(X_n)))$
$H^{(2)} \leftarrow \text{Dropout}(\text{ReLU}(\text{MLP}_2(H^{(1)})))$
$\mu \leftarrow \text{Linear}_\mu(H^{(2)})$
$\log \sigma \leftarrow \text{Linear}_{\log \sigma}(H^{(2)})$
Sample $\epsilon \sim \mathcal{N}(0, I)$
$Z_n \leftarrow \mu + \exp(\log \sigma) \odot \epsilon$
$\hat{X}_n \leftarrow \text{DecoderMLP}(Z_n)$
$\mathcal{L} \leftarrow \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{KL}}$
Backpropagate and update parameters using $\mathcal{L}$
$Z_n \leftarrow \text{Normalize}(\mu)$
return Z_n

2.6.2 Extracting embeddings via HGT model

The Heterogeneous Graph Transformer is a neural architecture specifically designed for learning on heterogeneous graphs,wich contain multiple types of nodes and edges. Unlike traditional graph neural networks that assume a homogeneous structure, HGT explicitly distinguishes between different types of entities and interactions. This distinction is particularly well suited for our RDF-based setting, where semantic diversity is inherent in the data.
In our approach, the HGT model is applied after the initial feature preprocessing stage performed by the Variational Graph Autoencoder. The latent vectors generated by the VGAE,compact and denoised versions of the raw features,are used as input to the HGT. The transformer then leverages the topological and semantic heterogeneity of the graph to produce high quality, context aware node embeddings.

2.6.2.1 HGT Architecture Design

Figure 2.8 represent an overview of one HGT layer with integrated contrastive learning. Neighbor nodes send messages through relation-type specific projections and multihead attention. After message aggregation and node update via residual normalization, a contrastive loss is applied using semantically defined positive and negative pairs.

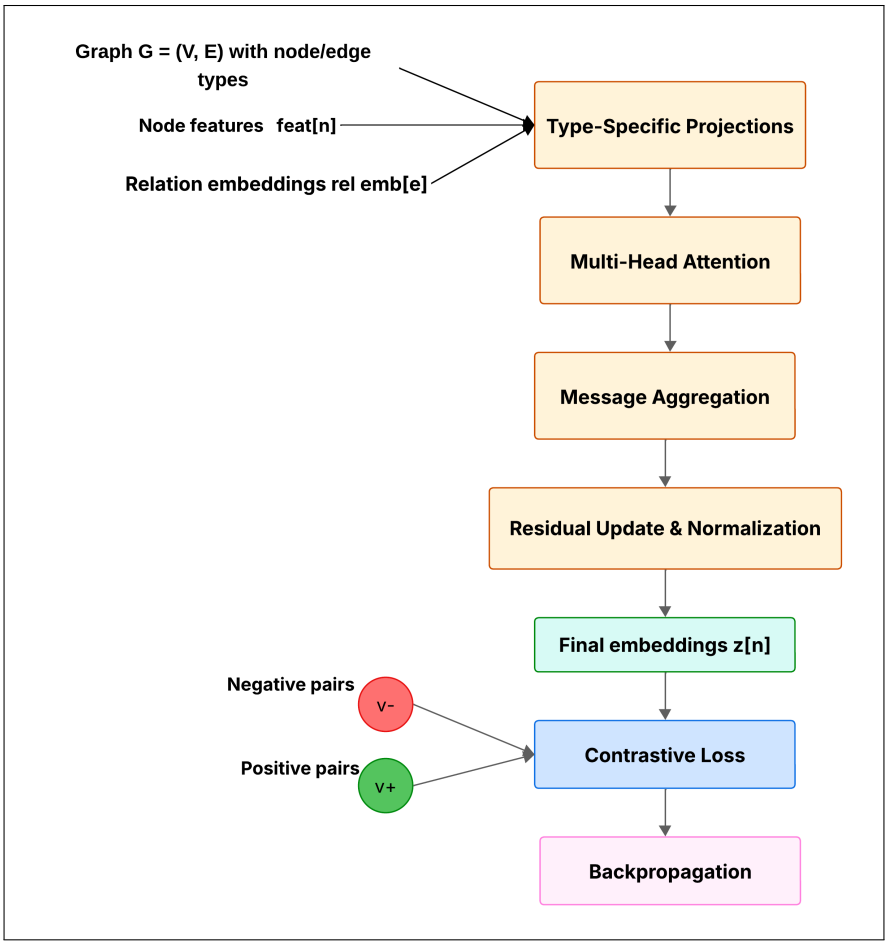

Figure 2.8: Pipeline of HGT layer

The HGT model implemented follows several key principles:

- **Type-Specific Projections**: Each node and relation type is associated with its own learnable projection matrices, allowing the model to encode type-specific semantics.
- **Multi-Head Attention Mechanism**: A multi-head attention mechanism enables the model to capture diverse patterns of interaction by allowing each attention head to focus on a different subspace.
- **Residual Connections and Normalization**: Each HGT layer is followed by layer normalization and residual connections, which facilitate deep learning and stabilize training.

Each HGT layer performs the following sequence of operations:

1. Compute attention scores between each node and its neighbors, conditioned on the node and edge types.
2. Aggregate messages from neighboring nodes, weighted by attention scores.
3. Apply a linear transformation based on the target node's type.
4. Update the node representation using residual summation followed by normalization.

We stack multiple HGT layers to enable information propagation across multi hop neighborhoods, while preserving type-aware modeling throughout the architecture.

2.6.2.2 Contrastive Learning Objective

To further enhance the expressiveness and discriminative power of the node embeddings generated by the HGT model, we adopt a contrastive learning approach during training.
Contrastive learning encourages the model to bring semantically similar nodes closer together in the latent space while pushing dissimilar nodes further apart.
This approach is particularly well suited to heterogeneous graphs, where semantically similar nodes (e.g., authors from the same research area, publications in the same venue, or entities sharing contextual relations) may be structurally distant in the raw topology but exhibit meaningful latent similarities.
In our pipeline, we define positive pairs (v, v^+) based on domain specific criteria. These include nodes with high co-occurrence frequency in RDF triples.
Negative samples v^- are drawn randomly from the remainder of the graph, ensuring type consistency to preserve the semantic validity of comparisons.
We formulate the contrastive loss using a node-level InfoNCE-based objective:

$$\mathcal{L}_{\text{contrastive}} = -\log \frac{\exp(\text{sim}(z_v, z_{v^+})/\tau)}{\sum_{v^- \in \mathcal{N}} \exp(\text{sim}(z_v, z_{v^-})/\tau)}$$

where:

- z_v, z_{v^+}, and z_{v^-} denote the HGT embeddings of the anchor, positive, and negative nodes, respectively;
- $\text{sim}(\cdot, \cdot)$ represents the similarity function, implemented as cosine similarity:

$$\text{sim}(\mathbf{z}_i, \mathbf{z}_j) = \frac{\mathbf{z}_i^\top \mathbf{z}_j}{\|\mathbf{z}_i\| \cdot \|\mathbf{z}_j\|}$$

- τ is a temperature hyperparameter that controls the sharpness of the probability distribution.
  - When τ is low (less than 1), the differences between logits are amplified, resulting in a sharper, more confident distribution concentrated on the top choices.

– When τ is high (greater than 1), the differences are diminished, producing a flatter distribution that encourages exploration of multiple options.

- $\mathcal{N}$ is the set of negative samples for a given anchor node v.

This loss function is jointly optimized with the HGT encoder during training. Moreover, the contrastive module is inserted after each HGT layer to promote representational consistency across hierarchical levels of abstraction.

By explicitly defining semantically meaningful positive pairs, we guide the model to align with the underlying structure of the knowledge graph. This significantly enhances the cohesion of embeddings within conceptual clusters and improves the performance of the downstream deep graph clustering .

Algorithm 7 shows the training the Heterogeneous Graph Transformer model with contrastive loss by encoding typed nodes and edges, propagating information through multi-head attention layers, and optimizing node representations to ensure that semantically similar node pairs (positives) are embedded closer than dissimilar ones (negatives) in the latent space.

Algorithm 7 Heterogeneous Graph Transformer with Contrastive Loss

Input: Graph $G = (V, E)$ with node and edge types;
Node features feat$[n]$ for each node type n;
Relation embeddings rel_emb$[e]$ for each edge type e;
Positive node pairs $P = \{(a_i, b_i)\}$ for contrastive learning;
Output: Final node embeddings $z[n]$ for all node types n

Initialize one input linear layer per node type
Initialize L HGT layers
Initialize one output linear layer per node type
foreach *node type* n **do**
    $h[n] \leftarrow \text{InputLayer}_n(\text{feat}[n])$
for $\ell = 1$ **to** L **do**
    $h \leftarrow \text{HGTLayer}_\ell(G, h, \text{rel\_emb})$
    Add residual connection: $h[n] \leftarrow h[n] +$ previous $h[n]$ for all n
foreach *node type* n **do**
    $z[n] \leftarrow \text{OutputLayer}_n(h[n])$
    Apply batch normalization on $z[n]$ if number of nodes > 1
Compute contrastive loss: $\mathcal{L} \leftarrow \text{ContrastiveLoss}(z[n], P, \tau, m)$
Backpropagate and update model parameters using $\mathcal{L}$
return $z[n]$ for each node type n

2.7 Phase 3 : Clustering

After extracting the embeddings from our data using the HGT model, we are able to apply different clustering algorithms.

We applied two clustering approaches: global clustering and per-node-type clustering.

We experimented with several clustering algorithms, including traditional methods such as K-Means, Hierarchical Clustering, and spectral clustering , as well as neural methods such as Deep Embedded Clustering (DEC).

- **Determine the Optimal Number of Clusters**
  To determine the optimal number of clusters, we adopted a hybrid strategy. When ground truth labels were available, we directly used the number of unique labels as the predefined number of clusters. However, in the absence of such labels, we relied on internal evaluation techniques. Specifically, we applied the Silhouette Score to estimate the most appropriate number of clusters based on the data's structure. These complementary approaches allowed for both supervised and unsupervised determination of cluster counts, as summarized in the following table 2.2

Method	**Description**
Ground Truth Labels	Use known class labels to guide or validate clustering.
Silhouette Score	Measures how similar an object is to its own cluster compared to other clusters.

Table 2.2: Methods for Determining the Optimal Number of Clusters

- **Cluster Label Alignment**
  In unsupervised clustering, the assigned cluster labels are arbitrary and do not directly correspond to ground truth classes. To evaluate clustering performance using supervised metrics (e.g., Accuracy, F1-score), an alignment step is required. We perform the following:

  - **Issue with raw clustering labels:** Cluster IDs produced by clustering algorithm (e.g., 0, 1, 2, ...) do not necessarily match the true class labels.
  - **Optimal matching:** We use the **Hungarian algorithm** (Kuhn-Munkres) to find the best one-to-one mapping between predicted clusters and ground truth labels.
    which is an optimal assignment problem algorithm. It allows to find the optimal matching of two sets (e.g., forecast clusters and actual classes) to achieve a maximum number of correct assignments.
    In clustering, since cluster labels are arbitrary (e.g., cluster 0 is the same as true class 2), this algorithm gives a method to map every cluster to the correct class. It does this by comparing all the possible mappings and then picking the one with the most agreements (well classified instances).
    Specifically, it takes as input a confusion matrix (i.e., the number of samples of each cluster in each class), and finds the cluster to class mapping that gives the best overall fit.

– **Accurate evaluation:** This alignment enables reliable computation of evaluation metrics like Accuracy, NMI, ARI, and F1-score.

2.7.1 Partitioning Method: k-means

K-means clustering is an unsupervised learning algorithm for data clustering in which unlabeled data points are divided into groups or clusters.
K-means is an iterative algorithm for centroid-based clustering that partitions a dataset into clusters with analogous elements based on the distance of their centroids. The centroid or center of the cluster may be the mean or median of all the points in the cluster, depending on the type of data.

2.7.1.1 Adaptation of k-means algorithm to our approch

In our work, we applied K-Means clustering to the node embeddings extracted from the encoder module. The goal is to group similar nodes based on their learned latent representations.
The pipeline of k-means clustering is illustrated in Figure 2.9

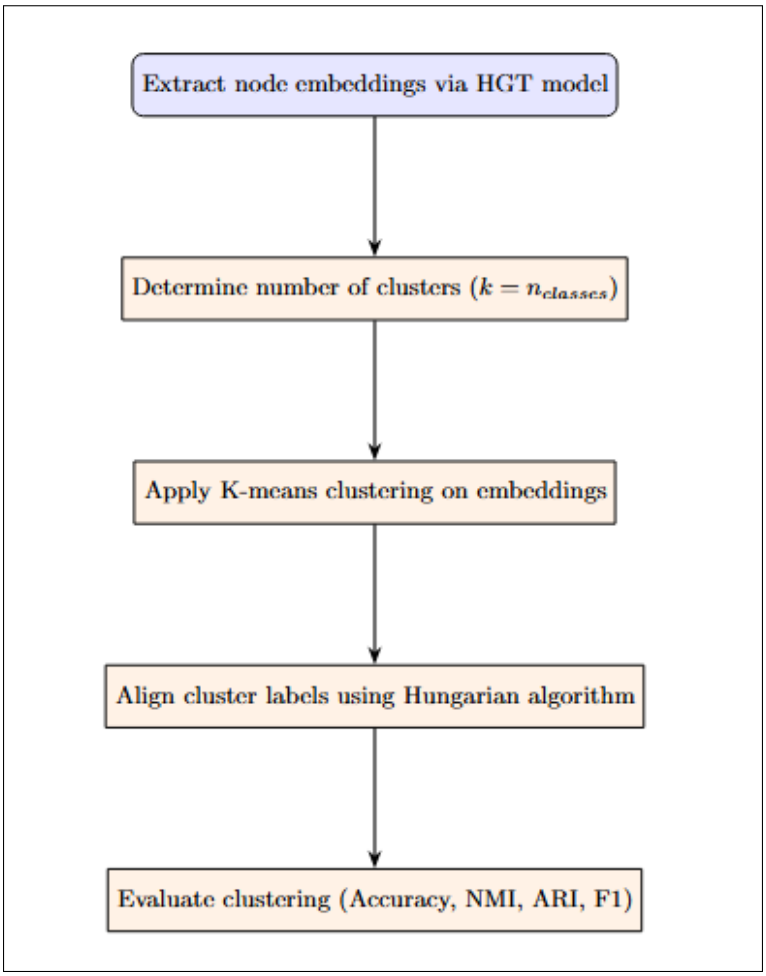

Figure 2.9: Pipeline of k-means clustering

- **Distance Metric**
  We employed the standard Euclidean distance to measure the similarity between node embeddings. Given two embedding vectors $\mathbf{z}_i$ and $\mathbf{z}_j$, the Euclidean distance is defined as:

$$d(\mathbf{z}_i, \mathbf{z}_j) = \|\mathbf{z}_i - \mathbf{z}_j\|_2$$

  This metric is well-suited for continuous, dense embeddings in the latent space.
  This strategy allows for a direct evaluation of the model's ability to recover the actual categorical structure via supervised metrics (NMI, ARI, F1, accuracy).
- **Clustering Process**
  The overall clustering process using KMeans can be summarized in three main steps:
  1. Determine number of clusters.
  2. The KMeans algorithm is applied to partition the embeddings into K clusters, using the Euclidean distance and multiple initializations for robustness.
  3. The cluster labels are then optimally aligned with the actual labels to evaluate the clustering quality using supervised metrics (NMI, ARI, F1-score, accuracy).

2.7.2 Hierarchical methods : Agglomerative Hierarchical Clustering

Hierarchical clustering is a method that builds a nested hierarchy of clusters through either agglomerative or divisive strategies. In our work, we applied agglomerative hierarchical clustering on the node embeddings extracted by the HGT module.

2.7.2.1 Adaptation of Agglomerative algorithm to our approch

We applied hierarchical clustering to the node embeddings extracted from the encoder module. The goal is to group similar nodes based on their learned latent representations.
The pipeline of hierarchical clustering is illustrated in Figure 2.10

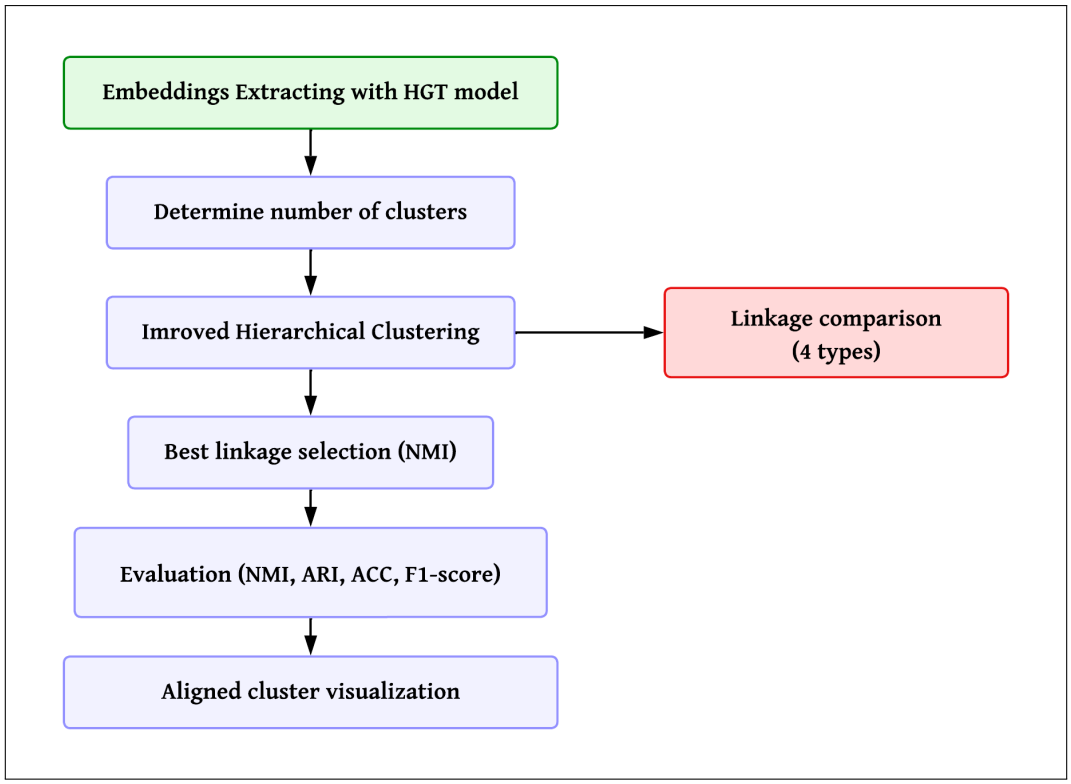

Figure 2.10: Pipeline of hierarchical clustering

In our work, we experimented with several variants of hierarchical clustering, including Ward, Complete, Average, and Single linkage methods.

- **Linkage:**
  Defines the method used to compute the distance between clusters. Four types of linkage are used:

Method	**Formula**	**Principle**
Ward	$D(c_1, c_2) = \frac{\lvert c_1 \rvert \lvert c_2 \rvert}{\lvert c_1 \rvert + \lvert c_2 \rvert} \lVert \mu_{c_1} - \mu_{c_2} \rVert^2$	Minimizes the intra-cluster variance. This is ideal for compact and well-separated clusters.
Complete	$D(c_1, c_2) = \max D(x_1, x_2)$	Uses the maximum distance between two points from different clusters. This favors more separated clusters.
Single	$D(c_1, c_2) = \min D(x_1, x_2)$	Minimizes the distance between the closest points of each cluster. This can lead to elongated or chained clusters.
Average	$D(c_1, c_2) = \frac{1}{\lvert c_1 \rvert} \cdot \frac{1}{\lvert c_2 \rvert} \sum \sum D(x_1, x_2)$	Computes the average of all pairwise distances between points in the two clusters. This promotes more balanced clusters.

Table 2.3: Comparison of Hierarchical Clustering Linkage Methods

- **Distance Metric:**
  We used the Euclidean distance as the base metric, which aligns well with the embedding space produced by the HGT encoder.

- **General Process:**
  The overall clustering process using Agglomerative clustering can be summarized in four main steps:

  1. First, we determine the number of clusters.
  2. Next we apply agglomerative hierarchical clustering using all four linkage methods.
  3. Each configuration is evaluated using clustering metrics, and we select the linkage method that yields the best performance.
  4. Finally, we visualize the resulting clusters to analyze the distribution and separation of node groups in the latent space.

2.7.3 Spectral clustering

Spectral Clustering is a clustering method that uses the spectral properties (eigenvalues) of an affinity matrix to group data. It is particularly effective for capturing complex and non-linear structures in the data.
This method is especially well-suited for graph-structured data, as it exploits the local similarity relationships between nodes. One of its key advantages is its ability to detect non-convex clusters, which is crucial in heterogeneous graphs, where the boundaries between node classes can be highly intricate and irregular.
- Affinity in Spectral Clustering
In spectral clustering, the affinity matrix (also called the similarity matrix) captures the pairwise similarity between data points. Each entry in this matrix indicates how "close" two nodes are, typically based on some distance metric higher values reflect stronger similarity.

2.7.3.1 Adaptation of Spectral clustering to Our Approach

Spectral Clustering is used during the evaluation phase of embeddings, after their extraction by HGT model on graph, we apply spectral clustering directly on the embedding vectors to identify meaningful groups of nodes.
Figure 2.11 illustrate the pipeline of spectral clustering

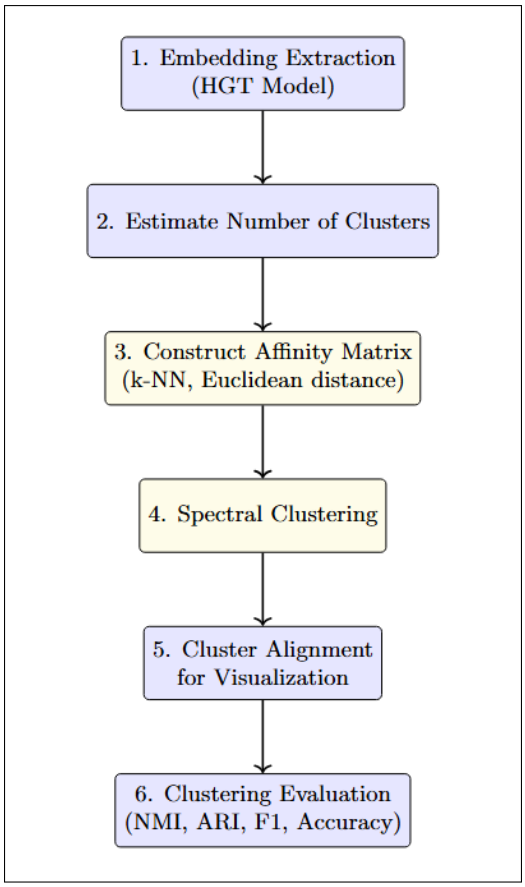

Figure 2.11: Spectral clustering pipeline

- **Affinity**
  To build the affinity matrix required by spectral clustering, we use a k-nearest neighbors (k-NN) approach. In this method, a similarity graph is constructed where each node is connected only to its k nearest neighbors based on Euclidean distance.
  This results in a sparse affinity matrix, which helps better capture the local manifold structure of the data, reducing noise from distant or irrelevant nodes.

- **General Process:**
  The overall clustering process using Spectral clustering can be summarized in this steps:

  1. First, we determine the number of clusters.
  2. Next we construct Affinity Matrix using k-NN approach, and we apply the spectral clustering.
  3. The clusters results are evaluated using clustering metrics.
  4. Finally, we visualize the resulting clusters aligned .

2.7.4 Neural Clustering: Deep Embedded Clustering (DEC)

Deep Embedded Clustering is a neural clustering method that jointly optimizes the latent representation of data and the cluster assignments. It starts with an initial clustering using K-Means, then refines the assignments through a differentiable optimization process based on a Student's t-distribution and Kullback–Leibler divergence.

- Mechanism of DEC

Figure 2.12 shows the overall pipeline of clustering by DEC

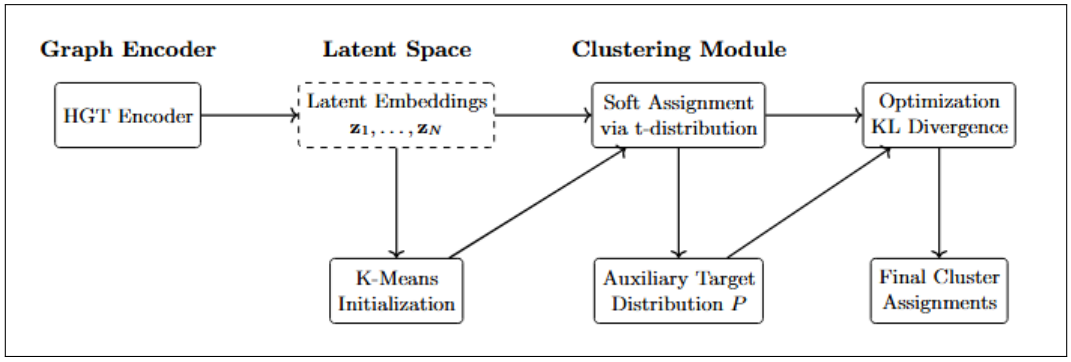

Figure 2.12: Pipeline of clustering by DEC

- **Initialization with K-Means:**
  The algorithm begins by applying K-Means to the embedded data to initialize the cluster centers. This provides a suitable starting point that facilitates convergence during optimization.

- **Student's t-Distribution for Soft Assignment:**
  A Student's t-distribution is used to compute the soft assignment probabilities of data points to clusters. This distribution assigns higher probabilities to nearby points while reducing the influence of distant ones, thus capturing both local and global structure effectively.

- **Auxiliary Target Distribution (P):**
  An auxiliary distribution is constructed to emphasize high-confidence assignments. It sharpens the soft labels produced by the t-distribution and amplifies the influence of points that are close to cluster centers.

- **Optimization via KL Divergence:**
  The model is optimized by minimizing the Kullback–Leibler divergence between the soft assignment distribution (Q) and the auxiliary distribution (P). Both the encoder parameters and the cluster centers are updated iteratively to reduce this divergence and improve clustering quality.

- **Convergence Criterion:**
  The optimization process continues until the KL divergence between successive iterations becomes negligible, indicating that the model has reached a stable clustering solution.

2.7.4.1 Adaptation of DEC to Our Approach

To effectively cluster nodes from a heterogeneous graph, we adapted the Deep Embedded Clustering model to operate on embeddings obtained from our graph representation learning pipeline.
The adaptation involves the following steps:

- **Subsequently Training Strategy:**
  In our approach, we first learn the embeddings with HGT, and then apply DEC in a separate step on these embeddings.

- **Cluster Initialization and Update:**
  Initial cluster centers are computed using K-Means on the embeddings. These centers are then iteratively refined by minimizing the KL divergence between the soft assignment distribution (Q) and the sharpened auxiliary target distribution (P), as in standard DEC.

2.7.5 Clustering Strategies: Global vs. Type-Specific

In order to evaluate the effectiveness of clustering on heterogeneous graph embeddings, we explored two complementary strategies: **global clustering** and **clustering by node type**. Each strategy reflects a different goal and offers distinct insights into the structure of the graph.
Figure 2.13 illustrate the two clustering strategies

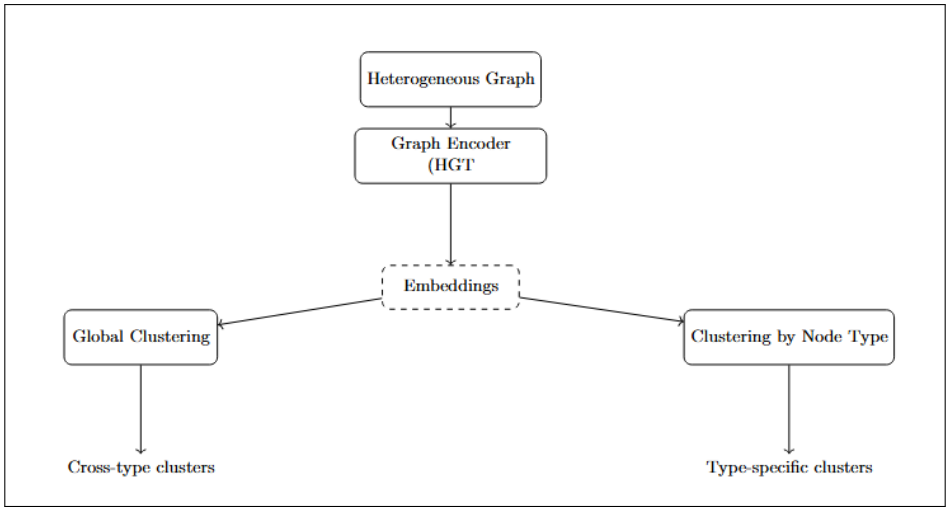

Figure 2.13: Different clustering strategies used

Approach 1: Global Clustering In this approach, we project all node types (e.g., authors, publications, venues, domains) into a shared latent embedding space using our graph representation model. Then, we apply clustering algorithms directly on the union of all embeddings.

- We concatenate the embeddings of all node types.
- We apply clustering algorithms to this unified representation.
- The goal is to discover semantic clusters that span across different types, e.g., linking authors to publications and venues under shared domains.

This approach is particularly relevant when the clustering task is intended to capture **cross-type semantics**, such as domain-based grouping or multi-type recommendation.

Approach 2: Clustering by Node Type In this approach, we separate embeddings by node type and perform clustering independently on each group.

- Embeddings are grouped by node type (e.g., authors, publications).
- Clustering is applied separately to each type using the same set of algorithms.
- The objective is to identify coherent groups within a single type, such as author communities or topic-based clusters of publications.

This strategy enables type-specific analyses and is suited for tasks like author disambiguation, thematic classification, and venue profiling.

Comparison and Motivation Using both approaches allows us to:

- Explore global structures that emerge across node types.
- Gain insight into intra-type relationships specific to certain tasks.
- Choose the most appropriate strategy based on the intended application (e.g., joint recommendation vs. specialized analysis).

2.8 Phase 4 : Evaluation and visualization

After applying several clustering algorithms to the embeddings in the two approches used, we now reach a crucial phase aimed at evaluating and validating the quality of the resulting clusters. This will be presented in the next chapter, including evaluation metrics and clusters visualization.

2.9 Conclusion

Throughout this chapter, we have presented the detailed steps of our solution for clustering on a heterogeneous graph from linked data. We have detailed the three main phases: data preprocessing, extracting embeddings, as well as the exploration and implementation of various clustering approaches.
In the following chapter, we will implement what we have already proposed in this chapter, the detailed results of our experimentations will be presented together with their thourough discussion and analysis .

Chapter 3

Implementation and test

3.1 Introduction

After completing the design phase of our system, this chapter addresses its practical implementation, which is based on the semantic clustering of knowledge graphs introduced earlier. We begin by presenting the development environments and tools employed, followed by a detailed explanation of the implementation process.
This chapter addresses the evaluation of our method, including several evaluation metrics, and visualizations of the obtained clusters to support the analysis.
Finally, we describe the application of our system within a real or simulated context to demonstrate its effectiveness and usability.

3.2 Technologies and Languages Used

Python : is a high-level, interpreted programming language known for its simplicity, readability, and versatility. Released in 1991 by Guido van Rossum, it supports multiple paradigms, including object-oriented, procedural, and functional programming. Python is widely used across domains such as web development, data science, artificial intelligence, and automation, thanks to its rich standard library and extensive third-party ecosystem[1].
In this project, Python was used as the primary programming language due to its ease of use and the availability of robust libraries for scientific computing and graph-based deep learning.
Below is a summary of the main Python libraries and frameworks employed:

- **PyTorch:** A deep learning framework widely used for building and training neural networks. It was used in this project to implement autoencoder and neural components for graph representation learning[2].

[1] `https://www.python.org/doc/essays/blurb/`
[2] `https://pytorch.org/`

- **DGL (Deep Graph Library):** A Python package dedicated to deep learning on graphs. It supports both homogeneous and heterogeneous graphs and provides efficient implementations for graph neural network layers. It was used here to model and manipulate heterogeneous RDF-based graphs[3].
- **RDFlib:** A library for working with RDF data in Python. It was used to parse and query RDF files, transforming them into a format compatible with graph learning models[4].
- **Scikit-learn:** This library provided classical machine learning tools for clustering (e.g., KMeans), dimensionality reduction (e.g., PCA, t-SNE), and evaluation metrics such as Adjusted Rand Index (ARI), Normalized Mutual Information (NMI), F1-score, and Accuracy[5].
- **NumPy:** A fundamental package for scientific computing in Python. It was used for numerical operations, data manipulation, and handling multidimensional arrays[6].
- **Matplotlib:** A plotting library used for data visualization, particularly for visualizing the results of dimensionality reduction and clustering in two-dimensional space[7].
- **Pickle:** Used for object serialization and deserialization during training and evaluation phases, allowing for saving intermediate representations and model checkpoints[8].
- **Flask:** A lightweight web framework for building web applications and APIs in Python[9].
- **Other Standard Libraries:** Python's standard libraries such as **os**, **time**, **random**, and **collections** (e.g., **Counter**, **defaultdict**) were also used for file management, performance monitoring, randomness control, and data structure management[10].

Moreover, to create the interface of our system,we have used the following front-end technologies:

- **HTML (Hypertext Markup Language):** A tag-based code structure language used for markup. It supports static web page presentation and is used to write hypertext[11].
- **CSS (Cascading Style Sheets):** A language that is used to define the style to web documents. CSS is widely used in web development and is supported by all modern web browsers. The CSS standards are written by all modern web browsers. The standards defining CSS are maintained by the World Wide Web Consortium (W3C)[12].

[3] https://www.dgl.ai/
[4] https://rdflib.dev/
[5] https://scikit-learn.org/
[6] https://numpy.org/
[7] https://matplotlib.org/
[8] https://docs.python.org/3/library/pickle.html
[9] https://flask.palletsprojects.com/
[10] https://docs.python.org/3/library/
[11] https://www.w3.org/wiki/HTML/Training/What_is_HTML
[12] https://www.w3.org/Style/CSS/Overview.en.html

- **JavaScript:** A high-level scripting language used to create dynamic and interactive features on web pages, such as user interactions, animations, and asynchronous data exchange [13].

3.3 Hardware Used

All experiments and evaluations conducted in the context of this thesis were performed on a computing system with the hardware configuration detailed below.

Component	**Specification**
Processor	AMD Ryzen 5 5600G with integrated Radeon Graphics
Number of cores	6
Number of threads	12
Base clock frequency	3.90 GHz
RAM	16 GB
Operating System	Windows 11 Pro, 64-bit edition

Table 3.1: Hardware configuration used during experimentation

3.4 Software Used

The implementation, development, and evaluation tasks throughout this project were carried out using this software environment:
Visual Studio Code : VS Code is a free, open-source, and lightweight code editor developed by Microsoft. It supports multiple programming languages (e.g., Python, Java, C++) and offers features like IntelliSense, debugging, Git integration, an integrated terminal, and a rich extension marketplace. Thanks to its flexibility, cross-platform compatibility, and strong community support, VS Code is one of the most widely used editors in modern software development.

3.5 Test Procedure

To solve the issue of graph clustering, we investigated two primary approaches: type-based clustering and global clustering. Within each approach, we examined two architectural settings. Initially, we employed the Heterogeneous Graph Transformer alone with an array of clustering algorithms (like K-means, hierarchical clustering, and DEC). Subsequently, we tried a hybrid model using the Variational Graph Autoencoder followed by HGT, as well as the application of a number of clustering algorithms. This arrangement allowed us to directly compare the effects of various embedding approaches on clustering performance. Lastly, to validate the correctness of

[13] `https://www.w3schools.com/js/js_intro.asp`

our approach, we constructed a recommendation system over the output of global and type-based clustering, where we could verify the usability of each approach.

3.5.1 Evaluation of clustering results

To assess the quality of the clustering results, we used **true labels** derived from the high level research domains (e.g., artificial intelligence, databases, computer networks) associated with each node (publications, authors, venues). These domain labels were extracted and manually verified from the metadata, and used as ground truth for evaluation.

The following external clustering metrics were used to quantitatively measure the clustering performance:

- **Normalized Mutual Information (NMI):** Measures the mutual dependence between the predicted clusters and the true labels. NMI is bounded between 0 (no mutual information) and 1 (perfect match), making it robust to label permutation.

- **Adjusted Rand Index (ARI):** Computes a similarity measure between two clusterings by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in both the predicted and true clusterings. ARI is adjusted for chance, and its value ranges from -1 to 1.

- **Accuracy:** The clustering accuracy was computed by first aligning predicted clusters with true labels using the Hungarian algorithm, then computing the proportion of correctly assigned samples.

- **F1-score:** The harmonic mean of precision and recall across all classes. This score is especially informative when evaluating performance on unbalanced datasets.

These metrics allowed us to rigorously evaluate the alignment between the learned clusters and the semantic grouping of entities by research domain.

3.5.2 Global Heterogeneous Deep Graph Clustering evaluation

In global heterogeneous deep graph clustering, we apply a unified clustering process across all node types simultaneously, leveraging joint embeddings that incorporate the structural and semantic diversity of the graph.

3.5.2.1 Results of evaluation metrics

Table 3.2 presents the evaluation metric results for Global Clustering.

Table 3.2: Global clustering results

Method	NMI	ARI	F1	Accuracy
HGT + KMeans	0.4867	0.3274	0.5657	0.4959
HGT + Hierarchical	0.4984	0.3003	0.6145	0.5556
HGT + DEC	0.2767	0.0998	0.0780	0.0838
VGAE + HGT + KMeans	0.6387	0.3737	0.5645	0.5452
VGAE + HGT + Hierarchical	0.6439	0.2241	0.4463	0.5027
VGAE + HGT + Spectral	0.5900	0.1100	0.3374	0.4653
VGAE + HGT + DEC	**0.8553**	**0.6752**	**0.7276**	**0.7281**

The results presented in the previous table have been represented as a histogram in the following Figure 3.1

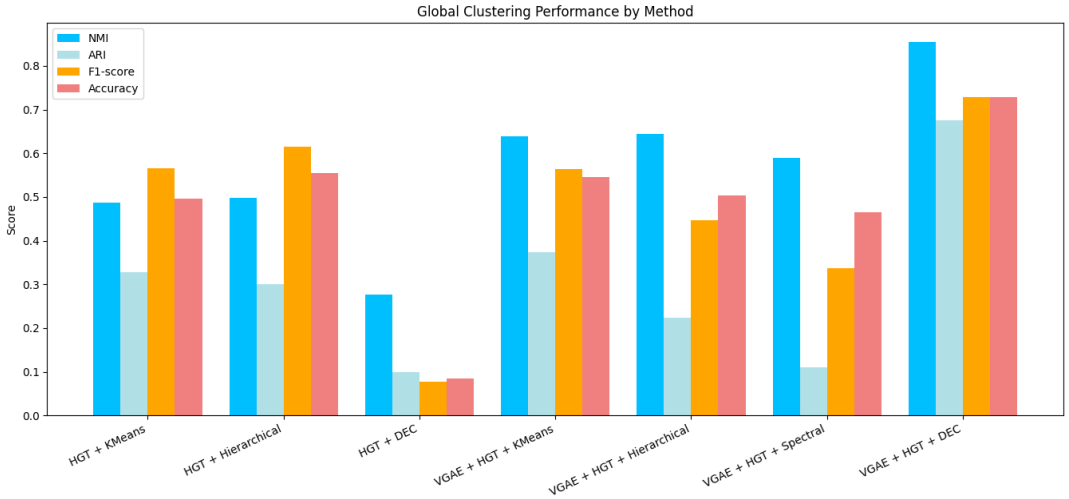

Figure 3.1: Comparison of global clustering metrics

Major Findings On the Deep Clustering of Heterogeneous Graphs and analysis:

- **Impact of VGAE Preprocessing:**
  - Integrating VGAE as a preprocessing step yields a substantial improvement in overall clustering performance.
  - If applied alone, the DEC model is the worst performer:
    - NMI = 0.2767
    - ARI = 0.0998
  - This shows the extreme sensitivity of DEC to the input embeddings quality.

- **Combination of the VGAE + HGT + DEC Model:**
  - The hybrid architecture of VGAE, HGT, and DEC performs very well.The performance is:
    - NMI = 0.8553
    - ARI = 0.6752
  - It implies a series of:
    - +209% in NMI
    - +576% in ARI (versus DEC with no VGAE)
  - The percentage improvements are computed using the standard formula:

$$\text{Percentage Increase} = \left(\frac{\text{New Value} - \text{Old Value}}{|\text{Old Value}|} \right) \times 100$$

  - The significant performance improvements stem from the variational regularization. correction effect of VGAE which enables the model to learn a more discriminative and regularized latent space, which in turn helps support DEC optimization.
- **Classical Methods (K-means, Hierarchical Clustering):**
  - These methods also benefit from VGAE preprocessing, showing moderate but consistent performance improvements.
  - Their performance continues to be way below the VGAE + HGT + DEC architecture.
- **Concluding Summary:**
  - VGAE is important in graph representation enrichment prior to clustering.
  - The combination of VGAE, HGT, and DEC is the most efficient method in global clustering.

3.5.2.2 Visualization of clusters

To qualitatively assess the clustering structure, we present a visualization of the latent embeddings generated by the best-performing architecture, as these results are directly used in the downstream application. Specifically, we apply t-SNE (t-distributed Stochastic Neighbor Embedding) to project the high-dimensional embeddings into a 2D space for visual inspection.
The visualization includes two views: on the left, nodes are colored according to their true domain labels (used as ground truth), and on the right, they are colored by their predicted cluster assignments.
The VGAE + HGT + DEC pipeline provides the most compact and well-separated clusters, demonstrating its ability to learn discriminative and semantically meaningful embeddings for

heterogeneous graph clustering.
Figure 3.2 illustrates the 2D projections of these embeddings:

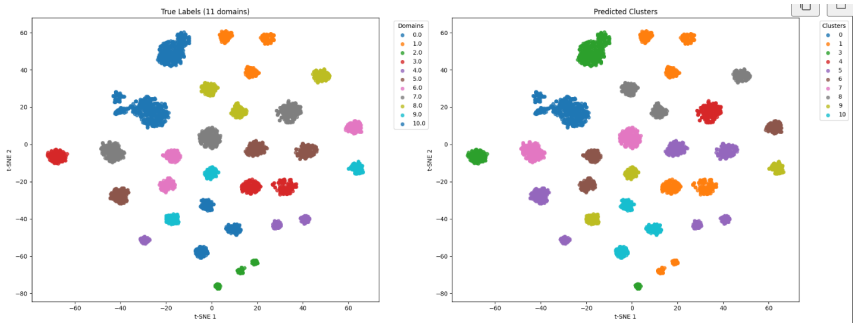

Figure 3.2: Visualization of latent embeddings: (Left) True labels by domain; (Right) Predicted clusters by HGT+DEC

Analysis : From a qualitative perspective, we provide hereafter the general interpretation of the clusters Results:

- **Distinct and well-separated clusters:** The t-SNE projection reveals clearly defined, compact, and non-overlapping clusters, suggesting that the model has effectively captured meaningful structural and semantic patterns within the heterogeneous graph data.
- **High intra-cluster cohesion:** Data points belonging to the same cluster are closely grouped in the latent space, indicating a strong internal consistency within each cluster.
- **Strong inter-cluster separation:** The clusters are clearly separated from one another, reflecting the model's capacity to learn discriminative representations that distinguish different domains or classes.
- **Effective domain discrimination:** The latent embeddings demonstrate a high degree of semantic alignment, enabling the model to differentiate between nodes belonging to different domains.

3.5.3 Clustering Performance evaluation by Node Type

This section gives the comparative evaluation of clustering results for three various categories of nodes. in the heterogeneous graph: **Author**, **Publication**, and **Venue**. The clustering models incorporate a Heterogeneous Graph Transformer with different clustering algorithms both with and without Variational Graph Autoencoder pretraining. Metrics measures that are used for calculation are *Normalized Mutual Information*, *Adjusted Rand Index*, *F1-score*, and *Accuracy*.

3.5.3.1 Author Nodes

Table 3.3 presents the evaluation metric results for nodes of type **Author**.

Table 3.3: Clustering results for Author nodes

Method	NMI	ARI	F1	Accuracy
HGT + KMeans	0.7839	0.6990	0.8363	0.8240
HGT + Hierarchical	**0.9140**	**0.8707**	**0.9421**	**0.9339**
HGT + DEC	0.9014	0.8688	0.8858	0.9099
VGAE + HGT + KMeans	0.8966	0.8645	0.9186	0.9073
VGAE + HGT + Hierarchical	0.8914	0.8611	0.9164	0.9056
VGAE + HGT + Spectral	0.8816	0.8042	0.8663	0.8558
VGAE + HGT + DEC	0.8652	0.8097	0.8868	0.8833

The results presented in the previous table have been represented as a histogram in the following figure 3.3

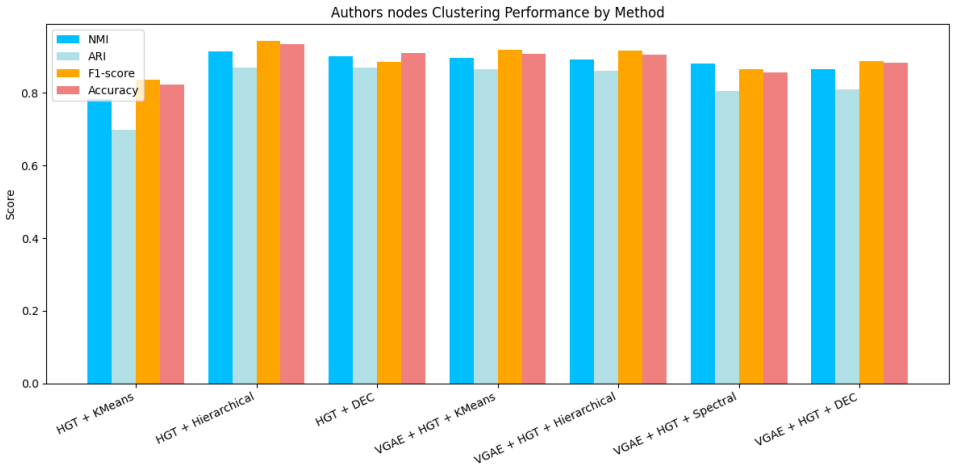

Figure 3.3: Author clustering comparison

Analysis : We observe that the **Author** node type consistently yielded good results across most clustering methods. Among these, Hierarchical Clustering produced the best outcomes in both configurations: when applied directly to HGT embeddings and when combined with VGAE and HGT. These results highlight the model's ability to effectively capture structural and semantic patterns within the author subgraph. VGAE also contributed positively to representation quality, as shown by the high scores obtained with VGAE + HGT + KMeans. In contrast, HGT embeddings clustered using KMeans alone yielded lower performance, indicating the limitations of clustering without prior representation refinement.

3.5.3.2 Publication Nodes

Table 3.4 presents the evaluation metric results for nodes of type **Publication**.

Table 3.4: Clustering results for Publication nodes

Method	NMI	ARI	F1	Accuracy
HGT + KMeans	0.7549	0.5172	0.6353	0.6514
HGT + Hierarchical	0.8378	0.6304	0.8208	0.7789
HGT + DEC	0.8202	0.6448	0.7248	0.7748
VGAE + HGT + KMeans	0.8225	0.6712	0.7944	0.7623
VGAE + HGT + Hierarchical	**0.8546**	**0.6923**	**0.8750**	**0.8087**
VGAE + HGT + Spectral	0.8219	0.6416	0.7714	0.7450
VGAE + HGT + DEC	0.2383	0.0956	0.3177	0.3507

The following Figure 3.4 illustrates the results from the previous table in the form of a histogram:

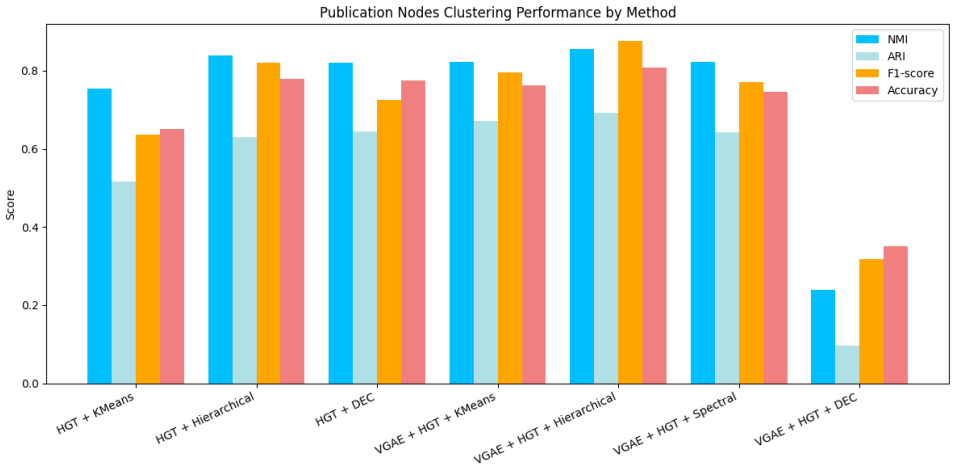

Figure 3.4: Publication clustering comparison

Analysis : The publication node clustering exhibited moderately strong results. The best method was **VGAE + HGT + Hierarchical Clustering**,with the following results:

- NMI = 0.8546
- ARI = 0.6923
- F1-score = 0.8750
- Accuracy = 0.8087

For publication nodes, methods incorporating VGAE consistently achieved the highest clustering quality. Although the combination of **HGT + Hierarchical Clustering** also produced strong results, its performance remained below that of VGAE-based approaches. In contrast, the **VGAE + HGT + DEC** configuration yielded the weakest outcomes, with significantly lower clustering quality.

3.5.3.3 Venue Nodes

Table 3.4 presents the evaluation metric results for nodes of type **Venue**.

Table 3.5: Clustering results for Venue nodes

Method	NMI	ARI	F1	Accuracy
HGT + KMeans	0.1251	0.0390	0.5898	0.4297
HGT + Hierarchical	0.7997	0.5004	0.7500	0.7023
HGT + DEC	0.0062	0.0014	0.2204	0.2389
VGAE + HGT + KMeans	0.7968	0.5978	0.8185	0.7646
VGAE + HGT + Hierarchical	**0.8099**	**0.6037**	**0.8225**	**0.7692**
VGAE + HGT + Spectral	0.7642	0.5507	0.7291	0.7042
VGAE + HGT + DEC	0.7334	0.5173	0.6910	0.7009

The following Figure 3.5 illustrates the results from the previous table in the form of a histogram:

Figure 3.5: Venue clustering comparison

Analysis : Venue nodes posed the toughest clustering problem, generally with lower scores across all methods. However, **VGAE + HGT + Hierarchical Clustering** and **VGAE + HGT + KMeans** produced reasonably acceptable scores:

- NMI ≈ 0.81
- ARI ≈ 0.60
- F1-score ≈ 0.82
- Accuracy ≈ 0.77

The **HGT + Hierarchical Clustering** also produced good scores, however scores were slightly lower than VGAE-based methods. In contrast, **HGT + DEC** ultimately produced

terrible results , indicating how difficult clustering venue nodes is without rich embeddings and appropriate structure context.

3.5.3.4 Summary of Analysis

Across all experiments, the following trends were observed:

- Pretraining using VGAE significantly improves clustering inference, especially when used with HGT.
- Hierarchical clustering strategies outperform flat clustering strategies (like KMeans and DEC).
- Author nodes are the most structurally cohesive and thus, the easiest to cluster, publications follow, venue nodes are the least cohesive and often required further modeling strategy.

These findings emphasize the importance of combining representation learning (e.g., VGAE) with structure-aware models (e.g., HGT)as well as clustering strategies that are appropriate for the complexity of the nodes interactions in a heterogeneous graph.

3.5.3.5 Visualization of clusters

We present the cluster visualizations for all node types. These plots highlight the semantic organization and latent space separability achieved by our pipeline.
This visualization highlights the best-performing configuration, which is also employed in the final application.
Figure 3.6 presents the 2D t-SNE projections of the latent embeddings corresponding to the **Author** node type, obtained from the HGT model combined with hierarchical clustering. In the visualization, the left side displays nodes colored according to their true domain labels (ground truth), while the right side shows the predicted cluster assignments produced by the model.

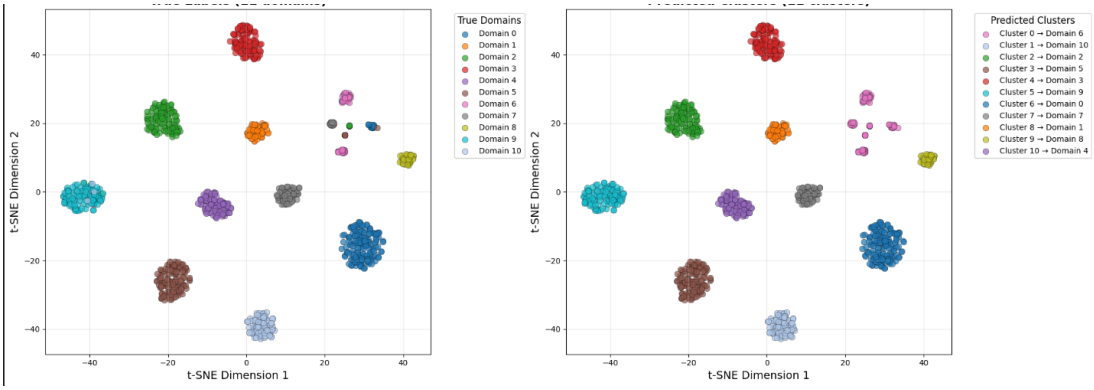

Figure 3.6: Visualization of Author latent embeddings: (Left) True labels by domain; (Right) Predicted clusters by HGT+hierarchical clustering

Figure 3.7 presents the 2D t-SNE projections of the latent embeddings corresponding to the **Publication** node type, obtained from the HGT model combined with VGAE model and hierarchical clustering. In the visualization, the left side displays nodes colored according to their true domain labels (ground truth), while the right side shows the predicted cluster assignments produced by the model.

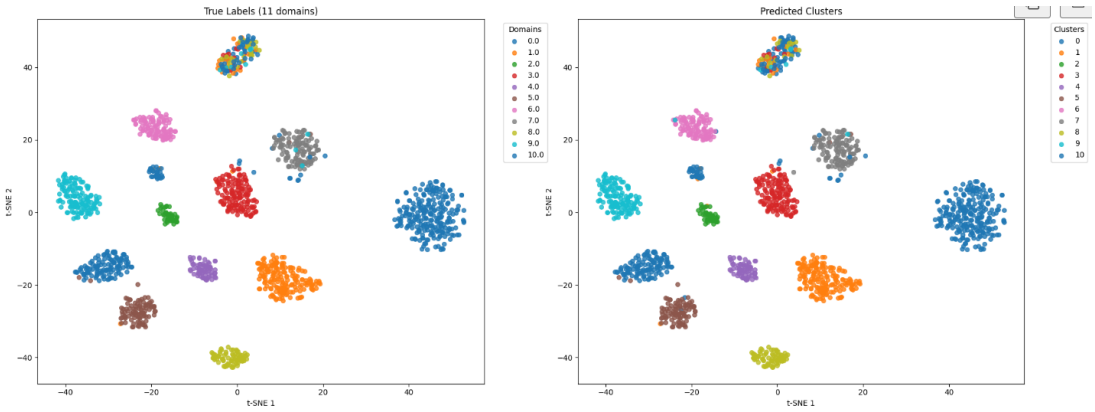

Figure 3.7: Visualization of Publication latent embeddings: (Left) True labels by domain; (Right) Predicted clusters by VGAE + HGT + hierarchical clustering

Figure 3.8 presents the 2D t-SNE projections of the latent embeddings corresponding to the **Venue** node type, obtained from the HGT model combined with VGAE model and hierarchical clustering. In the visualization, the left side displays nodes colored according to their true domain labels (ground truth), while the right side shows the predicted cluster assignments produced by the model.

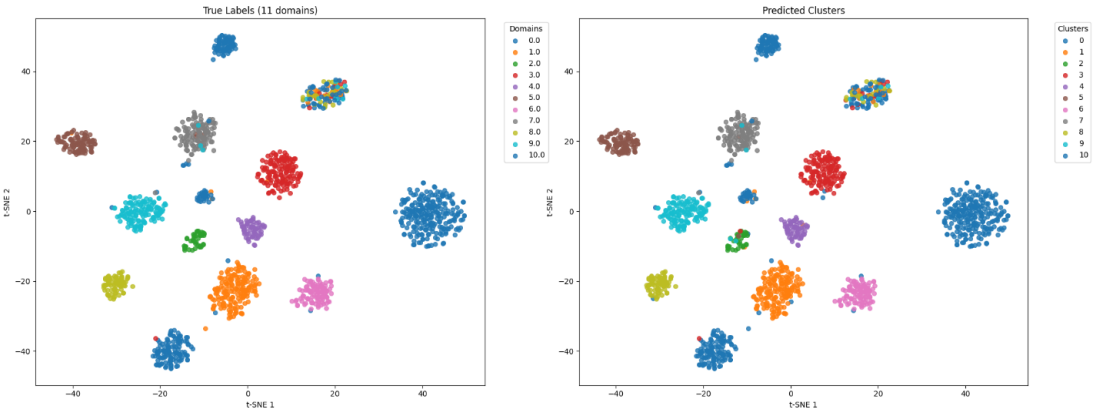

Figure 3.8: Visualization of Venue latent embeddings: (Left) True labels by domain; (Right) Predicted clusters by VGAE + HGT + hierarchical clustering

Analysis : From a qualitative analysis, we provide hereafter the general interpretation of the clusters Results:

- **Clear Cluster Separation:** Most predicted clusters exhibit tight cohesion and minimal overlap, indicating high intra-cluster similarity and good inter-cluster separability.

- **Semantic Consistency of Domains:** The preservation of domain structures in the latent space demonstrates the model's ability to generate semantically meaningful embeddings, enabling accurate unsupervised domain-level clustering in heterogeneous graphs.
- **High Alignment with Ground Truth:** Visual inspection reveals high correspondence between predicted clusters and known domain labels, which shows that the model accurately reflects the underlying semantic structure in data.
- **Detection of Cross-Cluster Anomalies:** The model is able to identify nodes with boundary or wrong-cluster embeddings, which may suggest semantic ambiguity or structural overlap—a valuable feature for gentle inspection of real graphs.

The experimental results indicate that the anticipated clustering pipeline can effectively gain valuable clusters based on structural and semantic patterns in a heterogeneous graph.
The consistency of predicted labels with true labels substantiates the validity of learnt embeddings and clustering approach.

3.6 From Clustering to Recommendation

Clustering is a very efficient method for finding the intrinsic structure and patterns of data. They find their real worth when interactions in heterogeneous graphs are being examined. employed for utilitarian purposes. One very important application is in recommendation systems, which transform grouped data into user-centered recommendations by detentifying similar entities by virtue of acquired similarities.Thanks to deep graph clustering results, these systems offer more accurate, diverse, and context-dependent recommendations. In the following section, we show how these clusters support two complementary recommendation systems: a global system and a type-based system.

3.6.1 Enhancing Recommendations with Semi-Supervised Clustering

Academic recommendation systems commonly must first cluster entities such as authors, publications, or venues. The challenge is the incomplete labeling: while a bin of items (e.g., Computer Science papers) might have labeled items, many are unlabeled. Supervised methods require labeled items, while unsupervised methods completely disregard labels. Both approaches struggle in settings where labeled and unlabeled items coexist and where the data is always changing and incomplete, as is the case with academic domains.
To overcome this barrier, we present a semi-supervised clustering method that accepts both labeled and unlabeled items. By utilizing existing labeling information, we can create meaningful clusters without comprehensive labeling, and we offer a solid framework for a dynamic dataset

like an academic dataset.
Moreover, in the context of recommendation systems, users can be broadly divided into two categories:

- those who already have an account and a research history (e.g., published papers, specific topics of interest), and thus their associated class or research domain is known;
- and those who are new to the system, without prior activity, and therefore without any assigned class.

Semi-supervised clustering effectively bridges this gap by propagating known user interests to similar, unlabeled users or items. This enables the system to make relevant recommendations even for new users, improving personalization from the outset and supporting cold-start scenarios.
This makes semi-supervised clustering an essential component for scalable and adaptive recommendation systems in academic environments.

Implementation Details:
This section presents the core implementation strategies used to enable semi-supervised clustering :

- **Label Propagation Algorithm:** For nodes with known labels, we apply the Label Propagation Algorithm, a classic graph-based semi-supervised learning method.

  It spreads label information through the graph , if two nodes are strongly connected and one has a label, the other is likely to share the same label.

  This algorithm is key in our semi-supervised setup, as it helps to exploit existing labels without requiring full supervision.

- **Initial Clustering using Random Forest:** We use a Random Forest classifier not for classification, but to estimate initial cluster memberships. By feeding labeled and partially labeled nodes to the forest, we obtain probabilistic outputs (soft assignments) instead of hard classes. These probabilities indicate the likelihood of a node belonging to different latent clusters, providing a flexible initialization for downstream clustering.

  This is another critical aspect of our semi-supervised clustering approach: we use supervised models like Random Forests not to produce final predictions, but to guide and initialize the clustering process.

- **Contrastive Learning :** We employ contrastive learning to refine the embeddings. The idea is to bring similar entities closer in the latent space while pushing dissimilar ones further apart.

Results obtained with the semi-supervised approach:

- **For Publications:**here are the results obtained for the publications clustering:

  - NMI = 0.9385
  - ARI = 0.9340
  - F1-score = 0.9720
  - Accuracy = 0.9723

  For a better understanding of the cluster distribution, please refer to Figure 3.9, which illustrates the cluster visualization.

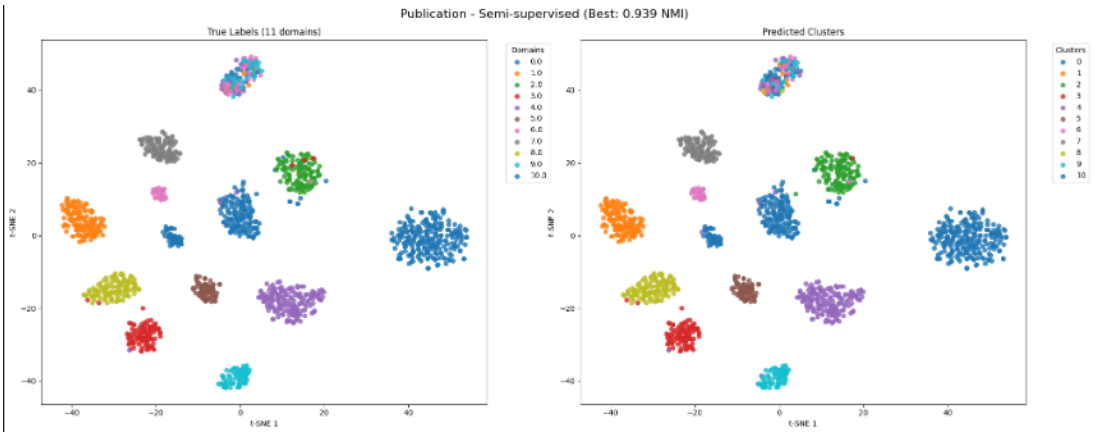

Figure 3.9: Publication Semi-Suppervised Clustering

- **For Authors:**here are the results obtained for the authors clustering:

  - NMI = 0.9475
  - ARI = 0.9483
  - F1-score = 0.9757
  - Accuracy = 0.9760

  For a better understanding of the cluster distribution, please refer to Figure 3.10, which illustrates the cluster visualization.

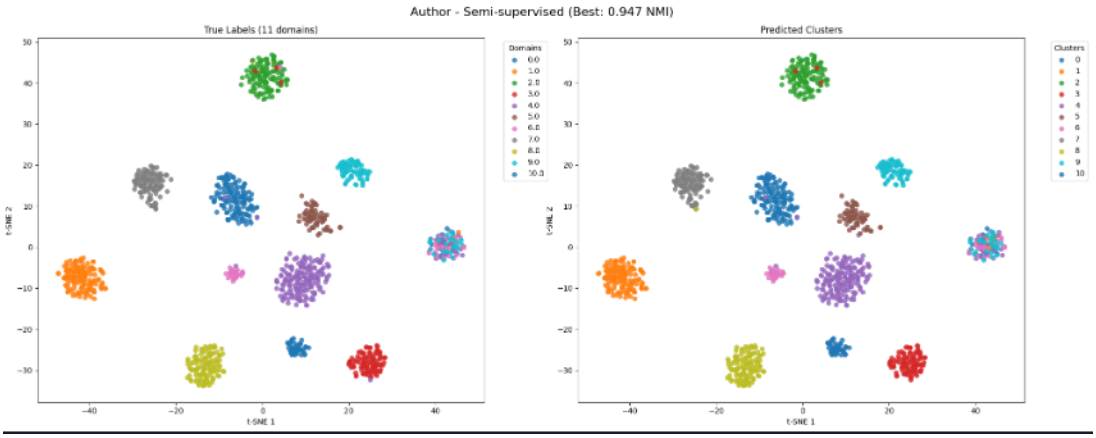

Figure 3.10: Author Semi-Suppervised Clustering

- **For Venues:**here are the results obtained for the venues clustering:
  - NMI = 0.9490
  - ARI = 0.9441
  - F1-score = 0.9770
  - Accuracy = 0.9775

  For a better understanding of the cluster distribution, please refer to Figure 3.11, which illustrates the cluster visualization.

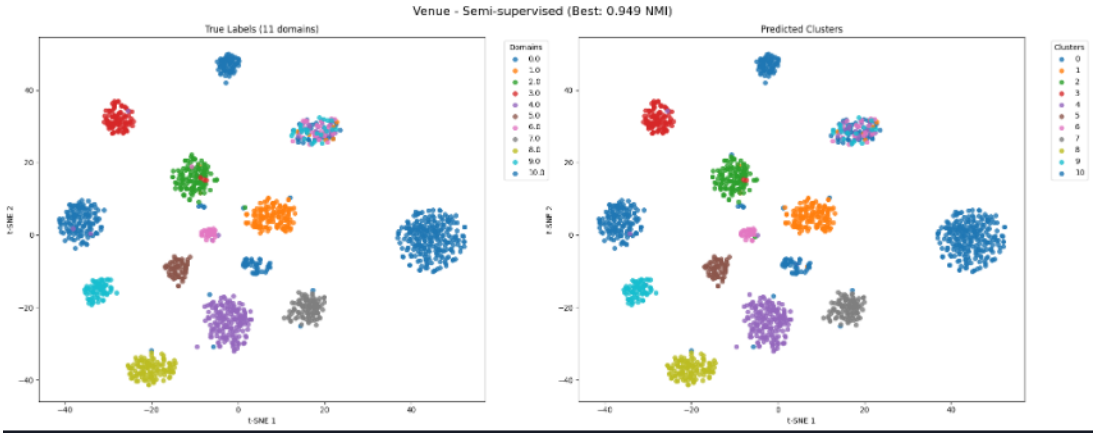

Figure 3.11: Venue Semi-Suppervised Clustering

Comparison with the Global clustering:here are the results obtained for the global clustering:

- NMI = 0.9336
- ARI = 0.8828

- F1-score = 0.9450
- Accuracy = 0.9461

For a better understanding of the cluster distribution, please refer to Figure 3.12, which illustrates the cluster visualization.

Figure 3.12: Semi-Suppervised Global Clustering

This semi-supervised strategy significantly enhances clustering results, especially in heterogeneous and complex graph structures. By leveraging limited labeled data, it enables more coherent, accurate, and semantically meaningful groupings. This improvement directly benefits downstream tasks such as recommendation, making the approach particularly valuable in real-world scenarios where annotated data is scarce.

3.6.2 Deep graph clustering exploitation

Clustering data in our project is the foundation for higher-level recommendation mechanisms in linked data. Once the data is clustered, we design two complementary recommendation systems:

1. Global Recommendation System:
   This system relies on global clustering, where nodes of different types (e.g., authors, publications, venues) are grouped together based on domain similarity using Cosine similarity in the embedding space. This enables cross-type recommendations: for example, given a selected author, the system can recommend related publications and venues that belong to the same or nearby clusters. A top-k similarity approach is used to retrieve the most relevant items.

2. Type-Based Recommendation System:
   This system performs type-specific clustering, grouping nodes of the same type (e.g., only authors or only publications) by domain. As a result, it offers recommendations within the same entity type, such as recommending authors similar to a selected author.

These two strategies demonstrate how clustering supports flexible and meaningful recommendations:

- Either by bridging types (cross-type recommendation)
- Or by preserving type constraints (type-based recommendation)

Thus, our approach leverages the structure of the linked data graph to provide intelligent, domain-aware recommendations in dynamic and heterogeneous academic data environments.

3.7 Application Interfaces

In this section, we present the main interfaces of our application.

- **Overview Interface:** The figure above in Figure 3.13 displays the application's main interface, which includes a set of navigation buttons positioned at the top. Each button corresponds to a specific functional module:
  - **Overview**: Provides a comprehensive summary of the project and delineates the key phases of the deep graph clustering pipeline.
  - **RDF Graph**: Enables users to examine the construction process of the RDF graph derived from the DBLP dataset.
  - **Encoding**: Illustrates the methodology for encoding node attributes and inter-node relationships using natural language processing models.
  - **DGL Graph**: Represents the conversion of the RDF graph into a format compatible with the Deep Graph Library.
  - **Embeddings**: Details the process through which graph neural networks are employed to generate node embeddings.
  - **Clustering**: Facilitates access to the implementation of deep clustering algorithms on the graph structured data.
  - **Visualizations**: This interface displays the visualizations of the clustering results.
  - **Type-Based Rec** and **Global Rec**: Direct users to two distinct recommendation systems constructed upon the clustering results the first provides type specific recommendations, while the second delivers global suggestions.

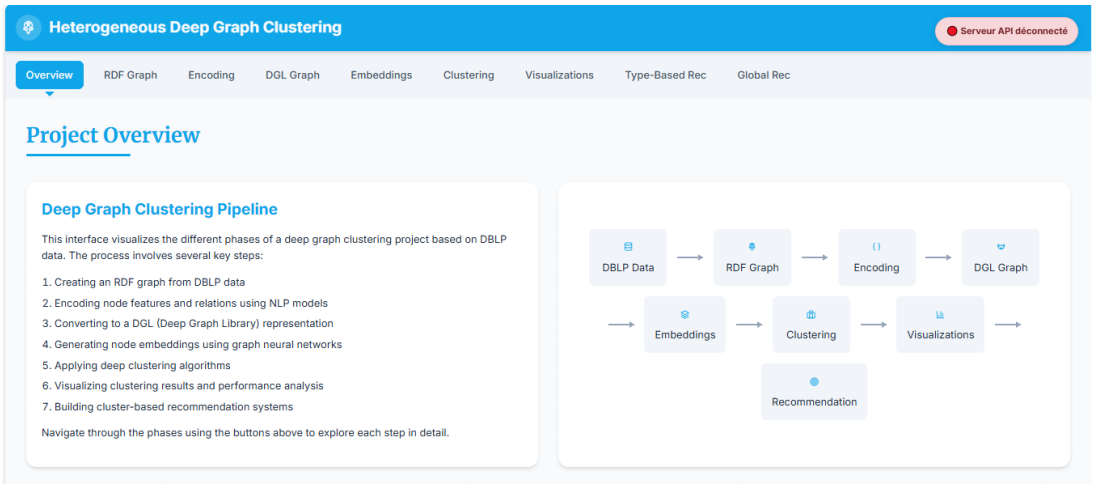

Figure 3.13: Overview Interface

This interface systematically guides users through the successive stages of the pipeline, thereby enhancing their comprehension and enabling effective interaction with the various components of the deep graph clustering framework.

- **RDF Graph Interface :** The interface in Figure 3.14 delineates the structural composition of the RDF graph used in this study. The "Node Types" panel on the left categorizes the graph into four principal node types Author, Publication, Venue, and Domain, each of which can be expanded to reveal their respective attributes and relational connections. In contrast, the "Extract from DBLP Dataset" panel on the right provides exemplar RDF representations for each node category, effectively demonstrating the structuring of empirical data within the graph. Together, these elements enable a thorough analysis of the principal entities, their attributes, and the methodology by which the DBLP dataset is encoded using RDF.

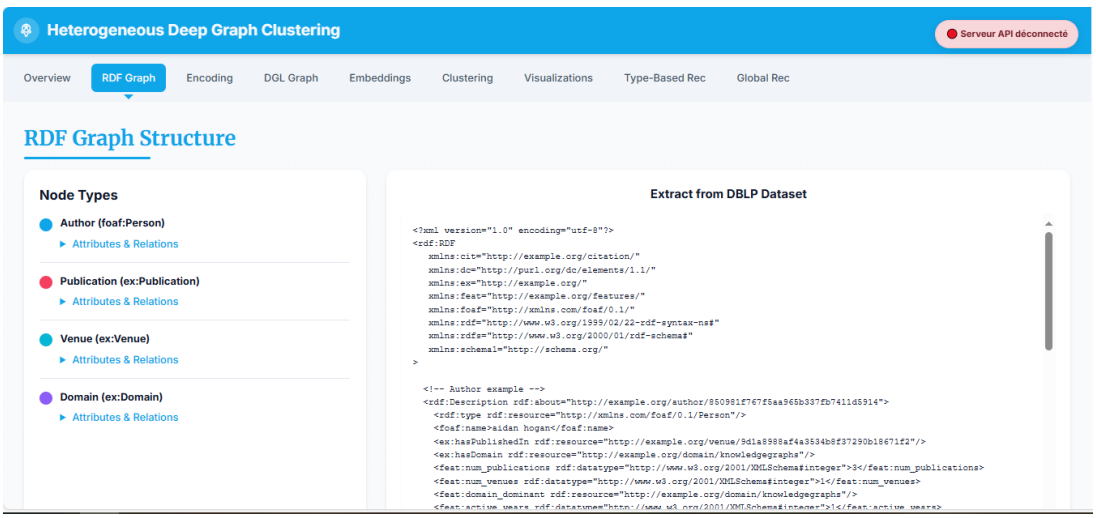

Figure 3.14: RDF Graph Interface

- **The feature encoding interface :** The Feature Encoding Interface shown in Figure 3.15 provides a comprehensive description of the preprocessing techniques used on node features before inputting them into graph learning models. It outlines the techniques used for encoding text and numerical features, including using transformer-based models to encode text and different scaling methods to encode numerical features. Besides, the interface explains the dimensionality of the eventual embeddings and emphasizes the central role of this process in the development of coherent and semantically meaningful representations that are crucial in the analysis of heterogeneous graphs.

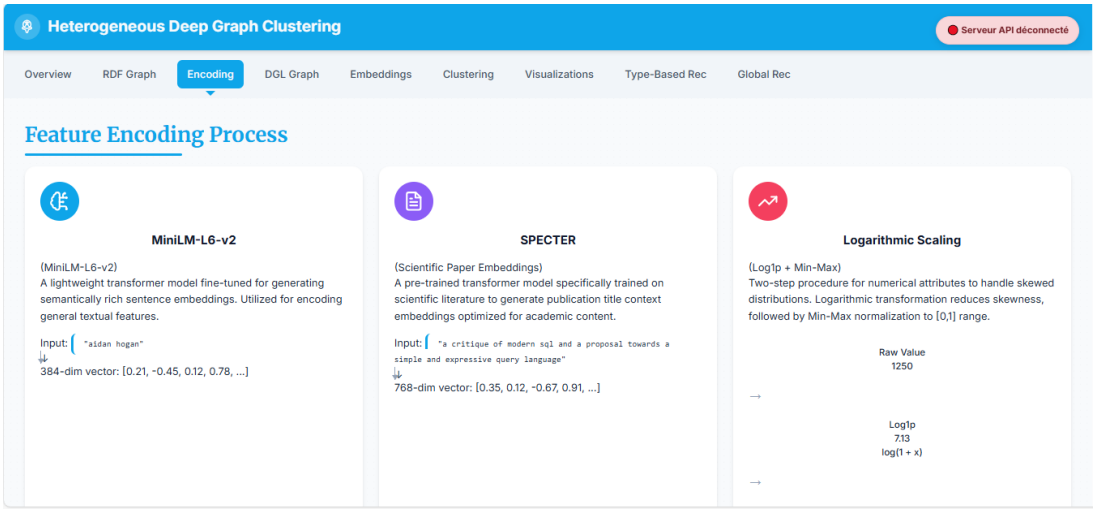

Figure 3.15: The feature encoding interface

- **DGL Graph Interface :** This interface in Figure 3.16 enables visualization of a representative sample.of the DGL graph utilized in this project. It is designed to give a comprehensive insight rather more so than a general characterization of the graph's architecture.

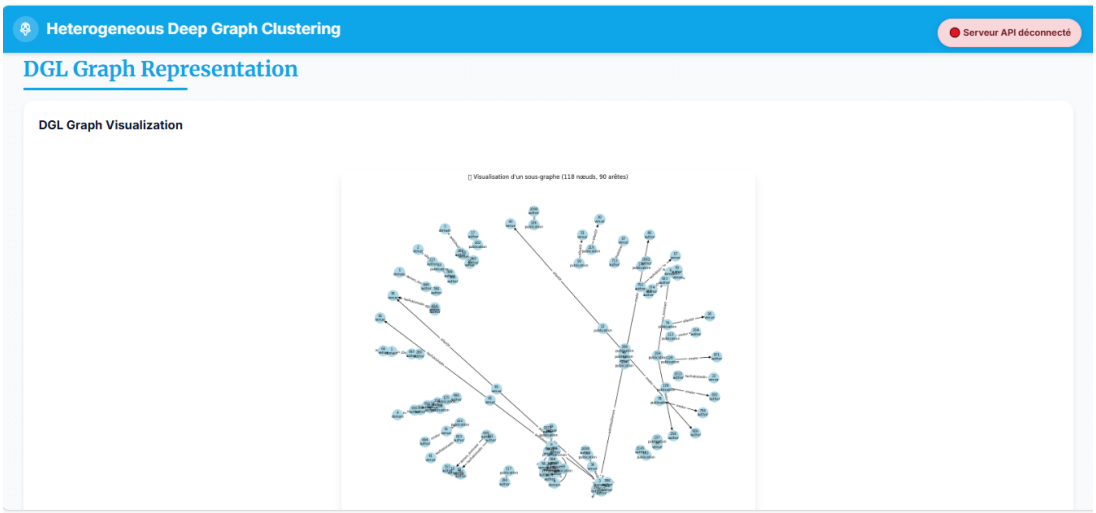

Figure 3.16: DGL Graph Interface

- **Embeddings Interface :** This interface in Figure 3.17 illustrates the sequential deep learning pipeline employed to produce node embeddings. It combines two models: first, a Variational Graph Autoencoder to encode the initial node representations by learning latent Probabilistic embeddings are then strengthened using a heterogeneous method. Graph Transformer with contrastive learning for enhancing the quality and the semantic coherence of the ultimate representations. The interface offers a wide This architecture gives an overview of its role in the context of representation learning.

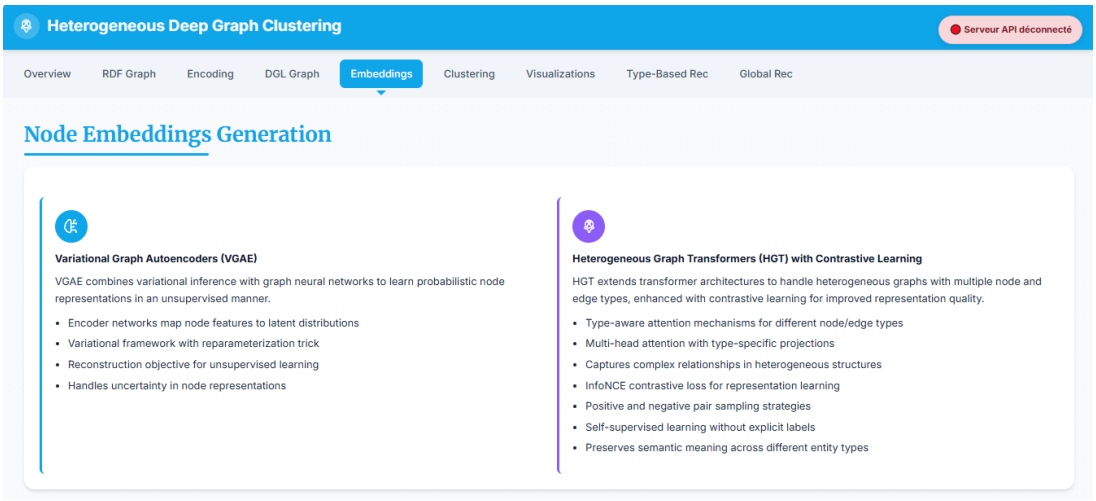

Figure 3.17: Embeddings Interface

- **Clustering Interface :** This interface in Figure 3.18 enables users to perform clustering on the embeddings derived. It offers two main modes, which are accessible from two distinct buttons: **Explore Type-Based** and **Explore Global**. Clicking on either button redirects the user to a dedicated results interface 3.19, where the clustering metrics are presented in structured tables.

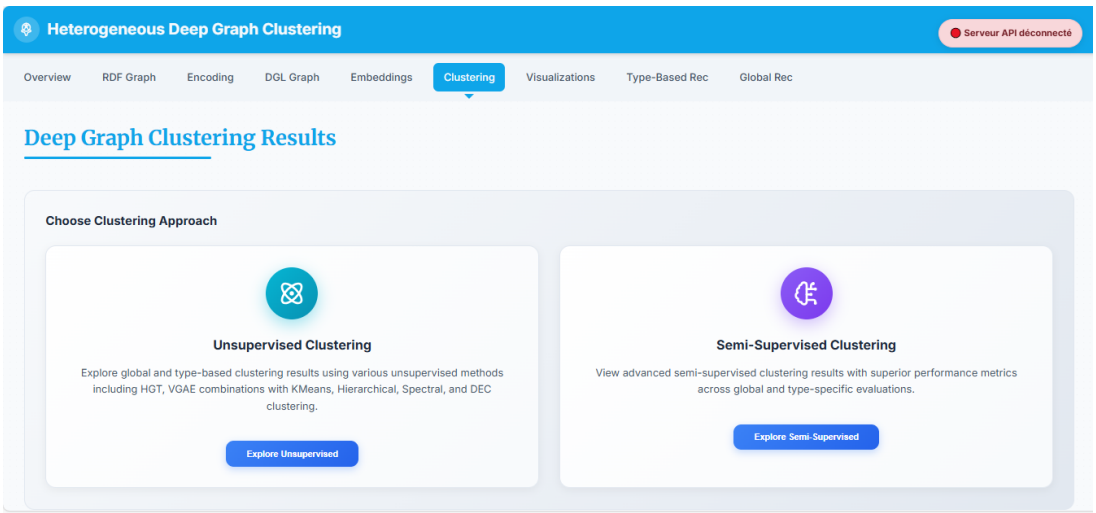

Figure 3.18: Clustering Interface

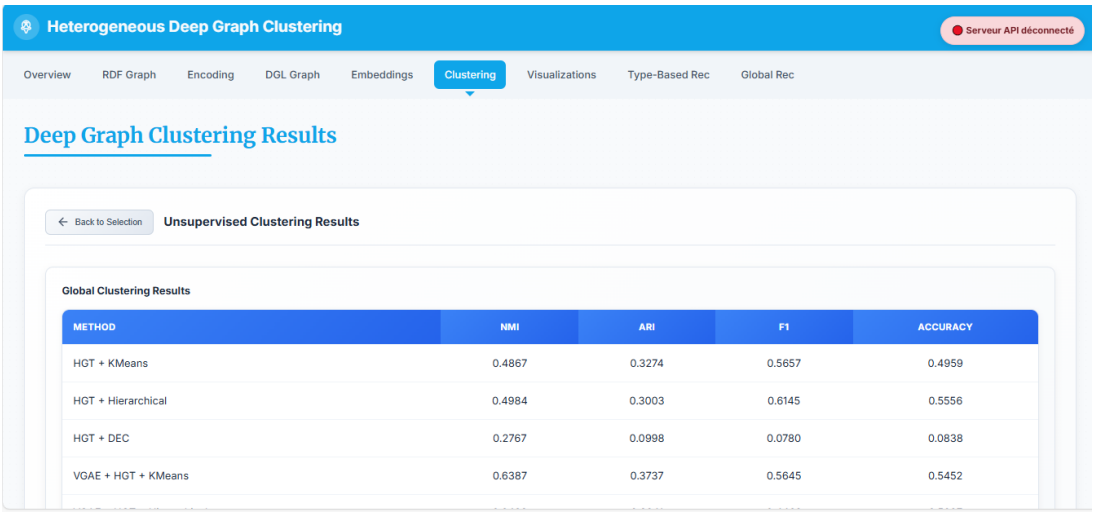

Figure 3.19: Metrics results interface

- **Visualizations Interface:** The interface in Figure 3.20 presents two primary buttons, namely "Unsupervised" and "Semi-Supervised", which enable the user to select the desired clustering mode. Upon selecting the Unsupervised mode, two supplementary buttons "Global Clustering" and "By Type Clustering" are displayed, allowing the user to manually choose the specific type of clustering visualization to be rendered. In contrast, selecting the Semi-Supervised mode automatically reveals both the global and type-based clustering visualizations, without necessitating any further user interaction.

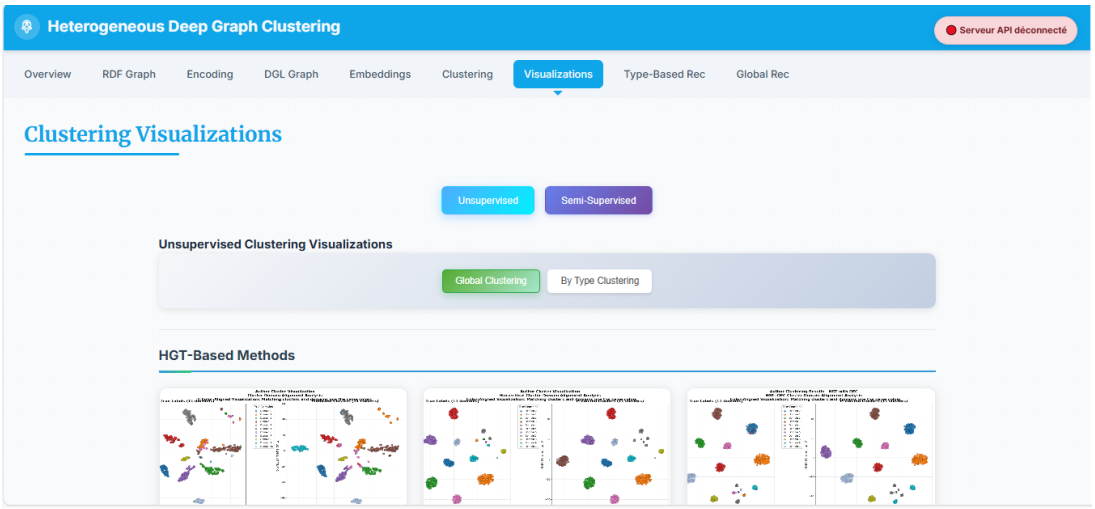

Figure 3.20: Visualizations Interface

- **Type-Based Recommendation Interface :** This interface in Figure 3.21 enables users to receive recommendations within the same node type in a heterogeneous graph. It supports three types of nodes: authors, publications, and venues. Users begin by selecting the target entity type from a dropdown list. They then provide a specific input, such as the name of a

writer, the title of a publication, or the label of a venue. The system also allows users to specify the number of recommendations they wish to receive. Once the query is submitted, the interface gives recommendations .

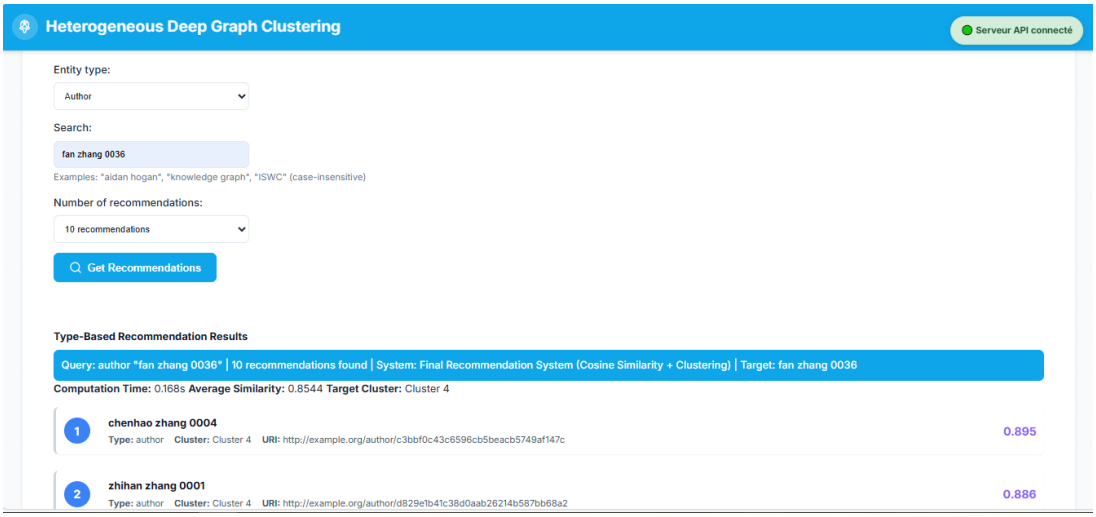

Figure 3.21: Type-Based Recommendation Interface

- **Gloal Recommendation Interface :** This interface in Figure 3.22 is similar to the type-based recommendation interface; however, it operates on global clusters, which may include nodes of different types. Unlike the type-based approach, where recommendations are limited to a single entity category, the global clustering approach allows for recommendations between different node types. Since each cluster can contain a mix of authors, publications, and venues, the system is capable of suggesting interrelated entities,for example, recommending relevant publications or venues based on a selected author.

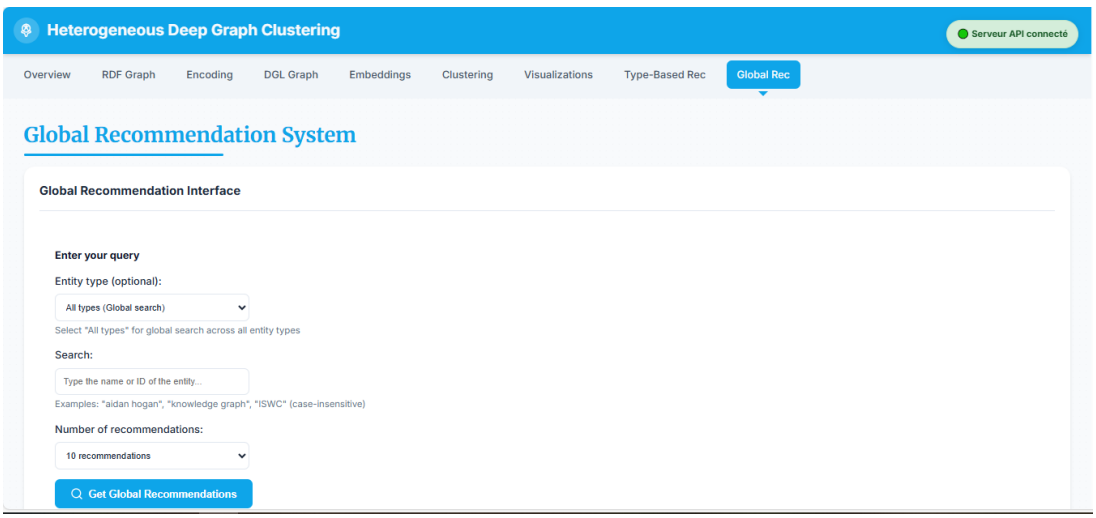

Figure 3.22: Gloal Recommendation Interface

3.8 Conclusion

In this chapter, we have described the tools and environment used for the development of our system. Subsequently, we conducted the testing phase to evaluate the system, focusing on interpreting the obtained results through a combination of quantitative evaluation metrics and qualitative cluster visualizations. Finally, we presented the user interfaces developed for the application.

General Conclusion

The Semantic Web and Linked Data, enables the representation of knowledge as richly interconnected graphs of heterogeneous entities. In this context, deep graph clustering emerges as a powerful technique for revealing hidden semantic and structural patterns by jointly learning from both the content of the data and the relationships that link it.
We propose a solution that combines the Heterogeneous Graph Transformer for learning type-aware node embeddings with a Variational Graph AutoEncoder for preprocessing node features. The resulting embeddings are then subjected to multiple clustering strategies, including KMeans, hierarchical clustering, and neural clustering methods. Furthermore, contrastive learning is integrated to enhance the semantic discriminability of the embeddings.
This deep clustering pipeline is specifically designed to capture the rich and nuanced semantic structures embedded in complex, sparse, and noisy knowledge graphs. In addition to structural analysis, the learned embeddings are leveraged to implement a recommendation system capable of suggesting authors, publications, and venues based on latent semantic similarities. This recommendation component enhances the utility of the system in digital libraries and scholarly communication platforms.
Experimental results obtained on partial RDF graphs such as subsets of the DBLP dataset demonstrated the system's ability to effectively cluster entities based on implicit semantic attributes. These findings highlight the potential of deep graph clustering as a scalable and interpretable solution for knowledge graph analysis.

Future work includes enhancing the pipeline's performance by evaluating the system on large-scale datasets and adapting the proposed approach to other Linked Data domains, in order to assess its scalability and generalizability. Finally, deploying the system in a cloud-based environment is planned to enable real-time processing.

Bibliography

[1] Berners-Lee, T. and Swick, R. R. (2006). The semantic web layer cake. `https://www.researchgate.net/figure/The-Semantic-Web-Layer-Cake_fig1_50251932`. Figure illustrating Semantic Web architecture.

[2] Bizer, C., Heath, T., and Berners-Lee, T. (2023). Linked data-the story so far. In *Linking the World's Information: Essays on Tim Berners-Lee's Invention of the World Wide Web*, pages 115–143.

[3] Dhulekar, K. and Devrankar, M. (2019). A review on semantic web. *Int. J. Eng. Technol. Manag. Res*, 6:22–28.

[4] Espinasse, B. (2019). Introduction au web sémantique.

[5] Harb, I. H., Abdel-Mageid, S., Farahat, H., and Farag, M. (2010). Enhanced semantic web layered architecture model. *New Aspects of Applied Informatics, Biomedical Electronics & Informatics and Communications, WSEAS, Taipei*.

[6] Hogan, A. and Hogan, A. (2020). *Web of data*. Springer.

[7] Li, Q., Liu, H., Jiang, R., and Wang, T. (2024a). Incorporating higher-order structural information for graph clustering. In *International Conference on Database Systems for Advanced Applications*, pages 507–517. Springer.

[8] Li, Y., Chen, J., Chen, C., Yang, L., and Zheng, Z. (2024b). Contrastive deep nonnegative matrix factorization for community detection. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6725–6729. IEEE.

[9] Liang, H. and Gao, J. (2021). Wasserstein adversarially regularized graph autoencoder. *Neurocomputing*, 541:126235.

[10] Liu, Y., Tu, W., Zhou, S., Liu, X., Song, L., Yang, X., and Zhu, E. (2021). Deep graph clustering via dual correlation reduction. *arXiv preprint arXiv:2112.14772*.

[11] Liu, Y., Xia, J., Zhou, S., Yang, X., Liang, K., Fan, C., Zhuang, Y., Li, S. Z., Liu, X., and He, K. (2022). A survey of deep graph clustering: Taxonomy, challenge, application, and open resource. *arXiv preprint arXiv:2211.12875*.

[12] Mondal, R., Ignatova, E., Walke, D., Broneske, D., Saake, G., and Heyer, R. (2024). Clustering graph data: the roadmap to spectral techniques. *Discover Artificial Intelligence*, 4(1):7.

[13] Navarange, R. B., Kate, R. R., Salunkhe, C., Sonawne, K., and Katale, V. (2023). Semantic web. *International Journal of Novel Research and Development*, 8(3):651–653.

[14] Ngomo, A.-C. N., Auer, S., Lehmann, J., and Zaveri, A. (2014). Introduction to linked data and its lifecycle on the web. *Reasoning Web. Reasoning on the Web in the Big Data Era: 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings 10*, pages 1–99.

[15] Shi, M., Tang, Y., Zhu, X., Wilson, D., and Liu, J. (2020). Multi-class imbalanced graph convolutional network learning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*.

[16] Sun, L., Huang, Z., Peng, H., Wang, Y., Liu, C., and Yu, P. S. (2024). Lsenet: Lorentz structural entropy neural network for deep graph clustering. *arXiv preprint arXiv:2405.11801*.

[17] Wang, C., Pan, S., Hu, R., Long, G., Jiang, J., and Zhang, C. (2019). Attributed graph clustering: A deep attentional embedding approach. *arXiv preprint arXiv:1906.06532*.

[18] Wu, B., Ding, S., Xu, X., Guo, L., Ding, L., and Wu, X. (2024). Synergistic deep graph clustering network. *arXiv preprint arXiv:2406.15797*.

[19] Xu, R. and Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678.