

POLITECNICO DI TORINO

Masters's Degree in Computer Engineering



Master's Degree Thesis

Graph Neural Networks on Semantic Trajectories

Supervisors

Prof. Paolo Garza

Candidate

Ali Elrida Shahimi

July 2026

Abstract

Semantic Environmental Trajectories of Territories (SETT) represent the environmental evolution of territorial units over time by combining their spatial, temporal, and semantic information. In the TRACES project [34], these territory trajectories are represented as Knowledge Graphs, where each territory is characterized by several thematic trajectories such as water, temperature, vegetation, snow, and urbanization. This representation is rich and expressive through KGs, but it raises several challenges for graph representation learning, as the data includes heterogeneous relations, temporal intervals, spatial descriptors, and thematic measurements in each thematic trajectory. This report studies how Graph Neural Networks (GNNs) can be used to learn meaningful representations from SETT Knowledge Graphs. The work first provides a state-of-the-art analysis of GNN approaches for Knowledge Graph representation learning, starting from static GNNs and moving toward spatial, temporal, and spatiotemporal models. Based on this study, an adapted architecture, called SETT-GNN, is proposed. The model is inspired by an existing spatiotemporal neural architecture, THAM [24], and adapted to fit the structure of SETT Knowledge Graphs. The proposed pipeline transforms SETT Knowledge Graphs into territory-episode snapshots. Each snapshot is encoded using a relation-aware GNN, while the sequence of episode representations is processed using temporal and positional encodings followed by a Transformer encoder. The final output is a trajectory-level embedding for each territory trajectory. The learned representations are discussed with respect to possible downstream tasks, including trajectory classification, clustering, and trajectory similarity. The experimental part mainly focuses on trajectory similarity, using self-supervised training, data ablation studies, and model ablation studies to evaluate whether the learned embeddings capture meaningful information about territory trajectories. Overall, this work provides a first step toward applying GNN-based representation learning to Semantic Environmental Trajectories of Territories and opens perspectives for future work on other semantic trajectory datasets, such as semantic life trajectories.

Acknowledgements

This work is not only the result of a semester-long research project; it also summarizes a journey of five years, full of dedication and adventures.

As this work comes to an end, I would like to express my sincere gratitude to my supervisors at LIG Professor Jérôme Gensel, and Professor Marlène Villanova for giving me the opportunity to discover how research could be done from scratch. Under the supervision of you and with Professor Camille Bernard, I learned a lot not only about the project topic itself, but also on my academic side, and the process of developing ideas into meaningful contributions. I'm grateful for your guidance, feedback, and the many meetings carried through this internship. I look forward that this work would give contributions in future for STORIES team.

I also want to thank the members of the Jury Professor Sergi Pujades, and Professor Cassia Trojahn Dos Santos, for your time making this defense possible.

To Politecnico di Torino and to the beautiful city of Turin, I'm deeply grateful for the four years I spent in this special university and environment; grateful for all the memories there, all hard and happy times I spent. I'm also grateful for the opportunity I had to do this Erasmus year in Grenoble, this would not happen if I was not a student in Polito. Also, I would like to sincerely thank my supervisor at Polito, Professor Paolo Garza, for accepting this internship and his follow-up and support throughout this period.

Thanks to all my friends and colleagues I've met through this journey. It was a privilege to encounter so many different cultures, languages, perspectives, and stories. I truly appreciate each one of you, and I am grateful for the moments, conversations, and memories we shared.

To my family, my brother, sister, and my dear parents: you are the source of my motivation and the reason behind every success. To my parents, thank you for your sacrifices, your encouragement, your prayers, and every detail you followed and cared about throughout this journey. This work is dedicated for you because without you I would not be here.

Finally, thank you for being by my side in this journey. This chapter is coming to an end, but the journey did not end yet, many adventures are yet coming so stay tuned. I promise that I will always give my best, for you and myself.

Until the next challenge.

Table of Contents

1	Introduction	2
1.1	Motivation	2
1.2	Problem statement	3
1.3	Main challenges	4
1.4	Contribution.....	4
1.5	Report outline	6
2	State-of-the-Art.....	7
2.1	Knowledge Graphs	7
2.2	Semantic Environmental Trajectories of Territories	8
2.3	Graph Neural Networks.....	15
2.3.1	Graph Representation Learning Through Message-Passing	15
2.3.2	Classic GNN.....	18
2.3.3	Graph Attention Network	20
2.3.4	Relational Graph Convolutional Network.....	22
2.3.5	Time Entity Aware Graph Neural Network	25
2.3.6	Recurrent Evolution Graph Neural Network.....	28
2.3.7	Time Aware Heterogeneous Graph Neural Network	34
2.4	Synthesis.....	41
2.5	Summary.....	42
3	A GNN-Based Spatiotemporal Representation Learning Approach for SETT	
	Knowledge Graphs	43
3.1	Problem Definition: Representation Learning on SETT KG	43
3.2	Input Data: SETT Knowledge Graphs and Trajectory Episodes.....	43
3.3	General Architecture of the Proposed SETT-GNN Model	45
3.3.1	Episode Encoder: Relational GCN for SETT KG episode-snapshots	45
3.3.2	Temporal and Positional Encoding.....	47
3.3.3	Transformer-Based Temporal Encoder	49
3.4	Output Representation and Downstream Tasks.....	50
3.4.1	Trajectory classification	50
3.4.2	Trajectory similarity	50
3.4.3	Clustering.....	50
3.5	Positioning to the State of the Art	51
4	Practical Implementation and Experimental Evaluation	52
4.1	Implementation Environment and Tools	52
4.2	SETT Snapshot Preprocessing.....	53
4.3	Input Representation for SETT-GNN.....	54
4.4	Model Implementation	56
4.4.1	Episode Encoder Implementation.....	57

4.4.2	Temporal and Positional Encoding Implementation	57
4.4.3	Transformer Encoder Implementation.....	59
4.4.4	End-to-End Model Integration.....	60
4.5	Evaluation Tasks and Metrics.....	60
4.5.1	Trajectory Similarity.....	60
4.5.2	Clustering.....	61
4.5.3	Trajectory Classification.....	61
4.6	Results and Ablation Studies	61
4.6.1	Classification Results	62
4.6.2	Clustering Results.....	62
4.6.3	Trajectory Similarity Results.....	63
4.6.4	Ablation Studies	65
4.7	Discussion.....	66
5	Conclusions and Future Work	68
5.1	Contributions	68
5.2	Lessons Learned	69
5.3	Limitations.....	70
5.4	Future Work.....	70
6	References	72

1 Introduction

1.1 Motivation

A *Graph Neural Network (GNN)* is a learnable (artificial) neural network that is specifically designed to operate on graph-structured data. Exploiting natural correspondence between neural network weights and the data graph, where nodes serve as artificial neurons, and edges serve as weighted connections [1], these models learn meaningful representations of the nodes and edges of the graph, based on its structure [2]. The model typically learns to map input features of the nodes to output features, so that once the GNN is built on top of the graph structure [1], the model learns to take the information attached to each node as input and produce a desired prediction or representation as output. GNNs are inspired by neighborhood aggregation ideas like convolution, but graphs are more irregular than images. Instead of using the fixed kernel size to aggregate information from surrounding pixels, GNN uses the neighbor nodes that share an edge with the target node to aggregate information. This approach has been shown to be effective in tasks such as graph classification, node classification (class prediction), and edge prediction [2]. With the independence between the number of learnable parameters and the number of nodes in the graph, GNNs can maintain good performance even when the size of the underlying graph grows.

Consequently, Graph Neural Networks have been successfully used in various domains such as medical databases and health [10], recommendation systems [11], urbanization [12], information security [13], drug discovery [2], and autonomous systems [2]. In most of these domains the datasets used are in the form of knowledge graphs, not as simple network of nodes and edges. *Knowledge graphs (KGs)*, are graphs where the connections between nodes and edges reflect a real-world knowledge.

Knowledge Graph is a concept used in literature in many fields, it has several definitions and different uses in each field. But the modern reconstruction of this notion stems from the 2012 announcement of the Google Knowledge Graph [3]. Knowledge Graphs are closely related to the Semantic Web as both aim to represent knowledge in a structured, interconnected, and machine-readable form. The Semantic Web introduced standards such as RDF, RDFS, OWL, and SPARQL [4,5], which enable data to be represented and stored as entities and relationships, queried using graph-based query languages, and reasoned over through explicit semantic definitions [1]. These standards support the creation of *Linked Open Data (LOD)* [6] which are public, cross-domain knowledge graphs encoding common knowledge, such as Dbpedia [7], Wikidata [8], or YAGO [1]. The growing interest in Semantic Web technologies and LOD has made KGs an active research topic, leading to numerous books, papers, definitions, techniques, and surveys addressing their foundations, construction, representation, and applications.

Graphs provide a concise and intuitive abstraction for a variety of domains, where edges capture the (potentially cyclical) relations between the entities inherent in social data, biological interactions, transport networks, road trajectories and so forth. While for a Knowledge Graph the definition remains contentious, varying from a specific technical proposal to more inclusive general proposals, Hogan *et al.* defines a Knowledge Graph as a *graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent relations between these entities* [1]. The term knowledge refers to a structured information from entities, their attributes, and the relationships between them, represented in a form that allow interpretation, and reasoning.

In this context, the notion of ontology is closely related to Knowledge Graphs. While a Knowledge Graph stores factual knowledge as entities and relations, an ontology provides the conceptual structure used to define and organize this knowledge. It specifies the main classes of a domain, the possible relations between them, and the constraints that give meaning to these relations. Therefore, an ontology helps ensure that the knowledge represented in the graph is consistent, semantically clear, and machine interpretable.

Different types of KG can be considered based on the domain and the type of data they represent. A fixed structure KG is known as a *Static knowledge graph (KG)*, while a KG carrying geographic or spatial information is referred to as a *Spatial knowledge graph (SKG)*, when the KG contains and represent temporal

and geographic triples it becomes a *Temporal knowledge graph (TKG)*, and the last type of KG is *spatiotemporal knowledge graph (STKG)* where the KG represent both time and space at once.

Several GNN-based methods have been proposed for static or temporal graph settings. However, most of these models are limited to deal with one type of graphs, and there is still a gap in representing spatiotemporal graphs because of its heterogeneous structure, temporal ordering, and semantic richness.

The STORIES (for Semantic Trajectories / Spatio-Temporal trajectories) team's research at LIG aims to design and exploit a digital artifact, called a **semantic trajectory**, dedicated to representing and analyzing the evolution of an object or phenomenon, or a group of objects or phenomena.

Generally speaking, a trajectory can be defined as a time-ordered sequence of observations describing the evolution of an object or entity over time. Traditionally, trajectories are often associated with moving objects, such as persons, vehicles, or organizations, and mainly describe their spatial movement through timestamps and locations. **Semantic trajectories** extend this representation by adding explicit contextual and descriptive information to each stage of the trajectory. They therefore describe not only where and when an entity changes, but also what happens, why it changes, and which characteristics are associated with this change.

In this study, the selected dataset is **Semantic Environmental Trajectories for Territories (SETT)** [14]. As its name suggests, SETT represents the trajectories of territories by describing their evolution through temporal, spatial, and semantic information, these semantic trajectories are constructed using a knowledge graph. Each territory is described through heterogeneous relations involving episodes, events, temporal intervals, spatial entities, and characteristic values.

A SETT is composed of several thematic trajectories, each covering a dimension linked to the environment such as water, snow, urbanization, vegetation, and temperature. Each of these thematic trajectories is built on a set of indicators associated with the corresponding theme and describes the thematic evolution of the observed territory. A SETT is much more complex than a traditional trajectory dataset because it does not only contain data series of measures of environmental indicators. It integrates multiple semantic layers within a knowledge graph, allowing each territory to be represented through interconnected events, episodes, places, times, and attributes.

Graph Neural Networks have recently become an active topic for researchers with its design and ability to process and learn graphs from its structure and content, as they are able to capture topological and relational dependencies through message passing. With this structure several GNN models were able to process KG in a successful manner providing surprising results and reasoning ability. However, in the field of spatiotemporal KG and specifically semantic trajectory, the complexity and volume of these spatiotemporal data make their application on GNN remains to be challenging, since the representation must preserve not only the structure, but it should consider the time interval, episode order, characteristics of each trajectory, and semantic context.

1.2 Problem statement

A trajectory of an object (person, territorial unit, company, etc..) represents the evolution of this object over time. It captures the successive states, changes, and events associated with the object, and can therefore become increasingly rich and complex as more temporal observations and semantic information are included. Spatiotemporal knowledge graphs (STKGs) provide a structured framework for this purpose by organizing data as nodes and edges, enriched with semantic meaning through an ontology. In this way, they act as a structured memory capable of representing entities, relations, and their temporal evolution in an explicit and interpretable form.

Such richness makes semantic trajectories difficult to process using standard graph learning approaches. Traditional knowledge graph embedding methods or conventional Graph Neural Network pipelines are generally not designed to directly handle this combination of heterogeneous relations, temporal intervals, events, and multi-level semantic organization. This creates the need for a dedicated representation and learning framework capable of preserving both the structural semantics of the graph and the temporal evolution of the trajectory, modeling this information in a stable vector representation, allowing us to apply downstream tasks as classification, prediction, and entity alignment.

This work is therefore guided by the assumption that Graph Neural Networks can provide a suitable foundation for learning representations from SETT Knowledge Graphs, provided that their semantic, heterogeneous, and temporal characteristics are considered.

Based on this research direction, this work addresses the following questions:

1. What are the existing Graph Neural Network approaches for Knowledge Graph representation learning?
2. Which of these approaches can be applied to or adapted for SETT Knowledge Graphs?
3. How can SETT KG be exploited within a learning pipeline for trajectory representation learning?
4. Can the learned representations capture meaningful information about territory trajectories?
5. What downstream tasks can be performed using the learned trajectory representations?

1.3 Main challenges

The problem addressed in this work is difficult because Semantic Environmental Trajectories of Territories are richer than standard graph data. A territory trajectory is not represented as a simple sequence or as a homogeneous graph. Instead, it combines semantic relations, temporal evolution, and several thematic descriptions of the same territory, such as water, vegetation, snow, temperature, and urbanization. This creates several challenges for graph representation learning.

The first challenge is the **semantic and structural heterogeneity**. SETT Knowledge Graphs contain different types of entities and relations. As a result, standard graph learning models that assume a single type of node or edge are not directly sufficient to represent this structure.

The second challenge is the **multi-thematic representation**. A territory is described through several thematic views, each representing a different aspect of its evolution. A learning model must therefore preserve these different sources of information while producing a unified representation of the territory trajectory.

The third challenge is the **temporal organization**. Territory trajectories describe changes over time through ordered temporal parts. The model must therefore capture not only the semantic structure of the graph, but also the order and duration of the different stages of the trajectory.

The fourth challenge is the **trajectory-level representation learning**. The objective is not only to represent isolated nodes or individual graph snapshots, but to learn a stable embedding of the complete territory trajectory. This requires combining local graph information with the global evolution of the territory over time.

The fifth challenge is the **evaluation of the learned representations**. Since explicit labels describing territory behaviors may be limited, the quality of the learned embeddings must be assessed through downstream tasks such as trajectory similarity, clustering, and classification when labels are available.

1.4 Contribution

The principal aim of this thesis is to study existing GNN approaches for SETT KGs, to design and evaluate a learnable and stable framework for processing spatiotemporal knowledge graphs that evolves in time through episodes and its corresponding attributes. The research advances through four stages. First, it provides survey on the state-of-the-art of GNNs, with particular attention on the possibility to apply each GNN model on SETT KGs. Second, based on this analysis of GNNs, it proposes an adapted model that is applied on SETT KGs. Third, following this proposal, it implements the proposed model with a full pipeline framework to process this type of STKGs. Finally, and after implementation, several evaluation and data ablation studies are led to assess the results and performance of the model. All these together form an end-to-end pipeline, from

theoretical aspect to implementation phase ensuring expressiveness, temporal and spatial integrity, and operational scalability while supporting downstream reasoning tasks.

This thesis contributes to the study and processing of spatiotemporal KGs through Graph Neural Networks in the context of the SETT project. Its main contributions are:

1. **A state-of-the-art study and analysis of GNN approaches for KGs and semantic trajectories**, covering vanilla GCN, moving toward spatial, temporal, and spatiotemporal GNNs. Discussing the KG representation in each model and the possibility to apply each architecture on SETT KGs or not.
2. **The proposal of an adapted GNN architecture** based on the studied approaches and designed to represent SETT spatiotemporal knowledge graphs, suggesting possible downstream tasks on these learned representations.
3. **The design of a complete end-to-end pipeline** for processing SETT STKGs, from graph representation handling time, episodes, and themes. Moving to the model implementation and training, achieving representation learning.
4. **An experimental evaluation of the proposed model**, testing the stability and performance of the model on several ablation studies and different tasks, to assess the effectiveness, and scalability of the approach.

Together, these contributions demonstrate that combining state-of-the-art neural networks, and with the correct training pipeline yields to a strong, powerful and stable spatiotemporal processing model able to learn from rich and complex semantic trajectories. The resulting framework serves as a base foundation for deploying hybrid, self-supervised spatiotemporal model applied on semantic trajectories where they are developing fast recently.

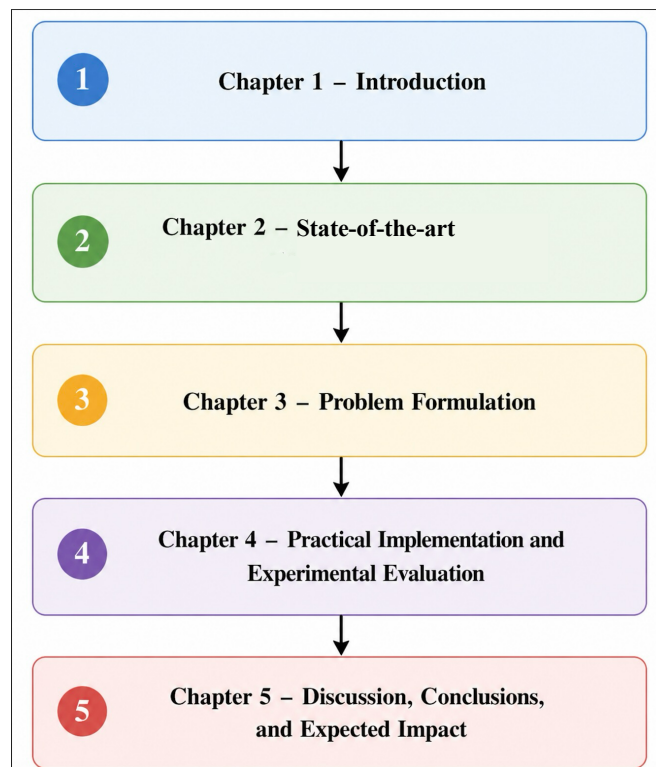


Figure 1 Overview of the thesis structure and logical progression across chapters

1.5 Report outline

This thesis is organized in a logical sequence that gradually builds from foundational theory to applied research. It starts with an overview on background concepts and related work, moving to explaining the problem and what are the possibilities to address it, progressing to the implementation framework and concludes with analysis, evaluation and future works. As illustrated in Figure 1, each chapter contributes a distinct layer of understanding such as conceptual, structural, analytical and evaluating leading to a comprehensive framework for intelligent and adaptive system.

This thesis is structured into 5 chapters, each addressing a key component of the proposed research. The organization is as follows:

1. **Chapter 1: Introduction**

This chapter establishes the motivation and significance of the research, defines the problem space and challenges facing, outlines the proposed contribution showing the need for an adapted Spatiotemporal Neural Network to process semantic trajectories.

2. **Chapter 2: State of the Art**

This chapter starts with a review on KG and SETT trajectories. It then reviews existing Graph Neural Network models applied to knowledge graphs, focusing on how each model represents the knowledge graph and whether it is suitable for SETT KGs or not.

3. **Chapter 3: Problem Formulation**

This chapter explains in detail what are the input data and how they are processed in thematic and episode snapshots for territories, discuss workflow of the pipeline and what are the architectures used for this model, and motivate how the model was trained and what downstream tasks applied.

4. **Chapter 4: Practical Implementation and Experimental Evaluation**

This chapter presents the implementation of each model focusing on their roles and libraries used, preprocessing of data before passing the data to the model, what results were obtained, and ablation studies to see model behavior on changes.

5. **Chapter 5: Conclusion and Future work**

This chapter summarizes the contribution, discusses limitations faced during the study, reflects the contributions on semantic trajectories and spatiotemporal KGs and outlines multiple directions for future research directions including semantic life trajectories.

2 State-of-the-Art

To study and understand the various models and approaches of GNNs on knowledge graphs, we need first to discover what type of knowledge graphs we have, and what are semantic trajectories specifically. After understanding this data structure, we need to understand what is the mechanism that allow neural networks and especially graph neural networks to learn from a graph. Building this knowledge allow us to initiate our study on the state-of-the art of graph neural networks applied on several types of knowledge graphs, highlighting their architecture and their ability to be applied on semantic trajectories.

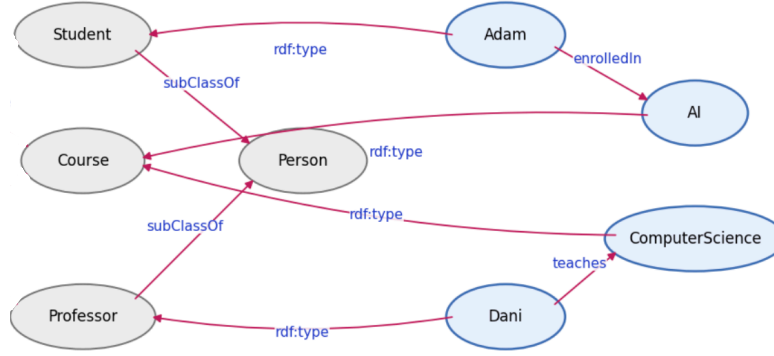


Figure 2 Example schema of a knowledge graph based on domain ontology

2.1 Knowledge Graphs

Knowledge Graphs serve as structured semantic data representations that encode real-world facts as entities and encode relationships between them in the form of triples (*subject, relation, object*), where subject and object represent entities of interest, and relation is the predicate that connects them [13]. A Knowledge Graph represents knowledge as a set of entities connected by meaningful relations. For example, the statement “Dani teaches Computer Science” can be represented by the triple (Dani, teaches, ComputerScience), while “Adam is enrolled in AI” can be represented by (Adam, enrolledIn, AI).

An ontology provides the formal conceptual structure of a domain. It defines the main classes of entities, the relationships that can connect them, and the constraints that specify how these elements should be used. In the context of Knowledge Graphs, an ontology gives semantic meaning to the graph structure and ensures that the represented knowledge is consistent, explicit, and machine interpretable. In Figure 2, the blue nodes represent individuals in the Knowledge Graph, such as Adam, Dani, AI, and ComputerScience, while grey nodes correspond to concepts/classes such as Student, Professor, Course, and Person. In addition, pink arrows such as teaches and enrolledIn correspond to properties/relations.

Ontologies define the vocabulary that describe a Knowledge Graph, as it builds the main classes of the domain, and the properties that connect them with the semantic relations between these elements. For example, in Figure 2, the ontology defines that Student and Professor are sub-classes of Person which is also a class. With this ontology representation, class hierarchy is defined, meaning in this case that students and professors are both considered persons in the domain model.

The ontology also defines properties through their domain and range. The domain indicates the class of entities from which a relationship starts, while the range indicates the class of entities to which the relationship points. For instance, the property teaches has Professor as its domain and Course as its range, meaning that this relationship connects professors to courses. In other words, if the knowledge graph contains the triple (Dani, teaches, ComputerScience), then, based on the ontology, an interpreter can infer that Dani is a Professor and that ComputerScience is a Course. Likewise, the triple (Adam, enrolledIn, AI) uses the property enrolledIn, which is defined as a relation between a student and a course.

Thus, the ontology does not replace the Knowledge Graph; rather, it provides the conceptual and semantic framework that allows the Knowledge Graph to be interpreted consistently. The Knowledge Graph contains concrete facts about individuals, while the domain ontologies define the classes, properties, and structural constraints used to represent those facts.

The heterogeneity of KGs allowed them to represent several types of data, such as, static, geo-spatial, temporal, and spatio-temporal data structures. In this sub-section we are going to introduce briefly each of these types, to form a good understanding on KGs when we talk about them later in the study on GNNs. Knowledge graphs are divided into four main categories:

- **Static knowledge graph (KG)** where the graph structure is fixed and does not change over time or space, taking as example a university knowledge graph representing students enrolled in courses and professors teaching the courses without time (dates) and/or space (location).
- **Spatial knowledge graph (SKG)** where entities and relations are associated with geographic or spatial information, for example a city infrastructure graph, indicating each building with its type and location, latitude and longitude. Spatial knowledge graphs are important when location and geographic proximity are essential for understanding how entities are related in the real world.
- **Temporal knowledge graph (TKG)** where entities and relations evolve over time, as for a company history KG recording its CEOs with their periods and temporal events (companyA - acquired - CompanyB in 2022). Temporal KGs are useful when the validity of facts depends on time and when the evolution of entities and relations must be explicitly modeled.
- **Spatiotemporal knowledge graph (STKG)** is a graph representing entities relations, spatial information, and temporal evolution all together. For example, a territory trajectory graph, monitoring each territory evolution over time but at the same time updating its spatial information and semantic relations with surrounding territories/entities.

Based on this introduction of KG types, we can move to the next part of this section that introduces semantic trajectories of territories which are represented through KGs and specifically STKG.

2.2 Semantic Environmental Trajectories of Territories

Semantic Environmental Trajectories of Territories-SETT is a multi-layered ontology designed to represent the environmental evolution of Territorial Units over time within the LOD cloud [14]. A Territorial Unit corresponds to a defined portion of the Earth's surface considered in a territorial study, such as municipalities in France, or canton in Switzerland. The main principle behind SETT is to describe the development of a region through environmental factors such as vegetation, water, snow, temperature, and urbanization. These environmental factors are described by spectral indicators of multiple environmental themes, such as Normalized Difference Vegetation Index (NDVI) for vegetation, Normalized Difference Built-up Index (NDBI) for urbanization, Normalized Difference Snow Index (NDSI) for snow cover, Normalized Difference Water Index (NDWI), and Temperature Index.

SETTs were introduced in Daniela Fernanda Milon Flores PhD thesis [14] as an ontology-based model for representing environmental trajectories of territories as a part of the TRACES project[traces-anr-fns.imag.fr/]. TRACES project is a French Swiss international collaborative research program focused on constructing a knowledge graph that integrates diverse data sources to provide insights into the environmental evolution of municipalities. Where territorial units in TRACES datasets are representing municipalities.

The *Structured trajectory* is the second layer of the SETT ontology, which describes the segments and transition points identified within the raw trajectories, corresponding to the blue nodes in Figure 3. This layer provides a vocabulary for describing segments such as decrease or increase, depending on the observed trend or slope direction of each segment. In this way, raw observations are transformed into a more structured representation that captures changes and trends in the evolution of territorial indicators. The identification of segments and inflection points as denoted in [15] can be carried out using several segmentation or change-point detection methods. The goal using these methods was mainly to detect segments from satellite indices affected by strong seasonal variations, and for this they used BEAST (Bayesian Estimator of Abrupt Change, Seasonality, and Trend) [16] and CCDC (Continuous Change Detection and Classification) [17]. Moreover, to detect transition points in these segments they used libraries, such as Ruptures [18], Claspy [19], or Facebook Prophet (<https://facebook.github.io/prophet/>). Therefore, the structured trajectory provides a description of the raw observations in terms of segments, transition points, and slope direction, enriching the semantic representation for each territory trajectory.

The *Thematic trajectory* is the last layer in the SETT ontology. This layer provides a more detailed semantic interpretation of the segments and transition points in the structural level by defining themes (denoted by pink nodes in Figure 3) [15]. At this layer, each segment and transition point are described with a specialized vocabulary according to the variation in each indicator. Following this enrichment, segments and transition points at this level are referred to as episodes and events belonging to a specific theme depending on the indicator. For example, an increasing segment belonging to temperature index will become “warming”, or “greening” in the case of vegetation index. This representation of episodes and events defined in this thematic level allow other data in the LOD cloud to be connect to them, enabling enrichment, explanation, or contextualization of the observed trajectory [15].

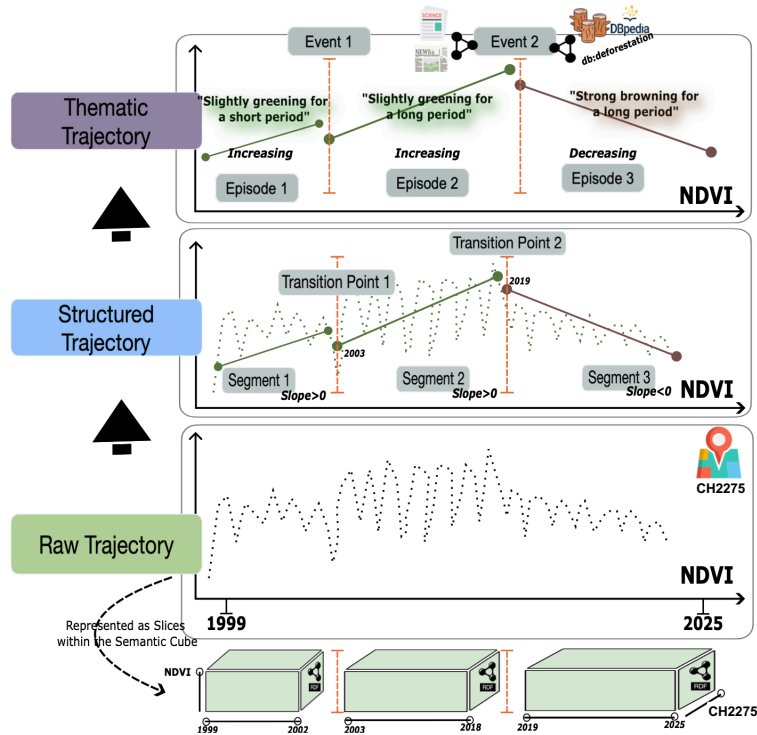


Figure 4 Example of a Vegetation time series (NDVI) structured into the three layers of SETT [14]

These three layers together with their hierarchy forms the full structure for SETT. The raw layer provides the initial spatiotemporal observations, the structured layer identifies segments and transition points, and the thematic layer interprets these elements as meaningful episodes and events. Understanding the special representation of these trajectories in terms of episodes and events form the basic and important step to start our study of GNNs.

Figure 4 shows an example of how the raw, structured, and thematic vegetation trajectories are constructed from the NDVI time series. The raw trajectory stores the original observations as linked data, using a slice of the semantic data cube in LOD cloud. Then, based on using algorithms to structure the raw observations into a structured trajectory, two transition points were detected forming 3 segments, each representing a portion of the original data [14]. In addition, based on the structural features such as segments slopes and transition point dates, each segment is enriched. Finally, the thematic trajectory interprets each segment and transition point adding theme-specific annotation, such that we obtain two Events corresponding to the two transition points, and three Episodes corresponding to the three segments. Each Episode is associated to a semantic description that summarizes the vegetation behavior during a given period. For instance, the first episode may indicate a slight greening phase over a short duration, while the third episode may describe a strong browning phase over a longer period. Similarly, one of the detected Events, occurring in 2019, is considered significant because it marks an important change in the vegetation trajectory. This event can also be enriched with external contextual information. For example, the observed decrease in vegetation in the last episode may be explained by a deforestation event that affected the territory in 2019. This event can also be connected to open knowledge graph sources as DBpedia or Wikidata for an extra environmental knowledge they offer through their wide range of domains.

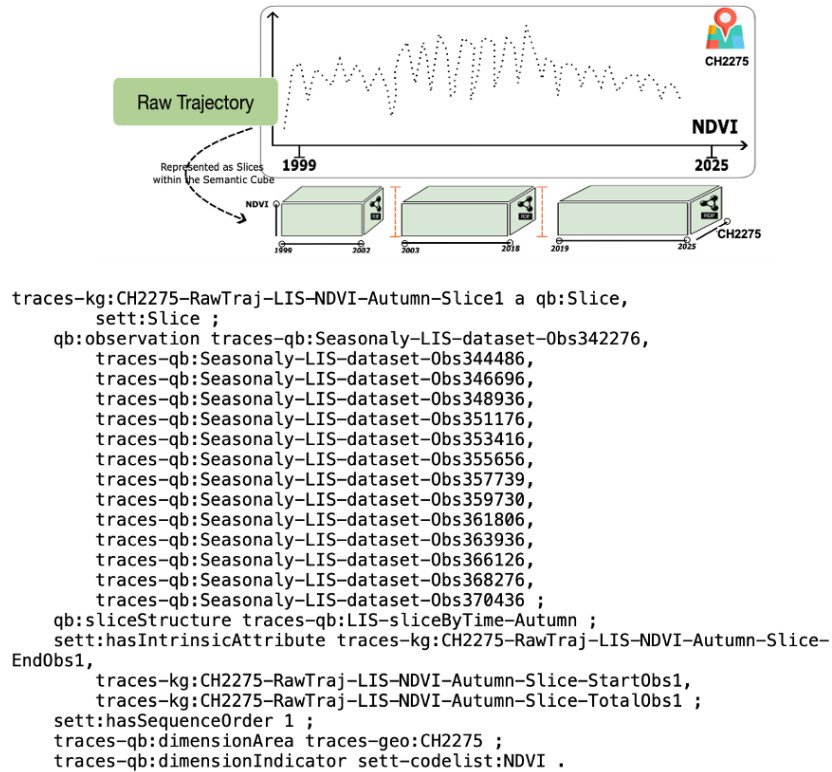


Figure 5 NDVI Raw Trajectory for the municipality of Murten

These three layers all together are represented in TRACES project by a spatiotemporal KG, with RDF turtle style [https://www.w3.org/TR/turtle/]. Figure 5 shows how the raw trajectory of the NDVI indicator in the autumn dataset is represented in RDF Turtle form for the municipality of Murten in Fribourg dataset, identified by the code CH2275. As shown in the figure, the raw NDVI trajectory is represented through slices. The RDF Turtle excerpt presented here corresponds to the first slice, denoted as `traces-kg:CH2275-RawTraj-LIS-NDVI-Autumn-Slice1`, which is declared as an instance of `qb:Slice` and `set:Slice`.

This slice is composed of several observations through the relation `qb:observation`. These observation nodes are represented by resources such as `traces-qb:Seasonally-LIS-dataset-Obs342276`, `traces-qb:Seasonally-LIS-dataset-Obs344486`, and the other observations listed in the RDF representation. The slice also contains intrinsic attributes, such as `Slice-StartObs1`, `Slice-EndObs1`,

and `Slice-TotalObs1`, which describe the beginning, the end, and the total number of observations contained in the slice.

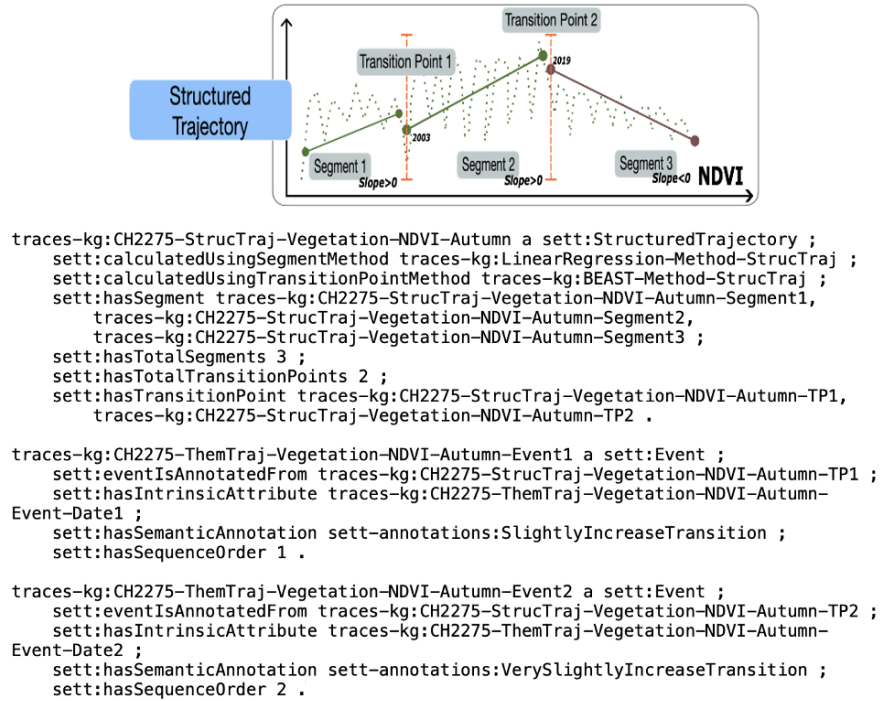


Figure 6 NDVI structured trajectory for the municipality of Murten

Finally, the slice is linked to the territorial unit through `traces-qb:dimensionArea`, which points to `traces-geo:CH2275`, and to the environmental indicator through `traces-qb:dimensionIndicator`, which specifies that this raw trajectory concerns the NDVI indicator. It is important to note that the RDF Turtle excerpt shown in the figure represents only the first raw trajectory slice. As illustrated on the left side of the figure, the complete raw trajectory is composed of additional slices, including two other slices that cover later periods of the trajectory.

Figure 6 shows how the structured trajectory of the NDVI indicator in the autumn dataset is represented in RDF Turtle form for the municipality of Murten. As shown in the figure, the structured trajectory is represented by the resource `traces-kg:CH2275-StrucTraj-Vegetation-NDVI-Autumn`, which is declared as an instance of `sett:StructuredTrajectory`.

This structured trajectory is linked to the methods used to construct it. The property `sett:calculatedUsingSegmentMethod` indicates the method used for segment calculation, here `traces-kg:LinearRegression-Method-StrucTraj`, while `sett:calculatedUsingTransitionPointMethod` indicates the method used for transition point detection, here `traces-kg:BEAST-Method-StrucTraj`.

The RDF representation shows that this structured trajectory is composed of three segments, linked through the property `sett:hasSegment`, and two transition points, linked through the property `sett:hasTransitionPoint`. This is also indicated explicitly by `sett:hasTotalSegments 3` and `sett:hasTotalTransitionPoints 2`.

The figure also shows how transition points are used to construct thematic events. For example, `traces-kg:CH2275-ThemTraj-Vegetation-NDVI-Autumn-Event1` is declared as an instance of `sett:Event` and is annotated from the first transition point through `sett:eventIsAnnotatedFrom`. It is also associated with an intrinsic attribute representing its date and with the semantic annotation `sett-annotations:SlightlyIncreaseTransition`. Similarly, `Event2` is annotated from the second

transition point and receives the semantic annotation `settlements:VerySlightlyIncreaseTransition`.

This RDF representation shows the link between the structured and thematic layers of SETT. The structured trajectory provides the detected segments and transition points, while the thematic trajectory interprets these transition points as meaningful events with semantic annotations and sequence orders.

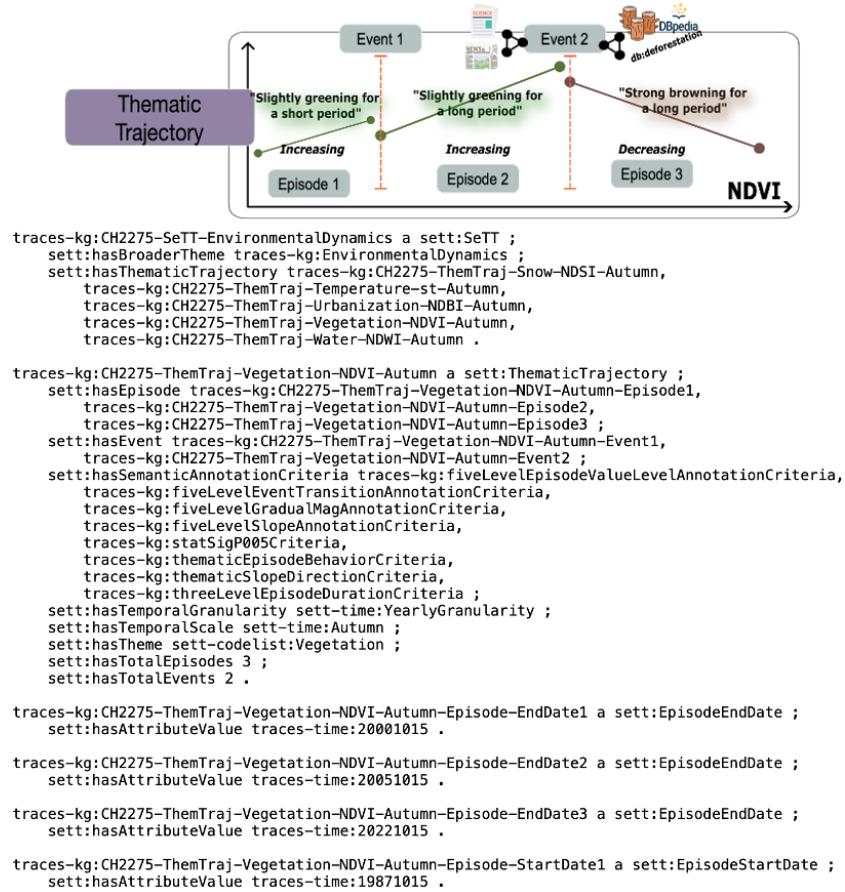


Figure 7 NDVI thematic trajectory for the municipality of Murten

Figure 7 shows how the thematic trajectory of the NDVI indicator in the autumn dataset is represented in RDF Turtle form for the municipality of Murten. As shown in the figure, the thematic trajectory is first connected to the global SETT trajectory of the territory, represented by `traces-kg:CH2275-SetT-EnvironmentalDynamics`, which is declared as an instance of `sett:SetT`.

This SETT trajectory is linked to several thematic trajectories through the property `sett:hasBroaderTheme`, such as urbanization, snow, temperature, water, and vegetation. The NDVI vegetation thematic trajectory is represented by the resource `traces-kg:CH2275-ThemTraj-Vegetation-NDVI-Autumn`, which is declared as an instance of `sett:ThematicTrajectory`.

The RDF representation shows that this thematic trajectory is composed of three episodes and two events. The episodes are linked through the property `sett:hasEpisode`, while the events are linked through the property `sett:hasEvent`. The total number of episodes and events is also explicitly indicated through `sett:hasTotalEpisodes 3` and `sett:hasTotalEvents 2`.

The thematic trajectory also contains semantic information that supports the interpretation of the observed evolution. For example, it is associated with annotation criteria such as `sett:hasSemanticAnnotationCriteria`, `sett:hasTemporalGranularity`, and

`sett:hasSeasonalGranularity`. These properties indicate how the thematic trajectory is semantically described and at which temporal scale it is interpreted.

Finally, the RDF excerpt also shows how the temporal boundaries of the episodes are represented. Each episode is associated with intrinsic attributes such as `EpisodeStartDate` and `EpisodeEndDate`, which indicate the beginning and end of the corresponding period. For example, the first episode starts in and ends in 20001015, while the following episodes cover later periods. This shows how the thematic layer transforms the structured segments and transition points into meaningful episodes and events describing the vegetation evolution of the territory.

In the next subsection, we introduce the main mechanism from which all graph neural networks stand on to learn graph representation. This mechanism is very important to understand how graph neural network works, and it provides the foundation for discussing the state-of-the-art architectures considered in this study.

2.3 Graph Neural Networks

This section provides an overview of the main families of Graph Neural Networks, starting from the most traditional ones to GNNs that represent space and time. This overview provides the basis understand the basic principles of GNNs and specific features of *temporal-GNNs*, *spatial-GNNs*, and *spatiotemporal-GNNs*. This overview also prepares the analysis of which models can be applied to or adapted for Semantic Trajectories of Territories represented in STKGs.

In this overview and for a better clarity, each family of GNNs is presented in the same structure. First, the family for which the model belongs. Second, the way the KG is represented and processed. Third, the architecture of the model. Fourth, the main tasks applied by the model. Finally, the suitability and limitation of the model with respect to SETT KGs.

Before beginning this overview, it is appropriate to introduce the principle of message passing from which Graph Neural Networks learn graph representation.

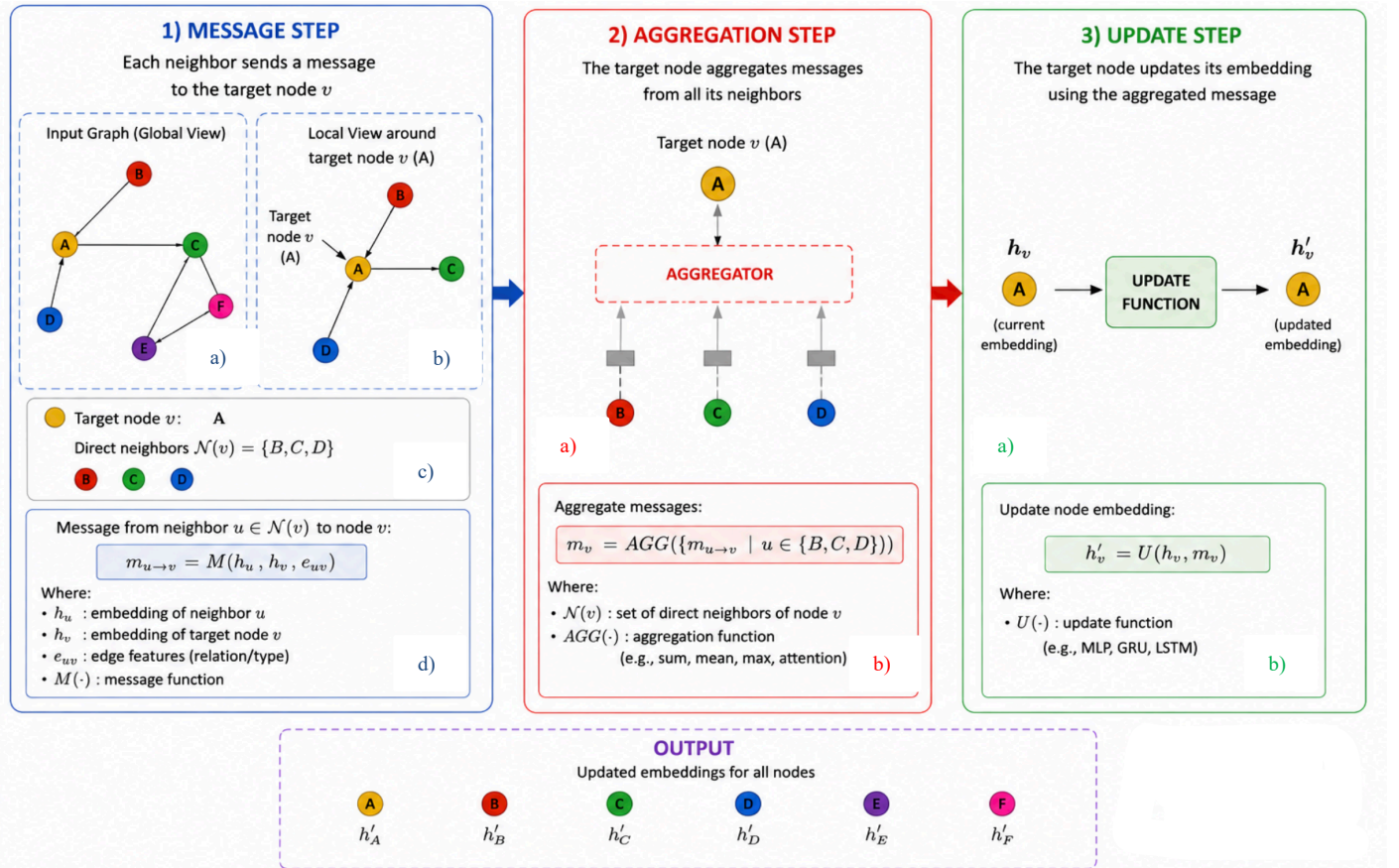


Figure 8 Illustration schema for message-passing GNN algorithm

2.3.1 Graph Representation Learning Through Message-Passing

The principle of graph representation learning is based on learning a continuous vector -of real space $\mathbb{R}$ with dimension d - for graph elements, such as nodes, edges, or entire graphs. This representation vector is called *embedding*. A node embedding is then, a d -dimensional representation vector that encodes information about the node itself, its attributes, and its structural position within the graph.

Let us consider a graph composed of nodes representing entities, and edges representing the relation between these entities. Each node v is associated with an initial feature vector of dimension d called feature entity vector, denoted by h_v . When explicit node attributes are available, such as in a student-course KG, for a student node, the age, study level, major, number of completed courses, or GPA, these attributes can be encoded and

used as the initial feature vector of the corresponding node. Otherwise, we can use one-hot encoding or random initialization for creating the feature vectors. These vectors will represent the initial structure of each entity. The edges indicate the connectivity of the graph; this information is stored in an edge index matrix. Edge attributes or relation types may also be represented when they are available in a graph, such that for an edge e we have a feature edge vector h_e .

In the field of Graph Neural Networks these embeddings are learned through a *message-passing layer*. In this mechanism, each node updates its representation based on information received from its direct neighboring nodes. For a target node v , its neighborhood is denoted by $N(v)$, it corresponds to the set of nodes which have a direct edge with node v . As illustrated in Figure 8 - step 1-a and 1-b, node A is considered a target node, and its directly connected with B, C, and D. This will imply, during message step that only these three nodes will send messages to node A.

The first step in message-passing is identifying the neighbors of each target node (see Figure 8 step1-c), such that each neighbor node $u \in N(v)$, sends a message to the target node. As indicated in Figure 8 step1-d, this message $m_{u \rightarrow v} = \mathbf{M}(h_u, h_v, e_{uv})$ is represented by the current embedding of the neighbor node h_u , the current embedding of the target node h_v , and the possible edge features or relation types if available e_{uv} . Where $\mathbf{M}(\cdot)$ is a message function that decides in which form the neighboring node embeddings will be transferred, because in some GNN models the messages are passed with weights or specific constraints. For simplicity, we assume at this stage that the embeddings will be passed to the next step in their original representation.

After the target node A receives messages (which are the node embeddings) from several neighbors, as indicated in Figure 8 step 2-a, node A will receive messages from nodes B, C, and D. It will now have more than one embedding from each neighbor u_i . An *aggregation* technique is needed at this step to combine the neighbor representations in a single representation. Usually, aggregation techniques are simple like sum, mean, max, or average, as shown in Figure 8 step 2-b. After the aggregation step, neighboring embeddings will be collected in one aggregated embeddings message represented as m_v :

$$m_v = AGG(\{m_{u \rightarrow v} \mid u \in N(v)\})$$

The aggregation function should be **permutation-invariant**, meaning that the order in which neighboring nodes are aggregated does not affect the final aggregated embedding.

Finally, the representation embedding for node A at this stage is still h_v . To integrate the neighboring structure into node A, we need to update the embedding of A based on the aggregated embeddings message m_v :

$$h'_v = \mathbf{U}(h_v, m_v)$$

The update function $\mathbf{U}(\cdot)$ combine the target embedding with the aggregated embeddings message, and follows it with a non-linear activation function, or a more complex mechanism such as a Multi-Layer Perceptron (MLP), or attention-based updates, depending on the GNN architecture. The resulting embedding h'_v is the updated embedding of the target node v , applying this process on all nodes allow GNN to produce updated

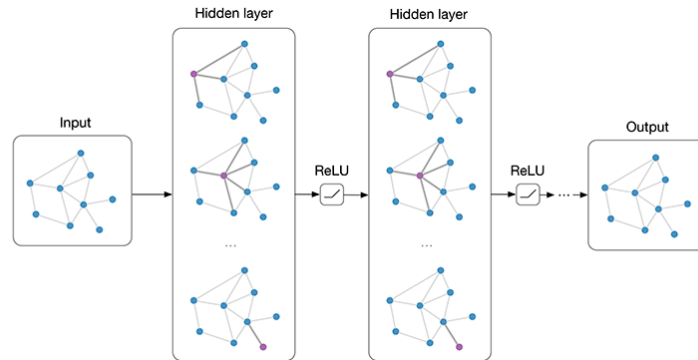


Figure 9 Multi-layer Graph Convolutional Network (GCN) [31]

embeddings for the entire graph, as shown in Figure 8 step 3-b.

A graph Neural Network could be constructed with one or more message-passing layers, where each layer applies the message-passing principle on the entire graph. Figure 9 describes the architecture of having several layers in a GNN, where each layer applies the message-passing mechanism on each node from the input graph.

Through this principle, a GNN can learn node embeddings from the graph structure and perform downstream tasks. The downstream tasks we focus on in this study are node class prediction, Entity similarity, and clustering. As shown in Figure 10-a, node class prediction, or node classification, is a task where we use the learned node embeddings from the message-passing layer. According to the embedding of the unknown node (C), a classifier compares it to the embeddings of other nodes, for example nodes (A), (B), (F), (D), and (E). Based on this comparison, (C) is assigned to the class whose nodes are the most similar to it. In this example, just given for illustration, let us assume it is assigned to class 1, shown in green.

Another task, shown in Figure 10-b, is clustering. This task is based completely on the learned node embeddings. After learning the embeddings, we can use clustering algorithms, such as k-means, to group nodes with similar representations. To visualize the result, dimensionality reduction methods, such as PCA or t-SNE, can be used to reduce the node embedding vectors to two dimensions. PCA refers to Principal Component Analysis, while t-SNE refers to t-distributed Stochastic Neighbor Embedding.



Figure 10 Examples of downstream tasks using learned graph embeddings

Another task we are interested in in this study is entity similarity. In this task, we use the learned embeddings of nodes to calculate a distance or similarity score between two entities. For example, if two entity embeddings have a small distance, such as a small L1/ Manhattan distance or L2 distance [22], this means that the entities are close in the embedding space and may be considered similar. On the other hand, if the distance is large, the entities are considered less similar. An illustration of this task is shown in Figure 10-c.

In recent years, Graph Neural Networks (GNNs) have gained significant interest as one of the most influential advances in modern machine learning. The main idea behind GNN, as we discussed previously, is to map nodes in a graph into a d-dimensional embedding such that similar nodes have embeddings close to each other's, and through message passing GNN can learn this mapping given a graph. With the increasing interest in KGs, several models of GNNs were developed and applied based on the task and type of KG (static, temporal, spatial, spatiotemporal). GNNs have demonstrated great potential in a wide range of tasks applied on various graph types, the next sections make an overview about each model discussing its model family, KG representation, message passing mechanism, main task, strengths, and limitations.

2.3.2 Classic GNN

2.3.2.1 Model family

The **Graph Convolutional Network (GCN)** is one of the foundations of Graph Neural Networks and plays a central role in graph-based deep learning development. Graph Convolutional Networks extend the principle of convolutional operations from Euclidean domain such as images, to graph-structured data. In this view, GCN can be viewed as a spatial graph neural network with spectral inspiration, where each node updates their representation based on information aggregated from their neighbors through message passing [32].

2.3.2.2 Knowledge Graph Representation

Graph Convolutional Networks are not fully adapted to work on KGs, since Knowledge Graphs are considered multi-relational graphs, with semantics associated with each edge type. A GCN treats all edges in a graph with equal weights without considering modelling edge types in the message propagation. For this reason, although GCN can be applied to simple or homogeneous graphs, it is less suitable for highly relational KG in their original form. However, they stand as the base models from which other models were developed, and for this we study its architecture.

2.3.2.3 GCN Layer

Taking into consideration a single Graph Convolutional layer l , let $h_v^{(l-1)}$ denote the representation of node v at layer $l-1$. Then the GCN layer can be written as follows:

- **Message Computation**

Each node u computes a message $m_u^{(l)}$ using a linear transformation layer, multiplying the linear matrix with each node representation vector of the previous layer [32], such that the message carries the latest information of the node from the previous layer:

$$m_u^{(l)} = W^{(l)} h_u^{(l-1)}$$

where $W^{(l)}$ is a learnable weight matrix.

- **Aggregation**

After calculating the messages from the neighbors, the target node needs to combine this information into a single representation such that it updates the previous state of the target node based on these messages. The aggregation function must be **permutation invariant** [32], common choices include Sum, Mean, or Max.

Each node v aggregates the message such that:

$$a_v^{(l)} = \text{AGG}^{(l)} \left(m_u^{(l)}, u \in N(v) \right)$$

- **Update**

The aggregated messages carry the structure and information from all neighboring nodes; the update step is to integrate this information with the current representation of the target node. For this, the aggregated message is transformed through an update function, followed by a non-linear activation [32]. In GCN the update equation is reduced to:

$$h_v^{(l)} = \sigma \left(a_v^{(l)} \right)$$

Thus, entire layer can be written compactly in matrix form:

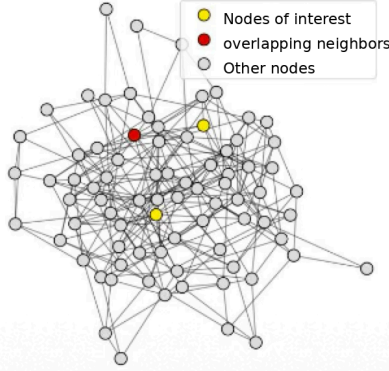
$$H^{(l)} = \sigma \left(A H^{(l-1)} W^{(l)} \right)$$

Where:

- $H^{(l-1)}$ is the matrix of node embeddings at layer l-1,
- A is the normalized adjacency matrix,
- $W^{(l)}$ is the learnable weight matrix,
- $\sigma(\cdot)$ is a non-linear activation function

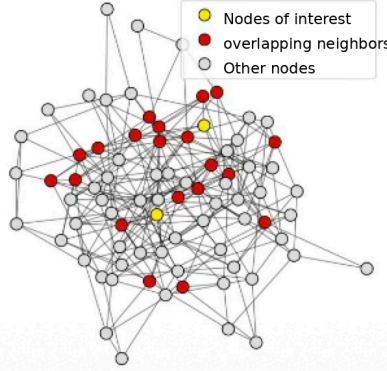
1 – GNN layer

Only 1 overlapping node



2 – GNN layers

About 20 overlapping nodes



3 – GNN layers

Almost all nodes overlapping

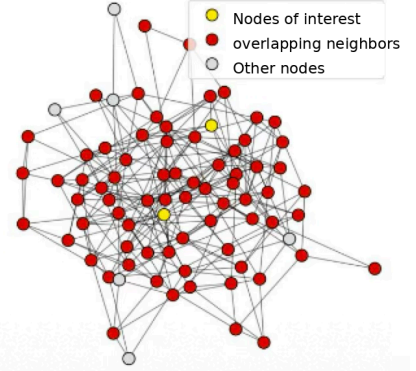


Figure 11 neighboring node overlap with the increase of message-passing layers

2.3.2.4 Main tasks

The main task of GCN, is node classification, which could be also seen as node class prediction [32]. Its core idea is simple: after applying several message-passing layers on each node, GCN captures local structural information while simultaneously exploiting node attributes. As a result, adding a classifier at the head of the network allow to predict the class of each node based on its embedding representation. In this context, the class of a node refers to the predefined label assigned to that node. For example, in a geographic knowledge graph, a node such as Grenoble may have the class City, while a node such as France may have the class Country.

2.3.2.5 Strengths and limitations

One of the main strengths of GCNs lays in its simplicity and the ease of implementation, providing an effective and efficient baseline for graph learning especially when node features are informative, and the graph is homogeneous. Its simple formulation formed the bases for many future models allowing to represent much more complex graphs and networks. On the other hand, GCNs have several important limitations. Since GCNs cannot distinguish between edge types, they not well suited to multi-relational heterogeneous graphs and so not well suited to KGs. In addition, stacking multiple GCN layers above each other sometimes causes smoothing for nodes since the group of neighbors for each target node will be similar to each other with the increase in range of neighbor-hops. As shown in Figure 11, if we take 3-hop neighbors which are 3 layers of GCN, each target node will receive messages from almost the whole graph nodes, as the figure how the overlapping nodes (denoted in red) are almost the whole graph in this case, this will result in having similar representations for all nodes as GCN do not take into account edge attributes. Furthermore, the model is mainly designed for static graphs and so it cannot model temporal or spatiotemporal knowledge graphs.

In relation to our work, GCN is important as a conceptual starting point for understanding graph-based message passing. It illustrates the basic principle of neighborhood aggregation and representation update on graphs. However, it is not sufficient on its own for modeling semantic trajectories represented as spatiotemporal knowledge graphs, since our setting involves multiple relation types, temporal evolution, and richer contextual structure. For these reasons, more specialized models such as R-GCN and temporal graph neural networks are more appropriate for our problem.

2.3.3 Graph Attention Network

2.3.3.1 Model family

Attention mechanisms have become the state-of-the-art in many sequence-based tasks such as machine translation and learning sentence representations. One of the major benefits of attention-based mechanisms is their ability to focus on the most relevant parts of the input to make decisions [20]. **Graph Attention Network (GAT)** is an attention-based graph neural network built on learning weights for each node such that it controls the contribution on other nodes. It was considered a novel foundation at its time as it combined the attention principle from the transformers with graph neural networks. GAT is also viewed as a *spatial graph neural network* as it extends the principle of GCN to learn from the graph structure both node representations, and importance of each node on its neighbors.

2.3.3.2 Knowledge Graph representation

Same as GCN, Graph Attention Network is intended for *graph-based data* structure and not specifically on KG or spatiotemporal knowledge graphs. It operates on nodes and its neighbors, but it does not model relation types, it does not model relation types, edge semantics, or time in a graph. So, GAT operates on *static graphs* and does not rely on an explicit triple-based scoring function, as it learns node embeddings through message passing.

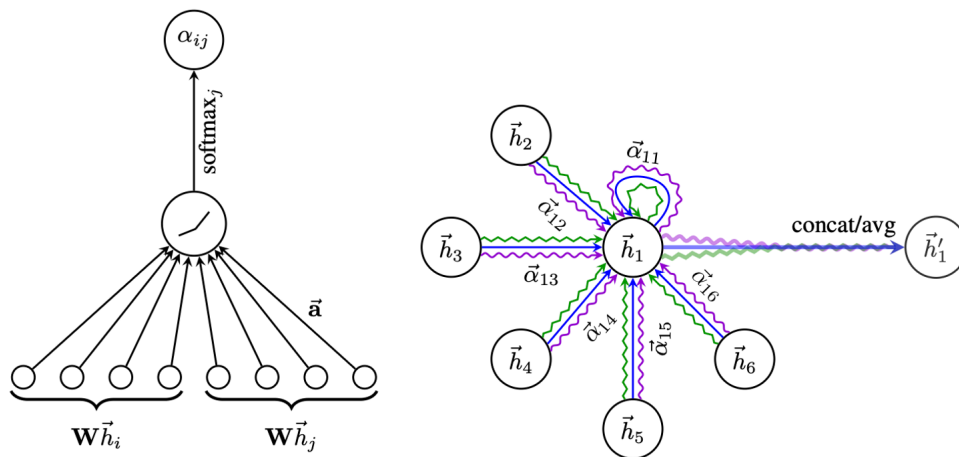


Figure 12 Schema illustrating graph attention weights showing the neighboring node weights contributing on each target node [20]

2.3.3.3 GAT Layer

The building block of Graph Attention Network is the graph attention layer, which is a variant of the aggregation function. Considering the next equations, let N be the number of nodes and F be the dimension of the feature vector for each node. The input to each graph attentional layer is a set of node features: $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, \dots, \vec{h}_N\}$: $\vec{h}_i \in \mathbb{R}^F$. The output of each graph attentional layer is a new set of node features, which may have a new dimension F' : $\mathbf{h}' = \{\vec{h}'_1, \vec{h}'_2, \dots, \vec{h}'_N\}$, $\vec{h}'_i \in \mathbb{R}^{F'}$. $\mathbf{W} \in \mathbb{R}^{F' \times F}$ is the linear transformation matrix.

Following the main blocks for a GNN layer, the graph attention layer is consisted of:

- Attention weight computation

The message passing layer for GAT is based on computing a learnable attention weight for each edge between a target node h_i and its corresponding neighboring nodes h_j , such that this weight controls the contribution of each neighboring message on its target. As shown in Figure 12, For the target node with representation vector h_1 , attention coefficients are computed for each of its neighbors performing self-attention on the node, using

a shared attention function $a: R^{F'} \times R^{F'} \rightarrow R$, that computes the attention coefficients capturing the importance of node j 's features to node i :

$$e_{ij} = a(W_l h_i, W_r h_j)$$

The most general formulation of self-attention allows every node to aggregate structural information from all other nodes. However, to account for graph structure in the attention mechanisms, they use a **masked attention**. In masked attention, a target node i computes attention coefficients $e_{\{ij\}}$ only for nodes $j \in N_i$ where N_i is some neighborhood of node i in the graph. To easily compare coefficients across different nodes, normalization across j is achieved, this is shown in the left part of Figure 12, such that the attention weight is:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in N_i} \exp(e_{ik})}$$

- **Aggregation And Update**

In each layer of GAT, after the attention coefficients are computed for that layer, the aggregation function is computed by a weighted sum of neighborhood messages. As shown in the right part of Figure 12, after calculating the attention weights α_{ij} , each neighboring node sends the message to the target node holding its weight to control the contribution on target node, α_{11} refers to self-loop case, where the target node assigns an attention weight to its own previous representation, this allows the updated representation of the node to preserve information from itself in addition to information from its neighbors. These aggregated features will serve as the final output features for every node:

$$h'_i = \sum_{j \in N_i} \alpha_{ij} W_r \vec{h}_j$$

- **Multi-Head Attention**

The advantage of using attention mechanism is the ability to have multi-head layers each is responsible for a specific task or focuses the attention on specific characteristics for nodes. To do this [20] use K independent attention mechanisms, called “heads”, which computes output features as in the above equations. Then, concatenate these output feature representations yields:

$$\vec{h}'_i = ||_{k=1}^K \left(\sum_{j \in N_i} \alpha_{ij}^{(k)} W_r^{(k)} \vec{h}_j^{(k)} \right)$$

Where $||$ is concatenation, $\alpha_{ij}^{(k)}$ are the normalized attention coefficients computed by the k -th attention mechanism ($\alpha^{(k)}$), and $W^{(k)}$ is the corresponding input linear transformation's weight matrix.

2.3.3.4 Main tasks

The main task GAT used for is node class prediction (node classification). After the GAT layers compute the embedding of each node by assigning different attention weights to its neighbors, the final output layer transforms each node embedding into a prediction. For standard single-label classification, this prediction is usually produced using a **softmax** function, which gives a probability distribution over the possible classes. For multi-label classification, where a node may belong to more than one class, a **sigmoid** function can be used instead. In the original GAT paper [20], the model was evaluated on both transductive and inductive node classification tasks, showing that attention-based message passing can be effective for predicting node labels in graph-structured data.

2.3.3.5 Strengths and limitations

The ability to learn different weights to different neighbors marked a main strength for GAT and provided it with a greater flexibility and importance compared to vanilla GCN. The model is reported as efficient, scalable

and parallelizable. It has been directly applied on inductive problems, including cases where test graph is unseen during training [20]. The original paper reports strong results on standard benchmarks such as Cora, Citeseer, Pubmed, and PPI, where GAT achieved or matched state-of-the-art performance. In particular, it outperformed GCN on Cora and Citeseer, highlighting the practical value of attention-based aggregation [20].

However, GAT has several limitations in the context of our work. First, it is not designed for multi-relational knowledge graphs, since all neighbors are treated through the same attention mechanism without explicit relation-aware transformations. Second, it does not model temporal evolution, making it unsuitable on its own for spatiotemporal knowledge graphs. Finally, because edge semantics are not explicitly incorporated in the original model, its direct application to semantic trajectories represented as rich spatiotemporal KGs remains limited. Consequently, although GAT is an important state-of-the-art GNN model and a useful reference for attention-based message passing, more specialized architectures are required for settings that involve multiple relation types, temporal dependencies, and richer semantic structure.

2.3.4 Relational Graph Convolutional Network

2.3.4.1 Model family

Relational Graph Convolutional Network (R-GCN) was introduced as an extension for GCN principle where it treats all edges with the same weights, to a new place where each edge has its own weight matrix, such that the model now can distinguish between different edges inside a multi-relational KG through relation-specific transformations [21]. R-GCN belongs to the family of **spatial graph neural networks**, where it generalizes the standard graph convolution mechanism from homogeneous graphs to relational knowledge graphs.

2.3.4.2 Knowledge graph representation

The structure of R-GCN allowed it to be applied directly on knowledge graphs, as multi-relational directed graphs, where: nodes represent entities, edges represent relations, and multiple relation types can connect the same entities. This makes R-GCN particularly appropriate for knowledge bases expressed as triples of the form (*subject, relation, object*).

Let the relational graph G be denoted by:

$$G = (V, E, R)$$

where:

- V is the set of nodes,
- E is the set of labeled edges,
- R is the set of relation types.

A labeled edge is written as:

$$(v_i, r, v_j) \in E$$

Where $r \in R$ is the relation type.

At layer l , each node v_l has a hidden representation:

$$h_i^{(l)} \in R^{d^{(l)}}$$

Where $d^{(l)}$ is the dimension of the feature vector at layer l [21].

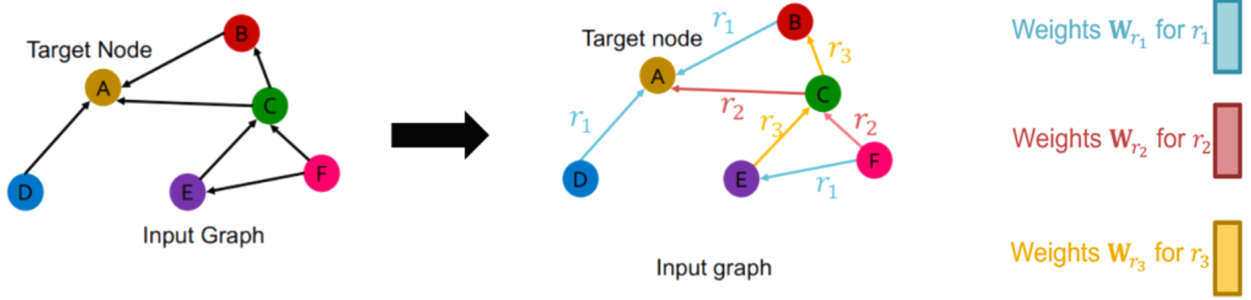


Figure 13 schema illustrating how Relational-GCN can distinguish between edge types in a graph [32]

2.3.4.3 R-GCN layer

- **Message computation**

In R-GCN, the message sent from each neighboring node j to target node i under relation r is obtained by a relation-specific linear transformation to the neighbor representation:

$$m_{ijr}^{(l)} = W_r^{(l)} h_j^{(l)}$$

This means that the model does not treat all edges in the same way. As shown in Figure 13, each relation type r_i has its own transformation matrix W_i , allowing the model to encode the semantics of different relations explicitly. This is one of the main differences between R-GCN and standard GCN [21].

- **Aggregation**

After computing the incoming messages, R-GCN aggregates them by summing over all neighbors and all relation types. The aggregation step can be written as:

$$\text{Agg}_i^{(l)} = \sum_{r \in R} \sum_{j \in \mathcal{N}_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)}$$

The normalization constant $c_{i,r}$ is used to control the scale of the sum. A common choice is:

$$c_{i,r} = \text{norm}(\mathcal{N}_i^r)$$

which corresponds to the number of neighbors of node i under relation r . This prevents nodes with many neighbors from dominating the representation [21].

- **Update**

The final update combines the aggregated relational messages with a self-loop contribution, so that the node's own previous representation is also preserved:

$$h_i^{(l+1)} = \sigma \left(\text{Agg}_i^{(l)} + W_0^{(l)} h_i^{(l)} \right)$$

Equivalently, by substituting the aggregation term:

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

where:

- σ is sigmoid
- $W_0^{(l)}$ is self linear transformation matrix
- $\mathcal{N}_i^r$ are neighbors of node i under relation r

The self-loop term $W_0^{(l)} h_i^{(l)}$ is important because it ensures that a node's next representation depends not only on its neighbors, but also on its own previous state.

2.3.4.4 Main tasks

Two main tasks are considered for evaluating the proposed model:

First task is **entity classification**, adding a softmax classifier on the head of the final layer, and the model is trained using cross entropy loss over labeled nodes defined as:

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

where:

- $\mathcal{Y}$ is the set of labeled nodes,
- K is the number of classes,
- t_{ik} is the ground-truth label indicator,
- $h_{ik}^{(L)}$ is the predicted class probability for node i at the last layer;

Second task, deals with link prediction, using R-GCN as an **encoder** to produce node embeddings, and a **decoder** is used then as a scoring function to give score among possible triples. In the paper [21], the decoder is **DistMult**, with scoring function:

$$f(s, r, o) = e_s^T R_r e_o$$

where:

- e_s is the embedding of the subject (e_s^T is the transpose of the vector embedding),
- e_o is the embedding of the object,
- R_r is the relation-specific diagonal matrix.

Thus, in link prediction, R-GCN is not used as a decoder or scoring function, it was used for relational entity representation to be used in a decoder.

2.3.4.5 Strength and limitations

There is no doubt that the main and most powerful advantage of R-GCNs is the relation aware advantage when compared with GCNs. This feature allowed graph neural networks to move into the domain of Knowledge Graphs and created the base for many more models on spatial and temporal KGs. Other strength of R-GCN lies in its ability to be projected on more than one task such as this model was applied on link prediction and node classification. [21] reports on strong results in both settings, especially on relational benchmarks.

However, such ability of representation created a limitation for R-GCN in the number of parameters. The number of parameters grows rapidly as the number of relations in a graph increases, since each type of relation has its own weight matrix. This can lead to overfitting and scalability issues, especially in large knowledge graphs with many relation types. To address this, the authors introduce basis decomposition as regularization techniques. In this context, a basis refers to a shared transformation matrix that captures a reusable pattern of message transformation, while each relation learns how to combine these bases through relation-specific coefficients.

In basis decomposition, each relation-specific matrix is expressed as:

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

This method reduces the number of parameters and improves regularization, by using a shared fixed basis V_b at each layer l and learning a smaller relation-specific attention vector a_{rb} , that differentiates between each relation characteristics. This way, the number of parameters is noticeably reduced.

Another limitation lies in the representation of graphs in original R-GCN, as it was modeled to represent static/spatial KGs, it cannot represent temporal or spatiotemporal KGs. This limits its application directly on semantic trajectories, but it opens a door for adaptation either on the model itself or on the STKG representation such that the original R-GCN could be applied on it.

2.3.5 Time Entity Aware Graph Neural Network

2.3.5.1 Model family

Time-Aware Graph Neural Network for Entity Alignment between Temporal Knowledge Graphs (TEA-GNN), is a temporal graph neural network designed for entity alignment between temporal knowledge graphs [22]. Unlike static GNN models, TEA-GNN explicitly incorporates both relational and temporal information during message passing. It introduces reverse relations to capture edge direction and uses temporal information associated with facts, such as their start and end times, to enrich the representation of entities and relations. The model also uses a time-aware attention mechanism based on orthogonal transformations. These transformations help preserve the norms and relative distances of entity embeddings while adapting them to different temporal contexts. Therefore, TEA-GNN belongs to the family of temporal GNNs and is more suitable than static models for graphs where facts and relations evolve over time. The original paper reports that TEA-GNN outperforms several state-of-the-art GNN-based entity alignment models [22].

2.3.5.2 Knowledge graph representation

The structure of TEA-GNN is created and applied on temporal KGs, where time is represented as an extra attribute such that each triple (e_s, r, e_o) is extended to a quadruple (e_s, r, e_o, τ) containing the timestamp of each relation as explained before. In order to incorporate direction information and represent the beginning and end of a fact in a KG, an inverse relation was added for each fact such that we have now (e_s, r, e_o, τ_b) and $(e_o, r^{-1}, e_s, \tau_e)$ to represent the start and end of a fact.

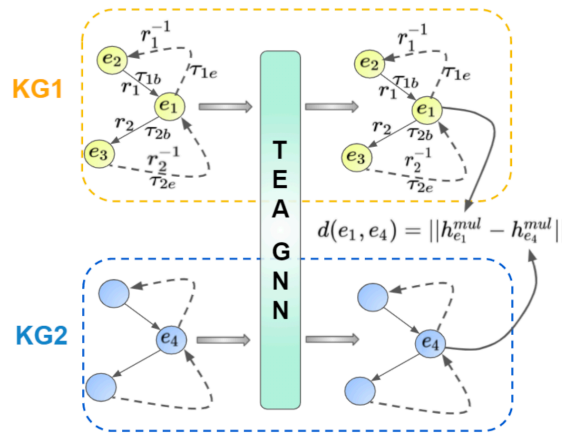


Figure 14 TEA-GNN Framework, dashed arrows show inverse relation [22]

As illustrated in Figure 14, the inverse relation (denoted by a dashed arrow) is a generated relation based on the original one. Taking as example in the figure relation r_1 between nodes e_2 and e_1 , the original fact is $(e_2, r_1, e_1, [\tau_{1b}, \tau_{1e}])$, to represent this fact in the KG authors extract two quadruples, $(e_2, r_1, e_1, \tau_{1b})$ that

indicates the beginning of the relation, and $(e_1, r^{-1}_1, e_2, \tau_{1e})$ that contains the inverse direction of the original relation and indicates the end of the fact. This representation allowed the model to handle: time points where $\tau_b = \tau_e$, time interval $\tau_b \neq \tau_e$, and missing points by using τ_0 [22].

2.3.5.3 TEA-GNN layer

The message computation for this model was adapted from previous models, to incorporate both temporal and relational information into the entity representations. To do so, the authors started by unifying the dimensionality of the vector space such that all entities, relations (including inverse relations), and timestamps are embedded into a shared vector space $\mathbb{R}^k$.

Let h_{e_i} , h_{r_j} and h_{τ_m} denote the embeddings of entity e_i , relation r_j , and timestamp τ_m , respectively.

- **Message computation**

To start with message computation and to ensure the stability in transformation for entities while incorporating time and relational contexts, the original paper of TEA-GNN introduced orthogonal transformation matrices for each timestamp and relation embedding, such that:

$$M_{r_j} = I - 2h_{r_j}h_{r_j}^T, \quad M_{\tau_m} = I - 2h_{\tau_m}h_{\tau_m}^T$$

where I is the identity matrix.

These matrices ensure that vector norms and relative distances between entity embeddings after transformation remains unchanged.

Then, for each neighboring node e_j of the target node e_i , an attention weight is computed for each neighboring node to control its contribution according to its importance for the target node. These weights are calculated using the orthogonal matrices through a time-aware attention mechanism to integrate both relational and temporal importance for each node.

For each incoming link from a neighboring entity e_j to the target entity e_i , two attention scores are computed: a time-specific importance $\alpha_{i,j,m}$ and a relation-specific importance $\beta_{i,j,m}$ associated with the m -th incoming link. These scores measure the contribution of neighbor e_j to the representation of e_i under the corresponding timestamp and relation.

Formally, the attention scores are computed as

$$\alpha_{i,j,m} = v_\tau^T \left[h_{e_i}^{in} \parallel M_{\tau_m} h_{e_j}^{in} \parallel h_{\tau_m} \right],$$

$$\beta_{i,j,m} = v_r^T \left[h_{e_i}^{in} \parallel M_{r_m} h_{e_j}^{in} \parallel h_{r_m} \right],$$

where $\parallel$ denotes vector concatenation, $h_{e_i}^{in}$ and $h_{e_j}^{in}$ are the input embeddings of entities e_i and e_j , h_{τ_m} and h_{r_m} are the embeddings of the timestamp and relation of the m -th link, and M_{τ_m} and M_{r_m} are the corresponding orthogonal transformation matrices for time and relation. The vectors v_τ and v_r are shared attention parameters that project the concatenated features into scalar attention scores.

Authors follow the GAT normalization in [20] for attention weights, and they normalized both $\alpha_{i,j,m}$ and $\beta_{i,j,m}$ passing them through a softmax function, so the final representation for attention weights is:

$$\omega_{i,j,m} = \frac{\exp(\alpha_{i,j,m})}{\sum_{e_j \in \mathcal{N}_i^e} \sum_{\tau_m \in \mathcal{L}_{ij}^\tau} \exp(\alpha_{i,j,m})}, \quad \nu_{i,j,m} = \frac{\exp(\beta_{i,j,m})}{\sum_{e_j \in \mathcal{N}_i^e} \sum_{r_m \in \mathcal{L}_{ij}^r} \exp(\beta_{i,j,m})}$$

where $\mathcal{N}_i^e$ denotes the set of neighboring entities of e_i , and $\mathcal{L}_{ij}^t$ and $\mathcal{L}_{ij}^r$ denote the sets of timestamps and relations associated with the links between e_j and e_i .

Now, given these attention weights, the authors compute the message between each neighboring node e_j of the target node e_i as:

$$\omega_{i,j,m} M_{\tau_m} h_{e_j}^{in} + v_{i,j,m} M_{r_m} h_{e_j}^{in}$$

- **Aggregation + update**

Finally, the updated version of entity e_i is obtained by aggregating all messages from its neighbors, followed by a non-linear activation function:

$$h_{e_i}^{out} = \sigma \left(\sum_{e_j \in \mathcal{N}_i^e} \sum_{[\tau_m, r_m] \in \mathcal{L}_{ij}} \omega_{i,j,m} M_{\tau_m} h_{e_j}^{in} + v_{i,j,m} M_{r_m} h_{e_j}^{in} \right).$$

This mechanism enables the model to learn entity representations that jointly capture structural, relational, and temporal information from the temporal knowledge graph.

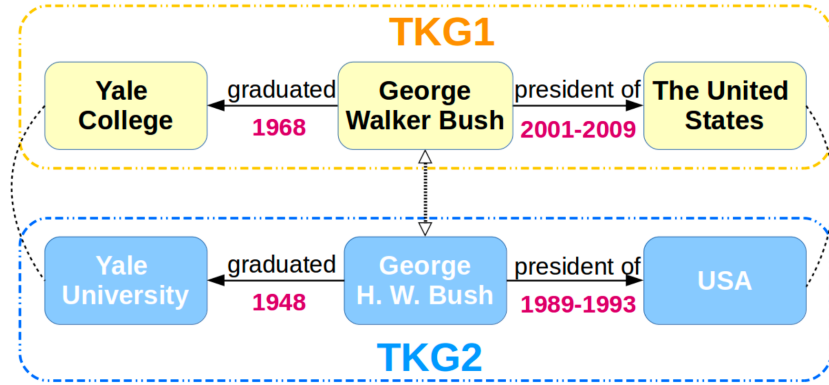


Figure 15 KG illustrating the importance of time in differentiating between similar entities [22]

2.3.5.4 Main tasks

The main task TEA-GNN was projected on is entity alignment, using the learned entity representations from two temporal KGs, the goal is to bring aligned entities closer while pushing non-aligned entities apart. Taking as example in Figure 15, the two KGs for George Bush are almost the same if we do not consider temporal facts, the model is able to distinguish between these two entities and increases the distance between their vector representations based on the different time intervals.

Several layers of TEA-GNN are used, and the output representation of each entity from these layers is concatenated, such that for entity e_i the final embedding representation after L TEA-GNN layers is:

$$[\widehat{h_{e_i}^{out}} = [h_{e_i}^{out(0)} \parallel h_{e_i}^{out(1)} \parallel \dots \parallel h_{e_i}^{out(L)}],]$$

To further incorporate temporal context, the average embedding of timestamps connected to the entity is concatenated with the cross-layer representation to form a multi-view entity representation. Entity alignment is then predicted based on the distance between the final representations of entities from two different knowledge graphs. For entities $e_i \in E_1$ and $e_j \in E_2$ the L1 distance is used:

$$d(e_i, e_j) = |h_{e_i}^{mul} - h_{e_j}^{mul}|_1.$$

The L1 distance, also known as Manhattan distance, measures the dissimilarity between two entity representations by summing the absolute differences between their corresponding embedding dimensions.

2.3.5.5 *Strengths and limitations*

TEA-GNN has several important strengths. First, it explicitly incorporates time information into the graph neural network, which addresses an important limitation of earlier time-agnostic entity alignment methods. Second, it jointly models relation semantics and temporal information through a time-aware attention mechanism, allowing it to distinguish entities with similar relational neighborhoods but different temporal contexts. Third, it avoids discretizing the temporal graph into independent snapshots, where a snapshot here means to collect all triples from a KG at a specific timestamp, but instead it treats timestamps as properties of links adding them to the triples becoming quadruples, which is efficient and flexible. Experimental results in the paper show that TEA-GNN significantly outperforms state-of-the-art baselines on multiple temporal knowledge graph alignment datasets [22].

However, TEA-GNN also has limitations. Its primary focus is entity alignment, not general temporal KG completion or broader downstream reasoning tasks. In addition, the model is designed specifically for pairwise alignment across two temporal KGs, which may limit its direct applicability to other spatio-temporal learning scenarios. Finally, although it incorporates temporal information effectively, its formulation is centered on link properties and alignment objectives, rather than on hierarchical semantic trajectories or episode-based representations such as those considered in our work. Therefore, TEA-GNN is highly relevant as a temporal and attention-based GNN model, but it is not directly designed for spatio-temporal semantic trajectory representation learning [22].

2.3.6 *Recurrent Evolution Graph Neural Network*

2.3.6.1 *Model family*

Recurrent Evolution Graph Neural Network (RE-GCN) learns temporal representation for entities and relations over timestamps, by modeling the Knowledge Graph sequence recurrently. It learns evolutionary representations of entities and relations by jointly capturing the structural dependencies within each snapshot, the sequential patterns across adjacent timestamps, and the static properties of entities.

This model is considered as a *spatiotemporal graph neural network* and more specifically as *recurrent spatiotemporal GNN* as it learns the representation from a temporal knowledge graph. The model combines:

- a relation-aware GCN for spatial/structural learning within each snapshot,
- recurrent gating mechanisms for temporal evolution,
- and a static graph constraint to preserve background entity semantics.

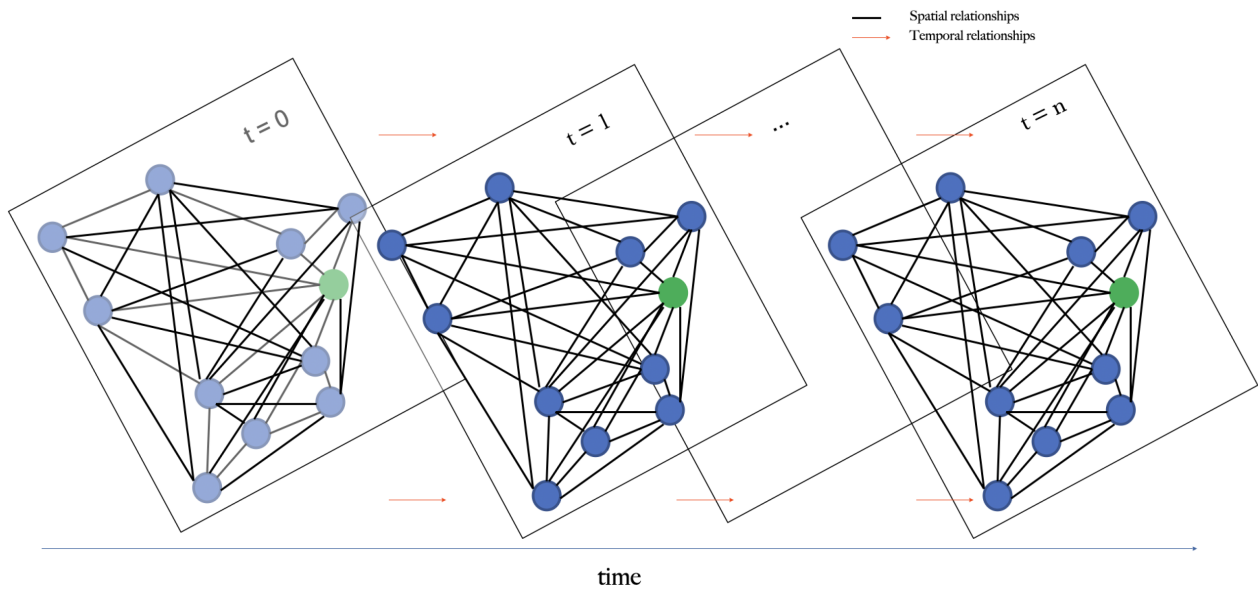


Figure 16 Illustrating spatiotemporal KG snapshots [33]

2.3.6.2 Knowledge graph representation

RE-GCN represents a temporal knowledge graph as a **sequence of relational graph snapshots**, rather than as isolated quadruples, or separate histories for each query. This allows the model to encode:

- structural dependencies among concurrent facts at each timestamp,
- temporal dependencies across adjacent timestamps,
- and background semantic information from a static graph.

A temporal knowledge graph is represented here as a sequence of snapshots:

$$G = \{G_1, G_2, \dots, G_t, \dots\}$$

As shown in Figure 16, at each time stamp t , there is a snapshot G_t constructed from the spatial relations inside the main graph G at the given time stamp t , such that a snapshot G_t is a set of triples from the main graph G where all have the same time stamp t . Each snapshot at timestamp t is denoted as:

$$G_t = (V, R, E_t)$$

where:

- V is the set of entities,
- R is the set of relations,
- E_t is the set of facts occurring at timestamp t .

A temporal fact is written as:

$$(s, r, o, t)$$

where s is the subject entity, o is the object entity, r is the relation, and t is the timestamp. For each fact, the inverse fact (o, r^{-1}, s, t) is also added to make information flow in both directions during message passing [23].

2.3.6.3 RE-GCN layer

A RE-GCN learns temporal embeddings for entities and relations in three main steps:

- Structural modelling function performed by a R-GCN
 - Temporal modelling represented by time gate for entities and relation
 - Static graph constraint
 - **Message computation**
- At each timestamp t , a relation-aware GCN (R-GCN) is applied to a graph snapshot G_t to capture the spatial and structural dependencies between entities through the relations that connect them [23]. An w -layer R-GCN is used to model the structural dependencies, more specifically, for an object entity o at layer $l \in [0, w - 1]$ given its subject entities s with a relation r , the message passing applied for structural modelling:

$$W_1^l(h_{s,t}^l + r_t)$$

- **Aggregation**

The aggregation is performed through a normalized sum over all incoming relational neighbors of the target entity.

Thus, the relation-aware message at layer l for the object entity o at timestamp t is computed as:

$$h_{o,t}^{l+1} = f\left(\frac{1}{c_o} \sum_{(s,r), \exists(s,r,o) \in E_t} W_1^l(h_{s,t}^l + r_t) + W_2^l h_{o,t}^l\right)$$

where:

- $h_{o,t}^l$ and $h_{s,t}^l$ are the embeddings of entities o and s at layer l and timestamp t ,
- r_t is the embedding of relation r at timestamp t ,
- W_1^l is the aggregation weight matrix,
- W_2^l is the self-loop weight matrix,
- c_o is a normalization constant equal to the in-degree of entity o ,
- $f(\cdot)$ is the activation function, specifically ReLU here.

After ω layers of R-GCN the model produces a structural representation of entities at timestamp t :

$$H_t^\omega$$

which represents the embeddings of all entities in the graph snapshot G_t at time t .

• Update

After structural aggregation, the model incorporates temporal evolution by combining the structural embeddings of the current timestamp H_t^ω with the embeddings of the previous time stamp H_{t-1}^ω using a time-gated mechanism [23]:

$$H_t = U_t \odot H_t^\omega + (1 - U_t) \odot H_{t-1}$$

U_t is a time gate learned by the model, computed as:

$$U_t = \sigma(W_4 H_{t-1} + b)$$

where W_4 is the weight matrix of the time gate, and $\sigma(\cdot)$ is the sigmoid.

This mechanism allows the model to balance between:

- Newly observed structural information from the current graph, and
- Historical entity behavior captured in previous embeddings.

As a result, the entity embeddings H_t captures both structural and temporal information.

For relation evolution, relation embeddings are influenced by the entities participating in those relations. For a given relation r , the set of entities involved in facts containing that relation at timestamp t is defined as:

$$V_{r,t} = \{i \mid (i, r, o, t) \text{ or } (s, r, i, t) \in E_t\}$$

These entities give an idea about the behavior of this relation in the current KG. The embeddings of these entities are used to construct the representation of the relational context.

To model the temporal evolution of relations, the model aggregates the embeddings of entities involved in relation r using mean pooling. The pooled representation is then concatenated with the previous embedding of relation r :

$$\vec{r}'_t = [\text{pooling}(H_{t-1, V_{r,t}}); \vec{r}]$$

where $\vec{r}$ denotes the embedding of relation r .

The resulting embedding $\vec{r}'_t$ is used as input to a Gated Recurrent Unit (GRU), which updates the relation embeddings:

$$R_t = \text{GRU}(R_{t-1}, R'_t)$$

where R'_t is a relation matrix consists of $\vec{r}'_t$ of all relations [23].

This mechanism allows relation embeddings to evolve over time based on:

- their previous representations,
- the entities participating in the corresponding relations at a given timestamp.

- **Static graph constraint**

The authors claimed that in KGs some characteristics for entities are static, so to keep these static structures in the embedding representations in addition to the learned structural and temporal representations, they proposed a static graph $G_s = (V_s, R_s, E_s)$, where entities are connected to property nodes such as types, categories, or other time-independent attributes. Then, static embeddings are computed for the entities and used as a reference for the temporal embeddings. More specifically, the authors impose an angular constraint between the temporal embedding and the static embedding, so that the temporal representation is not allowed to move too far away from the static semantic structure. This way the authors claim that they prevented temporal embeddings to lose this static structure which turn to be important in some cases.

To obtain static embeddings for entities, a one-layer R-GCN is applied:

$$h_i^s = \gamma \left(\frac{1}{c_i} \sum_{(r_s, j), \exists (i, r_s, j) \in E^s} W_{r_s} h_j^s \right)$$

where $\gamma(\cdot)$ is *ReLU*, h_i^s is the static embedding for entities in the static graph, W_{r_s} is the relation matrix of r_s in the static graph G_s , and c_i is a normalization constant corresponding to the number of entities connected to entity i

In simple terms, this equation computes the static representation of an entity by aggregating information from its static properties through the relations of the static graph.

To ensure consistency between the static and temporal evolutions, the angle between these two embeddings is constrained through:

$$\theta_x = \min(\gamma x, 90^\circ)$$

where γ controls how fast the allowed angle increases, and $x \in [0, 1, \dots, m]$ denotes the temporal step. This means that, for recent timestamps, the temporal embedding is expected to remain close to the static embedding. As time goes further back in the historical sequence, the model allows a larger deviation, because more temporal changes may have occurred. However, the maximum angle is limited to 90° , meaning that the temporal and static embeddings should not become opposite or semantically inconsistent.

The angle constraint is implemented using cosine similarity. The cosine similarity between the static embedding and the temporal embedding is given by:

$$\cos(h_i^s, h_{t-m+x,i}) = \frac{h_i^s \cdot h_{t-m+x,i}}{|h_i^s| |h_{t-m+x,i}|}$$

The embedding $h_{t-m+x,i}$ denotes the temporal embedding of entity i at a specific timestamp in the historical sequence $t - m + x$. Therefore, h_i^s represents what is generally stable about the entity, while $h_{t-m+x,i}$ represents how the entity is evolving at a given time.

Cosine similarity is directly related to the angle between two vectors. A high cosine similarity means that the two embeddings point in a similar direction, while a low cosine similarity means that they are far apart in the embedding space. Therefore, requiring the angle to be smaller than θ_x is equivalent to requiring the cosine similarity to be greater than $\cos(\theta_x)$.

To enforce the constraint that the angle between the two embeddings does not exceed θ_x , the following loss function is introduced:

$$L_x^{st} = \sum_{i=0}^{|V|-1} \max\{\cos \theta_x - \cos(h_i^s, h_{t-m+x,i}), 0\}.$$

This loss penalizes an entity only when the cosine similarity between its static embedding and its temporal embedding becomes smaller than the required threshold. In other words, if the temporal embedding stays within the allowed angle from the static embedding, no penalty is added. If it moves too far away, the model receives a penalty that pushes the temporal representation back toward the static semantic structure.

Overall, this constraint regularizes the temporal embeddings by keeping them aligned with the static embeddings. It allows entities to evolve over time while still preserving their fundamental semantic properties, such as type, category, or country.

2.3.6.4 Main tasks

After processing the sequence graph snapshots, the model produces temporal embeddings for entities and relations:

$$H_t \quad \text{and} \quad R_t$$

These embeddings are then used by a scoring function (such as ConvTransE) to compute probabilities for candidate facts, enabling tasks such as entity prediction and relation prediction.

- Entity prediction

Given a query:

$$(s, r, ?, t + 1)$$

RE-GCN aims to predict the missing object entity at the future timestamp $t+1$. The model first uses the historical graph sequence $G_{t-m+1:t}$ to learn the entity embedding matrix H_t and the relation embedding matrix R_t . These embeddings summarize the structural and temporal information observed in the previous graph snapshots. Therefore, the prediction can be written as:

$$\vec{p}(o \mid s, r, G_{t-m+1:t}) = \vec{p}(o \mid s, r, H_t, R_t)$$

This means that the probability of each candidate object entity is computed using the learned representations of the subject s , the relation r , and the historical temporal context encoded in H_t and R_t .

In the implementation, the probability vector over all entities is computed using a ConvTransE decoder:

$$\vec{p}(o \mid s, r, H_t, R_t) = \sigma(H_t \text{ConvTransE}(s_t, r_t))$$

where s_t and r_t are the embeddings of the subject and relation at timestamp t , and $\sigma(\cdot)$ is the sigmoid function. The term $\text{ConvTransE}(s_t, r_t)$ produces a query representation from the subject and relation embeddings. This representation is then compared with all entity embeddings in H_t . As a result, the model obtains a probability vector over all candidate entities, and the entity with the highest probability is selected as the predicted object.

- Relation prediction

Given a Query:

$$(s, ?, o, t + 1)$$

RE-GCN aims to predict the missing relation between the subject entity s and the object entity o at the future timestamp $t+1$. As in entity prediction, the model relies on the historical sequence $G_{t-m+1:t}$ to obtain the temporal entity and relation representations H_t and R_t . The prediction is therefore written as:

$$\vec{p}(r | s, o, G_{t-m+1:t}) = \vec{p}(r | s, o, H_t, R_t)$$

This probability vector assigns a score to each possible relation type, indicating how likely it is to connect s and o at the next timestamp.

In the implementation, the probability vector over all entities is computed using a ConvTransE decoder:

$$\vec{p}(r | s, o, H_t, R_t) = \sigma(R_t \text{ ConvTransE}(s_t, o_t))$$

where s_t and o_t are the embeddings of the subject and object entities at timestamp t . The decoder first combines the subject and object embeddings into a query representation. This representation is then compared with all relation embeddings in R_t , producing a probability score for each candidate relation. The relation with the highest probability is selected as the predicted relation.

In both tasks, ConvTransE acts as a scoring function or decoder. It transforms the available elements of the query into a vector representation and compares it with all possible candidates. For entity prediction, the candidates are entities in H_t . For relation prediction, the candidates are relations in R_t . This allows RE-GCN to perform temporal reasoning by using the embeddings learned from previous graph snapshots.

2.3.6.5 Strengths and limitations

RE-GCN has several strengths. First, it jointly models structural, temporal, and static semantic information, which makes it more expressive than methods that only use local query history or static graph structure. Second, by modeling the whole KG sequence rather than processing queries one by one, it is significantly more efficient than query-based temporal baselines. Third, it supports both entity prediction and relation prediction within a unified framework. The authors report substantial improvements over baselines on multiple benchmarks, including up to 11.46% MRR improvement, or Mean Reciprocal Rank, that measures how highly the model ranks the correct answer among all candidate entities or relations, meaning a higher MRR indicates that the correct answer is usually ranked closer to the top of the prediction list. Moreover, another improvement is 82× speedup relative to the state-of-the-art baseline [23].

However, RE-GCN also has limitations. Its temporal modeling is based on discrete snapshots, so it is more naturally suited to snapshot-based temporal KGs than to fine-grained continuous-time event streams. In addition, although it captures temporal evolution effectively, it is designed mainly for temporal fact reasoning, not for tasks such as semantic trajectory alignment or multi-view territory representation. Finally, its architecture relies on recurrent evolution and decoder-based fact prediction, which may be less directly interpretable than models designed around explicit episode or trajectory representations. Still, RE-GCN remains highly relevant as a strong temporal GNN baseline for reasoning over evolving knowledge graphs.

2.3.7 Time Aware Heterogeneous Graph Neural Network

2.3.7.1 Model family

Time Aware Heterogeneous Graph Neural Network (THAM) is a novel approach designed to capture temporal dynamics of an evolving object such as a disease, or a patient diagnosis. Representing the heterogeneous and rich semantics of each object evolution [24]. The model incorporates temporal data into visit-level embeddings learned from a heterogeneous GNN and leverages a time-aware transformer with its attention mechanism to produce patient representation. The model is therefore considered a hybrid model combining two neural networks, and **Time Aware Heterogeneous GNN** is classified as a **spatiotemporal representation learning framework** for health event prediction.

More specifically, the model integrates seven major ideas:

- hierarchical disease representation based on domain specific (medical) ontology,
- heterogeneous graph relation learning over disease–drug and disease–disease relations,
- time-aware sequence modeling over visit trajectories,
- a Time-aware Transformer Encoder for modelling visit sequences,
- a preliminary attention mechanism,
- a comprehensive attention mechanism,
- and an adaptive attention merging mechanism.

2.3.7.2 Knowledge graph representation

In THAM, to represent a patient visit trajectory, each patient is represented by a sequence of her/his visit:

$$p = \{V_1, V_2, \dots, V_T\}$$

where each visit V_t contains a set of medical codes and drugs:

$$V_t = \{C_t, D_t\}$$

Here:

$$C_t \subseteq C$$

is the set of medical codes or diseases diagnosed during visit t , and:

$$D_t \subseteq D$$

is the set of drugs used during the same visit.

Based on the set of diseases and drugs of each patient, and in order to capture the relations of each visit, the authors consider a pair of drug-disease heterogeneous co-occurrence graph and a disease ontology co-occurrence graph, denoted as $M = \{M_{DC}, M_{CC}\}$ [24].

M_{DC} is a heterogeneous drug-disease graph that capture the relations in each visit for a patient, between the diagnosed disease c_j and its drugs used to treat it d_i . This allows the model to capture hidden similarities between diseases that may be treated by the same or similar drugs. The graph is represented by the matrix:

$$B_{DC} \in R^{|D| \times |C|}$$

where $B_{DC}[i][j]$ stores the co-occurrence strength between drug d_i and disease c_j .

M_{CC} is a disease ontology graph that is made to connect diseases diagnosed in the same visit. Such that, if two diseases c_i and c_j were diagnosed for the same patient in the same visit, an edge is created between these two diseases. However, THAM assumes that the influence between diseases is not necessarily symmetric. Therefore, the disease-disease adjacency matrix is asymmetric:

$$A_{CC} \in R^{|C| \times |C|}$$

This matrix represents the degree of influence or co-occurrence between diseases and improves interpretability by making disease relationships explicit.

In a spatiotemporal KG setting, this representation can be adapted by replacing medical visits with temporal KG episodes or snapshots:

$$G = \{G_1, G_2, \dots, G_t, \dots, G_T\}$$

where each snapshot is:

$$G_t = (V_t, R_t, E_t)$$

Here:

- V_t is the set of entities appearing at timestamp t ,
- R_t is the set of relation types,
- E_t is the set of facts or edges observed at timestamp t ;

Each snapshot or episode, denoted as a set of triples occurring at a specific timestamp t where a timestamp could be either time instant, or an interval, can be treated as a local heterogeneous KG. A heterogeneous or relational GNN can then encode each episode into an episode embedding, and the sequence of episode embeddings can be processed by a temporal encoder such as the Time-aware Transformer Encoder.

2.3.7.3 THAM layer

The THAM layer can be understood as a multi-stage layer that transforms raw heterogeneous visit information into a final temporal representation. It contains three major parts:

1. heterogeneous graph relation learning,
2. time-aware visit representation,
3. transformer-based temporal attention and adaptive attention merging.

- **Heterogeneous graph relation learning**

THAM designed a Hetero-Graph Neural Network based on R-GCN, that uses the heterogeneous co-occurrence graph and the disease ontology co-occurrence graph to derive meaningful representations of diseases in each visit.

They first initialize disease/code and drug representations. The medical code embeddings are initialized using hierarchical disease knowledge, based on the disease hierarchy tree, such that each disease is characterized by one single parent defining the family this disease belongs, and several leaf nodes denoting the other diseases that can come from this disease [24]. For example, pneumonia has respiratory disease as its parent node, and bacterial pneumonia, viral pneumonia, and aspiration pneumonia as its leaf nodes, since they are more specific diseases branched from it. If the disease hierarchy has H levels, with each layer h hosting m_h nodes, the representation of a disease c_i is obtained by concatenating its embeddings from different hierarchy levels:

$$L_i = l_1^i \oplus l_2^i \oplus \dots \oplus l_H^i$$

where $\oplus$ denotes concatenation.

The final hierarchical disease embedding matrix is:

$$L \in R^{|C| \times H m_c}$$

This allows each disease to be represented not only by its specific code, but also by its higher-level semantic categories.

The initial hidden representations are:

$$\begin{aligned} H_D^{(0)} &= N \\ H_C^{(0)} &= L \end{aligned}$$

where:

- N is the drug embedding matrix,
- L is the hierarchical medical code embedding matrix,
- $H_D^{(l)}$ is the drug representation at layer l ,

- $H_C^{(l)}$ is the medical code representation at layer l.

The aggregation for drug nodes is:

$$M_D^{(l)} = H_D^{(l)} + B_{DC} H_C^{(l)} W_{CD}^{(l)}$$

where $W_{CD}^{(l)}$ is a trainable parameter utilized for mapping medical code embeddings to the dimension of drug embeddings

This means that each drug representation is updated using the representations of its connected diseases.

The aggregation for disease/code nodes is:

$$M_C^{(l)} = H_C^{(l)} + B_{DC}^T H_D^{(l)} W_{DC}^{(l)} + A_{CC} H_C^{(l)}$$

where $W_{DC}^{(l)}$ is also a trainable parameter used for mapping drug embeddings to the dimension of medical code embeddings.

This means that each disease representation is updated using both:

- drug-disease co-occurrence information,
- and disease-disease co-occurrence information.

Then, the update step is:

$$H_{\{D,C\}}^{(l+1)} = \sigma \left(\text{BatchNorm} \left(M_{\{D,C\}}^{(l)} W_{\{D,C\}}^{(l)} \right) \right)$$

where σ is the activation function. In the original paper, THAM uses **LeakyReLU** [24].

After L layers of heterogeneous graph learning, the model obtains the final disease/code representation:

$$H_C$$

which is then used to build visit embeddings.

• Time-aware visit representation

After obtaining the final disease/code representations from the GNN, using the data from the drug-disease heterogeneous co-occurrence graph, and disease ontology co-occurrence graph, the model constructs a time-aware visit representation. For each patient visit, all embeddings for the diseases diagnosed in that visit are collected and their average is computed. Therefore, the initial embedding vector o_t for visit t is represented as:

$$o_t = \frac{1}{|C_t|} \sum_{c_j \in C_t} H_C^j$$

Till now the initial embedding vector o_t is still static. To inject time information, the authors suggest a learnable function to represent time where the effect of time is not severe on the embeddings, due to the big time difference between one visit and the other for a patient. If the raw time value is injected directly into the representation, its numerical scale may dominate the smaller embedding values that describe medical codes, drugs, or other node attributes. As a result, the model may focus too much on the magnitude of the time interval and lose part of the semantic information contained in the original embedding. For this, they proposed a function that integrate temporal data into the visit vector, using two learnable matrices and two bias factor, such that:

$$f_t = W_f \left(1 - \tanh^2 \left(W_e \frac{\Delta_t}{180} + b_e \right) \right) + b_f$$

Where W_f and W_e are two linear transformations, b_f and b_e are two bias factors, Δ_t is the time difference between the visit t and $t+l$.

So, we can write the visit representation after this step as:

$$v_t = o_t + f_t$$

This step is important because this formulation for time representation is not strict on time difference value only. Instead, by using this learnable time function, the model can encode the effect of time in a controlled way. This allows temporal information to influence the representation without overwhelming the static embedding features.

- **Transformer-based temporal attention and adaptive attention merging**

After computing the visit sequence:

$$V = [v_1, v_2, \dots, v_T]$$

THAM adds positional encoding:

$$v'_t = v_t + Pos_t$$

where:

$$Pos(t, 2i) = \sin\left(\frac{t}{10000^{2i/m}}\right)$$

$$Pos(t, 2i + 1) = \cos\left(\frac{t}{10000^{2i/m}}\right)$$

Pos_t is the positional encoding vector added to the visit representation v_t , while $Pos(t, 2i)$ and $Pos(t, 2i + 1)$ define its even and odd dimensions using sine and cosine functions to encode the position or time step t in a continuous vector form.

The sequence is then passed through a Time-aware Transformer Encoder:

$$[h_1, h_2, \dots, h_T] = TTE([v'_1, v'_2, \dots, v'_T])$$

where each h_t is a contextualized visit representation, where h_t represents the visit at time t after taking into account the other visits in the patient's sequence, not only the information inside that visit itself. This way, the transformer allows the model to capture long-range dependencies between visits or episodes.

THAM then computes two types of attention.

First, the preliminary attention weight is computed as:

$$\alpha = \text{Softmax}([h_1, h_2, \dots, h_T]w_\alpha)$$

This attention estimates the direct importance of each previous visit.

Second, THAM defines the latest hidden state as the comprehensive visit vector:

$$h^* = h_T$$

Then it transforms h^* into a query vector:

$$Q = \text{LeakyReLU}(W_Q h^* + b_Q)$$

where W_Q and b_Q are both trainable parameters.

Each temporal interval Δ_t is transformed into a key vector:

$$K_t = \text{LeakyReLU}\left(W_k \left(1 - \tanh^2\left(W_t \frac{\Delta_t}{180} + b_t\right)\right) + b_k\right)$$

The comprehensive attention weight is computed by scaled dot-product attention:

$$\beta = \text{Softmax}\left(\frac{QK^T}{\sqrt{q}}\right)$$

where $\sqrt{q}$ is the scaling factor, where q is usually the dimension of the query/key vectors.

This second attention mechanism allows the model to evaluate previous visits from the perspective of the most recent visit, which is assumed to summarize the patient's disease progression.

Finally, THAM merges the two attention weights adaptively. The comprehensive vector is mapped into a two-dimensional weight vector:

$$\delta = \text{Softmax}(W_x h^* + b_x)$$

Then:

$$\gamma = \alpha \oplus \beta$$

$$\eta = \gamma \odot \delta$$

where $\odot$ denotes element-wise multiplication.

The final normalized attention score is:

$$\eta'_t = \frac{\eta_t}{\sum_{i=1}^T \eta_i}$$

The final patient or trajectory representation is obtained by attention pooling:

$$O = \sum_{t=1}^T \eta'_t h_t$$

This representation is then passed to an MLP with a sigmoid activation for prediction. The model is trained using binary cross-entropy loss [24].

2.3.7.4 Main tasks

THAM has been applied on two main tasks for health event prediction.

The first task is **diagnosis prediction**. Based on the final patient trajectory representation, the model predicts based on the diagnosed diseases from the previous visits of the patient, the set of diseases that may occur in the next visit. This is formulated as a multi-label classification problem.

The second task is **heart failure prediction**. This task is a binary classification problem, such that given the patient trajectory representation based on the previous visits, the model predicts if the patient will be diagnosed with heart failure in the next visit or not.

2.3.7.5 *Strengths and limitations*

The first strength of THAM is that it combines **heterogeneous graph learning and temporal sequence modelling**. The model does not only learn from the sequence of visits but also learns meaningful disease and drug representations through heterogeneous graph structures. The second strength is the use of **domain knowledge**. THAM uses hierarchical medical knowledge to enrich disease representations. This allows the model to represent diseases not only as isolated codes, but also according to their broader semantic categories. The third strength is that THAM captures **hidden relationships between diseases through drug usage**. If two diseases are treated using the same or similar drugs, the model can use this information to infer potential similarity between them. This is especially useful when direct disease co-occurrence information is sparse. The fourth strength is its **time-aware modelling**. THAM explicitly integrates time intervals into visit embeddings and uses temporal information in the comprehensive attention phase. This allows it to model irregular temporal gaps between visits. The fifth strength is the **adaptive attention merging mechanism**. Instead of relying on only one attention score, THAM combines preliminary attention α and comprehensive attention β . This helps the model consider both the direct importance of each visit and its relevance to the latest global state. Finally, THAM shows strong empirical performance. In the original experiments, it outperformed several baseline models on diagnosis prediction and heart failure prediction using MIMIC-III and MIMIC-IV datasets [24].

Although THAM had many strengths, it had also some limitations. As the first limitation, THAM was originally designed for **Electronic Health Records**, not for general spatiotemporal knowledge graphs. Therefore, applying it to semantic trajectories or SETT KGs requires adaptation of the graph construction, node types, relation types, and prediction heads. The second limitation is computational complexity. THAM contains several components: hierarchical embeddings, heterogeneous GNN layers, transformer encoder layers, temporal functions, and two attention mechanisms. This makes it more complex than simpler recurrent or GCN-based models.

However, THAM stands as a strong foundation framework to represent trajectories. In a spatiotemporal KG setting, THAM representation can be adapted by replacing medical visits with temporal KG episodes or snapshots:

$$G = \{G_1, G_2, \dots, G_t, \dots, G_T\}$$

where each snapshot is:

$$G_t = (V_t, R_t, E_t)$$

where:

- V_t is the set of entities appearing at timestamp t ,
- R_t is the set of relation types,
- E_t is the set of facts or edges observed at timestamp t .

In this case, each snapshot or episode can be treated as a local heterogeneous KG subgraph. A heterogeneous or relational GNN can then encode each episode into an episode embedding, and the sequence of episode embeddings can be processed by a temporal encoder such as the Time-aware Transformer Encoder. In this way, THAM can be adapted to learn representations of complete semantic trajectories. Then we can apply several downstream tasks on the learned embeddings such as node similarity, classification, node class prediction, and next-episode prediction.

2.4 Synthesis

The previous subsections reviewed several graph neural architectures and related models, starting from general graph models such as GCN and GAT, then moving toward relational, temporal, and heterogeneous models such as R-GCN, TEA-GNN, RE-GCN, and THAM. The objective of this synthesis is to compare these models according to their ability to represent the main characteristics of Semantic Trajectories of Territories (SETT).

SETT data are not simple static graphs. They are heterogeneous Knowledge Graphs composed of several semantic layers, multiple relation types, spatial information, temporal evolution, and thematic trajectories organized into episodes and events. Therefore, a model suitable for SETT should ideally be able to capture: (i) graph structure, (ii) relation types, (iii) temporal evolution, (iv) spatial or trajectory-related information, and (v) heterogeneous semantic information.

Table 1 summarizes the main characteristics of the studied models and highlights their limitations with respect to SETT representation.

Model	Model type	KG support	Relation awareness	Temporal modeling	Trajectory representation	Main task	Suitability for SETT
GCN [32]	Message-passing GNN	Not specifically designed for KGs	No relation modeling	No temporal modeling	No trajectory representation	Node class prediction	--
GAT [20]	Attention-based GNN	Not specifically designed for KGs	No relation modeling	No temporal modeling	No trajectory representation	Node class prediction	--
R-GCN [21]	Relational aware GCN	Yes	Yes, through relation-specific transformations	No temporal modeling	No trajectory representation	Entity classification and link prediction	+
TEA-GNN [22]	Time-aware GNN for temporal KGs	Yes	Yes, using R-GCN	Partially, time is encoded as relation/edge information	No trajectory representation	Entity alignment between two temporal KGs	+
RE-GCN [23]	Recurrent GCN-based temporal KG model	Yes	Yes, using R-GCN	Yes, through recurrent evolution over graph snapshots	No trajectory representation	Temporal entity and relation prediction	+
THAM [24]	Time-aware heterogeneous graph Transformer	Partially, through heterogeneous medical graphs	Yes, through heterogeneous graph relation learning	Yes, through time encoding, positional encoding, and Transformer-based sequence modeling	Sequence oriented representation of trajectories	Next medical visit and diagnosis prediction	++

Table 1 Summary of the studied GNN models

Legend: -- = not suitable; - = less suitable; + = partially suitable; ++ = suitable.

From this comparison, we can observe that each model captures only part of the requirements imposed by SETT. GCN and GAT provide the basic message-passing mechanisms for graph representation learning, but they are not sufficient for SETT because they do not explicitly model relation types or temporal evolution. R-

GCN improves this by introducing relation-specific message passing, making it more suitable for Knowledge Graphs. However, it remains static and therefore cannot directly capture the evolution of territorial trajectories over time.

Temporal models such as TEA-GNN and RE-GCN are closer to the needs of SETT because they introduce temporal information into the graph representation. TEA-GNN uses temporal information for entity alignment between temporal KGs, while RE-GCN represents a temporal KG as a sequence of graph snapshots and learns evolving entity and relation representations. Nevertheless, these models are mainly designed for temporal KG completion or alignment tasks, and they do not directly address the specific structure of SETT, where each territory is represented through thematic trajectories, episodes, events, spatial information, and semantic attributes.

THAM is particularly interesting because it combines heterogeneous graph learning with time encoding, positional encoding, transformer-based temporal modeling, and attention over a sequence. Although it was originally proposed in the medical domain, its general idea is relevant to SETT: each patient visit can be seen as a local heterogeneous graph, and the patient history can be modeled as a sequence of visits. Similarly, in SETT, each territory episode can be represented as a local heterogeneous KG snapshot, and the full territory trajectory can be modeled as a sequence of episode snapshots.

Therefore, the synthesis of the state-of-the-art shows that no studied model can be directly applied to SETT without adaptation. A suitable model for SETT should combine the relation-aware representation capacity of R-GCN, the temporal snapshot perspective of RE-GCN, and the sequence modeling capacity of Transformer-based architectures such as THAM. This observation motivates the proposed approach, where each territory episode is first encoded as a heterogeneous relational KG snapshot, and the sequence of episode representations is then modeled to capture the temporal evolution of the full territory trajectory.

2.5 Summary

This chapter provided the theoretical and methodological background necessary for understanding the proposed work. Section 2.1 introduced the main types of Knowledge Graphs, including static, spatial, temporal, and spatiotemporal Knowledge Graphs. Section 2.2 then focused on Semantic Trajectories of Territories (SETT), explaining their multilayer structure and showing why they represent a complex spatiotemporal and semantic data structure.

Section 2.3 introduced the main principle of graph representation learning through Graph Neural Networks, with a particular focus on message passing as the core mechanism used to learn node and graph representations. Section 2.4 presented a state-of-the-art study of several GNN-based architectures, discussing their main characteristics, architectures, strengths, and limitations. Finally, Section 2.5 synthesized these models according to their ability to represent the specific characteristics of SETT trajectories.

Overall, this chapter showed that existing GNN models capture only part of the requirements needed for SETT representation. Some models are effective for structural graph learning, others for multi-relational Knowledge Graphs, and others for temporal modeling. However, SETT requires a model that can combine local semantic graph encoding with temporal trajectory modeling. Based on this observation, the next chapter introduces the proposed contribution: the SETT-GNN model.

3 A GNN-Based Spatiotemporal Representation Learning Approach for SETT Knowledge Graphs

This chapter introduces the proposed SETT-GNN model, a spatiotemporal representation learning architecture designed for Semantic Environmental Trajectories of Territories represented as Knowledge Graphs. The model is inspired by THAM and adapted to the specific structure of SETT data. It combines a Relational Graph Convolutional Network (R-GCN) for encoding the local semantic structure of each episode snapshot with a Transformer-based temporal encoder enriched with time and positional encodings.

This chapter presents first the problem addressed by the proposed approach, focusing on the challenges introduced by SETT thematic trajectories and their episode-based structure. Then it describes how the original SETT Knowledge Graph is transformed into a sequence of territory-episode snapshots that can be used as input to the model. After that, the full architecture of SETT-GNN is introduced, including the local graph encoder, the temporal and positional encoding components, and the Transformer encoder used to learn the evolution of each territory trajectory.

Finally, this chapter positions the proposed model with respect to the state-of-the-art study presented in [Chapter 2](#). In this sense, SETT-GNN is not introduced as an independent architecture, but as a continuation of the previous analysis, combining and adapting ideas from relational GNNs, temporal Knowledge Graph models, and Transformer-based sequence modeling.

3.1 Problem Definition: Representation Learning on SETT KG

Semantic Environmental Trajectories of Territories-SETT dataset used in this work is organized into four seasonal datasets, in addition to a combined all-year dataset. The dataset is collected in three regions (Evian, Fribourg, and Grand-Geneve). Each region is represented by a KG describing the three layers introduced in [Section 2.2](#).

As previously explained, each territory trajectory is composed of three layers constructing the semantics and characteristics of trajectories. The thematic layer, which is the last level, when added above the structural and raw levels, it results in five parallel thematic trajectories describing all together the territory trajectory, defined as: temporal, urbanization, snow, vegetation, and water trajectories. Each thematic trajectory contains episodes and events, where each episode starts and ends with an event and is defined by its own time interval.

This special structure of spatiotemporal knowledge graphs requires a dedicated representation-learning framework to deal with it. After the study of graph neural networks, GCN or R-GCN can capture structural relations in a KG, but they do not model temporal evolution. On the other hand, temporal GNNs (TEA-GNN, RE-GCN) can model evolution, but many are designed for homogeneous graphs or standard temporal edges and for a specific task only, as entity alignment for TEA-GNN. Models such as RE-GCN show how recurrent temporal evolution can be used for temporal KGs, and how we can represent a TKG in a sequence of snapshots. THAM is especially interesting as it combines heterogeneous graph learning, time encoding, positional encoding, transformer-based temporal modelling, and attention over a sequence of visits.

The proposed approach is motivated by the need to learn representations from SETT KGs at two complementary levels: the local semantic structure of each episode and the temporal evolution of the full trajectory. Inspired by models such as THAM [\[24\]](#), the proposed architecture first encodes each territory episode in the KG using a relational graph neural network, then injects temporal and positional information, and finally models the sequence of episode embeddings using a Transformer encoder. As the model is applied on SETT KGs, it is called **SETT-GNN model**.

3.2 Input Data: SETT Knowledge Graphs and Trajectory Episodes

A SETT knowledge graph is constructed using Semantic Web standards, it is based on RDF Data Cube, and OWL-Time from W3C [\[https://www.w3.org/TR/owl-ref/\]](https://www.w3.org/TR/owl-ref/), used to build the graph and define its ontology [\[15\]](#). The need for a specific representation for such dataset is essential to start with SETT model pipeline. Based on the structure of territory trajectories where each territory is represented by its thematic, structural, and raw

levels, it is worth to note that each territory can be represented as an independent trajectory. This structure of trajectories allows us to separate territories from each other, such that a territory trajectory made of several episodes and semantic information can be represented as a sequence of territory-episode KG snapshots $G_{y_i} = \{G_{y_i}^{e_1}, G_{y_i}^{e_2}, \dots, G_{y_i}^{e_T}\}$, where each snapshot is $G_{y_i}^{e_j} = (V_{e_j}, R_{e_j}, E_{e_j})$, where V_{e_j} is the set of entities belonging to a territory y_i and one of its episodes e_j , R_{e_j} is the set of edge types belonging to the same territory y_i and its episodes e_j , and E_{e_j} is the set of facts or edges observed in the KG snapshot for territory y_i and its episodes e_j .

In a SETT KG, each edge is written in its original form as a triple (s, r, o) where s is the subject entity, r is the relation, and o is the object entity. The time interval for each snapshot is saved with the KG as an external feature to be integrated through the embeddings afterward by a time encoder.

Starting from the SETT KG dataset, we extract first snapshots based on each thematic trajectories of territories, so for each territory we collect all snapshots for all episodes in each theme independently. Then we combine all theme-snapshots that belong to the same territory with the same episode index together in one *territory-episode KG snapshot*. The time interval is collected for each episode in all theme-snapshots, and then we take the boundary time interval that cover all episodes together. This representation is possible due to the structure of territory trajectory that allow us to work on each thematic trajectory alone. This results as an input, a heterogeneous rich semantic KG, with a temporal features saved for each snapshot.

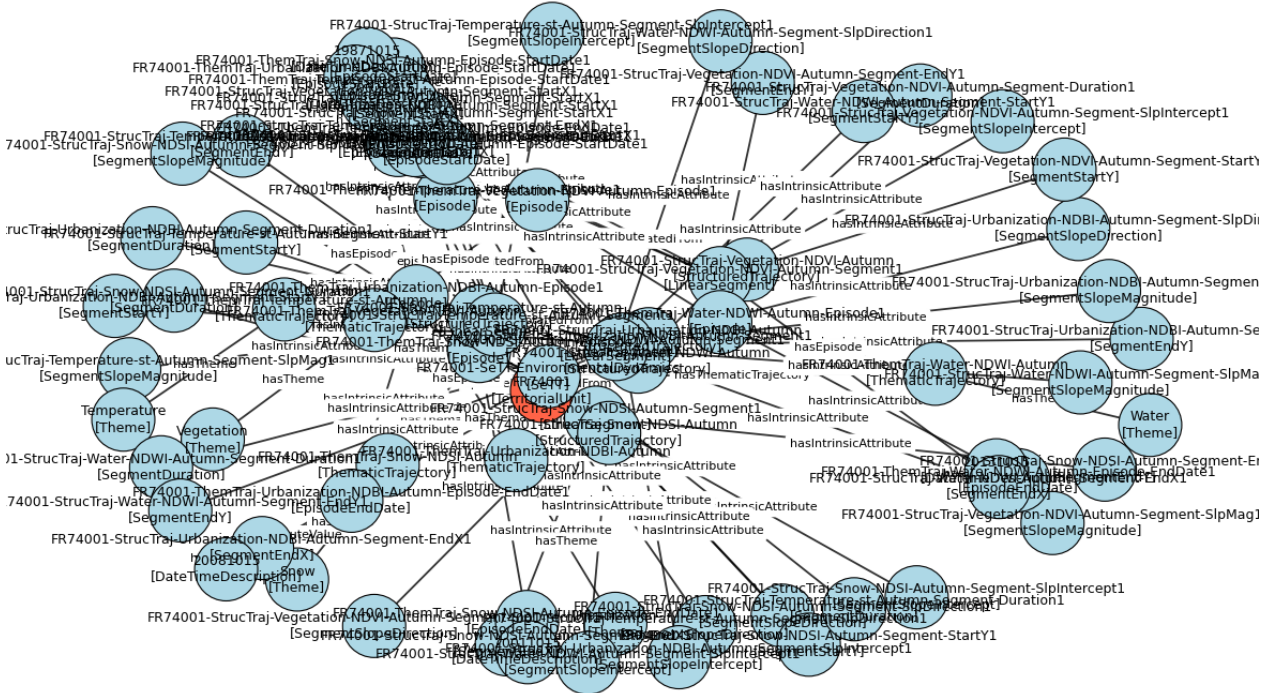


Figure 17 episode-snapshot including all themes for territory FR74001

To illustrate *territory-episode KG snapshot*, Figure 17 shows an episode-snapshot for territory (municipality) FR74001 combining all its thematic-snapshots together. It is clearly noticeable the high connectivity of relations, and the richness of information through attribute values and their measurements including the event node.

Based on the previous contribution on KG representation through episode-snapshots for each territory trajectory, the model treats each territory trajectory independently. For each territory we collect all its episode-snapshots, and these episode-snapshots serve as the input for the model. The purpose of this representation is to allow the model to learn node embeddings h_{node} from each territory-episode snapshot $G_{y_i}^{e_j}$, such that we can retrieve from each snapshot the embedding of interest for us, which is here the territory embedding $h_{y_i}^{e_j}$. After collecting the territory embedding $h_{y_i}^{e_j}$ from each territory-episode snapshot $G_{y_i}^{e_j}$, the model should learn from

these embeddings the whole territory trajectory, as each territory snapshot embedding should hold the whole information of the territory at that episode. This trajectory embedding can then be used for downstream tasks.

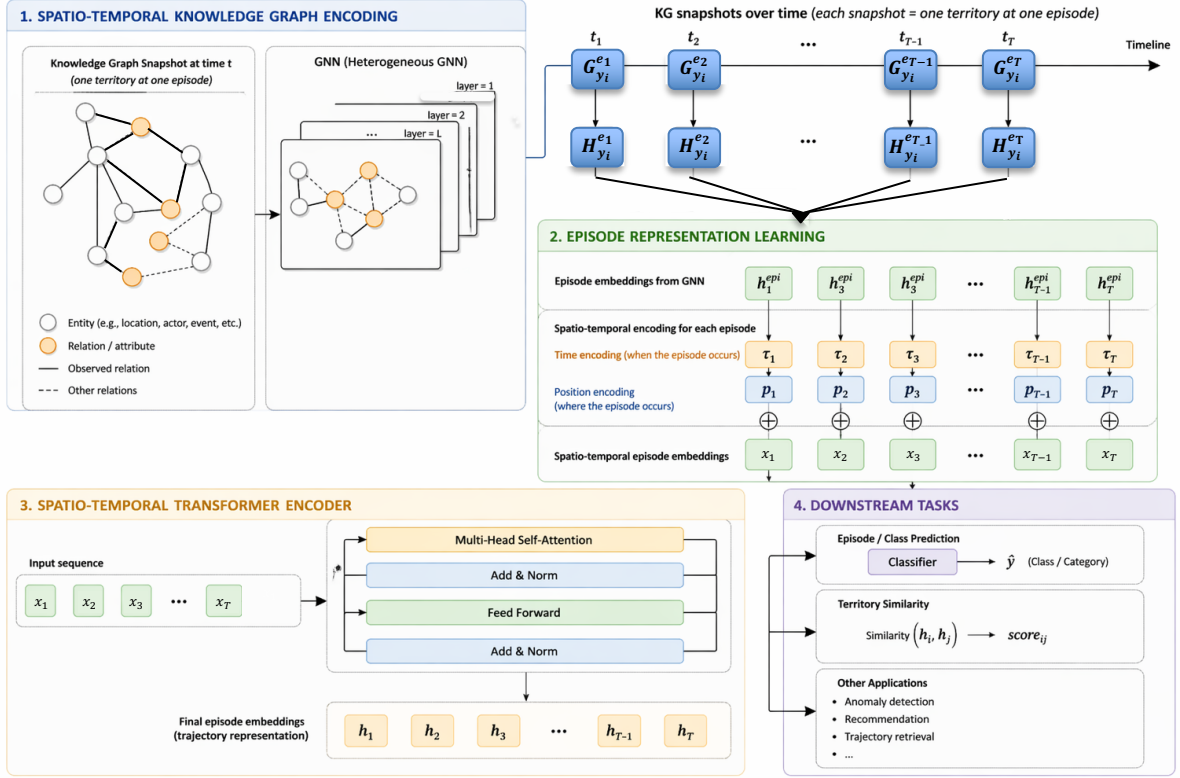


Figure 18 Full pipeline of the proposed SETT-GNN model

3.3 General Architecture of the Proposed SETT-GNN Model

Spatiotemporal Graph Neural Network for SETT Knowledge Graphs (SETT-GNN), is a hybrid model inspired by THAM architecture, made to represent Semantic Environmental Trajectories of Territories (SETT) through several steps of construction. Its objective is to learn a vector representation for each territory trajectory by combining the semantic structure of its episode snapshots with their temporal evolution.

As shown in Figure 18, the SETT-GNN model is organized into four main steps. First, for each territory, the corresponding episode Knowledge Graph snapshots are collected. Second, each snapshot is passed through an episode encoder to learn a territory-episode embedding. Third, temporal and positional information is injected into each episode embedding to represent both the duration of the episode and its order in the trajectory. Finally, the sequence of territory-episode embeddings related to one territory is passed to a Transformer encoder, which learns the global trajectory representation of the territory.

The model is applied independently to each territory trajectory. This means that, for a given territory, all its episode snapshots are processed as an ordered sequence. The final output is a trajectory-level embedding that summarizes the semantic and temporal evolution of that territory. In the following subsections, each component of the architecture is described in detail.

3.3.1 Episode Encoder: Relational GCN for SETT KG episode-snapshots

After creating the episode-snapshots $G^{e_j}_{y_i}$ from the SETT KG for territory y_i , in each of its episodes e_j . Each episode-snapshot is passed to a relational-GCN (R-GCN) to learn node embeddings. Since SETT KG snapshots contain multiple relation types, a standard GCN is not sufficient. Therefore, the model uses a R-GCN to distinguish between relation types during message passing. For a node i , the R-GCN update can be written as discussed in section 2.4.4:

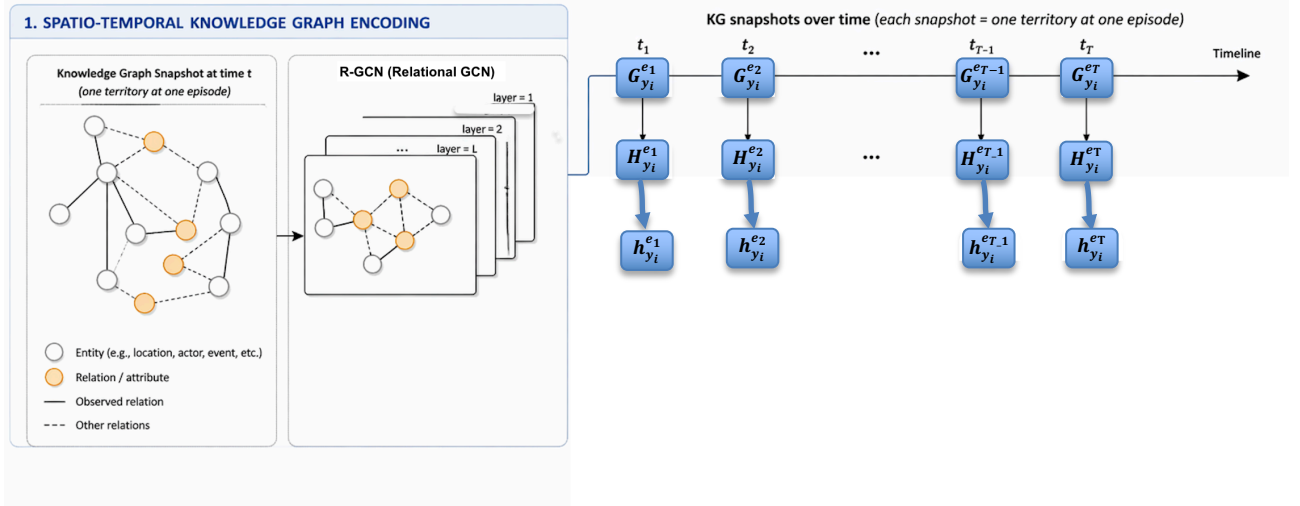


Figure 19 Spatiotemporal Knowledge Graph snapshots Encoding

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

As shown in Figure 19, after the R-GCN layers are applied to the episode snapshot $G_{y_i}^{e_j}$, the model obtains an embedding for each node in the snapshot represented in $H_{y_i}^{e_j}$. Since the territory node is the central node of the snapshot, the territory embedding $h_{y_i}^{e_j}$ is selected as the representation that describes the whole episode in its trajectory. This embedding is expected to contain the local semantic information of the territory during episode e_j , as it aggregates information from its relational neighborhood through the R-GCN message-passing process. The resulting territory-episode embeddings $\{h_{y_i}^{e_1}, h_{y_i}^{e_2}, \dots, h_{y_i}^{e_T}\}$ are then stored together with their corresponding temporal intervals. Finally, the embeddings obtained from successive episodes are arranged as an ordered sequence and passed to the temporal encoder in the next stage of the model.

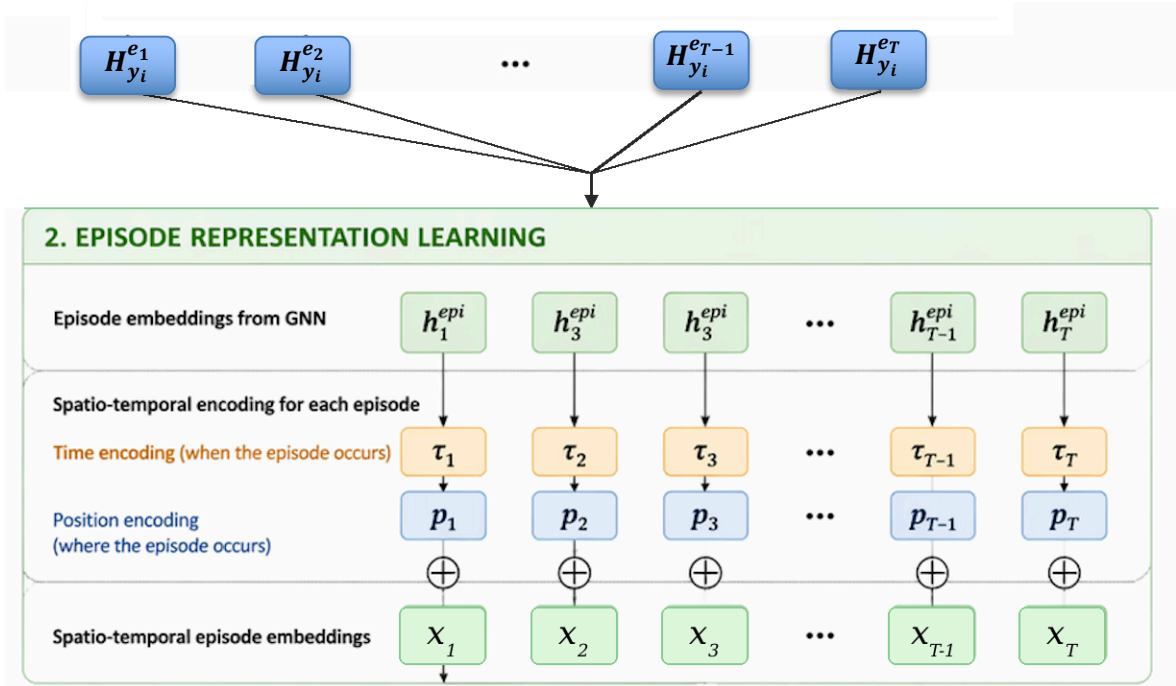


Figure 20 Temporal and Positional Encoders

3.3.2 Temporal and Positional Encoding

After the R-GCN is applied on each territory episode-snapshot $G_{y_i}^{e_j}$, each snapshot is represented by a territory-episode embedding $h_{y_i}^{e_j}$ and its corresponding episode time interval Δ_{e_j} beside it. However, this embedding only carries local semantic structure of the episode. It does not yet hold information about when the episode occurred, how far the episode is from the previous one, or where it appears in the trajectory sequence. Therefore, the model adds two complementary types of temporal information:

1. **Time encoding**, which represents the real temporal gap between episodes
2. **Positional encoding**, which represents the order of episodes in the sequence

As shown in Figure 20, each territory-episode embedding $h_{y_i}^{e_j}$ is passed through both the temporal and positional encoders, resulting with the new temporal-aware representation for each territory-episode embedding denoted as x_j . Since we are dealing with each territory trajectory independently, embedding representations from this step and forward refer to territory y_i and the episode index j will be indicated in the embedding index such that the territory-episode embedding for territory y_i at episode e_j after adding temporal and positional features is x_j .

3.3.2.1 Time encoding: modelling irregular temporal gaps

The time encoder receives each territory-episode embedding $h_{y_i}^{e_j}$ with its duration denoted as $\Delta_{e_j} = \text{largest temporal interval covering all themes in episode } e_j$. The objective is to produce a time-aware episode embedding v_{e_j} that combines both semantic information and temporal interval information.

To impose better stability for territory embeddings when injecting temporal features in it, we designed a gated continuous-time layer, adapted from THAM architecture [24], where for each episode we calculate both:

- **Raw scaled time** $\frac{\Delta_{e_j}}{s}$
- **Log-scaled time** $\frac{\log(1+\Delta_{e_j})}{\log(1+s)}$

where s is a time scaling constant passed from arguments.

The two-time features are then concatenated in q_{e_j} and passed through an MLP, such that we obtain the time feature vector u_{e_j} as:

$$q_e = \left[\frac{\Delta_e}{s} \parallel \frac{\log(1 + \Delta_e)}{\log(1 + s)} \right]$$

$$u_e = \text{MLP}(q_e)$$

where:

- q_{e_j} is the two-dimensional time feature vector,
- u_{e_j} is the learned continuous-time representation,
- $\parallel$ denotes concatenation.

This MLP allows the model to learn a non-linear transformation of the time interval instead of using a fixed decay function.

After obtaining the learned time representation u_{e_j} , the model computes a feature-wise gate:

$$g_{e_j} = \sigma(W_g u_{e_j})$$

and a candidate temporal contribution:

$$f_{e_j} = W_f u_{e_j}$$

where:

- W_g is the weight matrix used to compute the gate.
- W_f is the weight matrix used to compute the contribution of temporal feature.
- $\sigma(\cdot)$ is sigmoid

The final gated temporal contribution is then computed as:

$$\widetilde{f}_{e_j} = g_{e_j} \odot f_{e_j}$$

where $\odot$ denotes element-wise multiplication. The time-aware episode embedding is finally obtained by adding this gated temporal contribution to the original territory-episode embedding:

$$v_{e_j} = h_{y_i}^{e_j} + \widetilde{f}_{e_j}$$

The feature-wise gate allows the model to decide, for each territory-episode embedding dimension, how much the temporal interval should modify it. This is useful because not all learned features are equally time sensitive. Some dimensions may encode relatively stable semantic information about the territory, while others may encode dynamic patterns that should be more strongly influenced by the duration between episodes. In simple terms, the time gate acts like a set of small switches, one for each feature. For some features, the model may decide that the time interval is important and should modify them strongly. For other features, the model may decide that the time interval is less relevant and should have little effect.

3.3.2.2 Positional encoding: modelling order

In addition to continuous time encoding, the model also uses sinusoidal positional encoding to represent the order of episodes inside the trajectory. For an episode index e_j , the positional encoding is defined as:

$$PE(e_j, 2i) = \sin\left(\frac{e_j}{10000^{2i/d}}\right)$$

$$PE(e_j, 2i + 1) = \cos\left(\frac{e_j}{10000^{2i/d}}\right)$$

where d is the embedding dimension and i is the dimension index. The positional vector is then added to the time-aware episode embedding:

$$x_{e_j} = v_{e_j} + PE(e_j)$$

The resulting sequence:

$$X = [x_{e_1}, x_{e_2}, \dots, x_{e_T}]$$

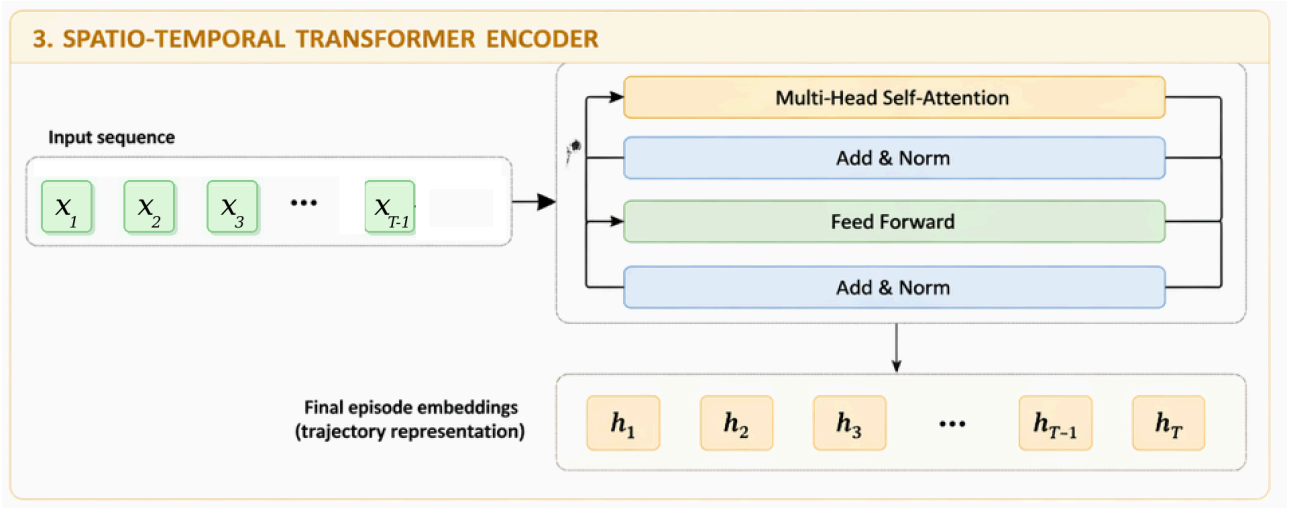


Figure 21 Transformer encoder for territory trajectory

contains structural information from the R-GCN, temporal gap information from the time encoder, and order information from the positional encoder. For each territory-episode embedding $h_{y_i}^{ej}$ we add temporal and positional features to obtain x_{e_j} . The sequence is then used as the input of the Transformer-based temporal encoder.

3.3.3 Transformer-Based Temporal Encoder

After the model obtains the sequence $X = [x_{e_1}, x_{e_2}, \dots, x_{e_T}]$ for territory y_i , it feeds the sequence to a Transformer encoder:

$$[h_1, h_2, \dots, h_T] = \text{Transformer}(X)$$

where each h_t is a contextualized episode embedding. Unlike the original R-GCN episode embeddings, which mainly encode the local semantic structure of individual episode snapshots, the Transformer output embeddings encode each episode in relation to the other episodes of the same territory trajectory.

Recently the use of Transformers has been extended to cover not only natural language processing, but also other fields as images, graphs, time-series, multimodal representations, and many other types of sequential data. This makes the Transformer architecture suitable for the proposed SETT representation, where a territory trajectory is represented as an ordered sequence of episode-snapshot embeddings.

Unlike recurrent models that mainly process the sequence step by step, the architecture of the transformer through self-attention mechanism, allows the model to directly compare each episode embedding with the other episode embeddings in the trajectory. Therefore, the model can capture long-range dependencies between distant episodes, rather than relying only on the immediately previous one in recurrent models. This is particularly useful for SETT trajectories, where the current state of a territory may depend on several previous episodes and not only on the last observed snapshot.

As shown in Figure 21, the elements of the input sequence X are treated as tokens, where each token corresponds to a time and position aware territory-episode embedding. Through the Transformer encoder, each token is updated by attending to the previous episode embeddings in the sequence. The final hidden representation associated with the last episode, h_T , is then selected as the output trajectory embedding of the model:

$$z_{y_i} = h_T$$

This embedding summarizes the evolution of territory y_i across its sequence of episode snapshots, since the input sequence already contains structural information from the R-GCN, temporal-span information from the

time encoder, and order information from the positional encoder, the final trajectory embedding z_{y_i} can be used for downstream tasks such as trajectory classification, node classification, node similarity, next-episode prediction, or future relation prediction.

3.4 Output Representation and Downstream Tasks

The SETT-GNN pipeline is applied independently to each territory. For each territory y_i , all corresponding episode snapshots are retrieved and ordered chronologically:

$$G_{y_i} = \{G_{y_i}^{e_1}, G_{y_i}^{e_2}, \dots, G_{y_i}^{e_T}\}$$

The model first encodes each episode snapshot using the R-GCN episode encoder, then enriches the resulting episode embeddings with temporal and positional information and finally processes the sequence using the Transformer-based temporal encoder. The final output of the model is the trajectory embedding of territory y_i :

$$z_{y_i} = h_T$$

where h_T is the final contextualized embedding produced by the Transformer. This embedding summarizes the semantic and temporal evolution of the territory across its episode sequence.

This trajectory embedding can be used for several downstream tasks.

3.4.1 Trajectory classification

The first possible task is trajectory classification, where the goal is to predict the class or type of the whole territory trajectory. In this case, the final trajectory embedding is passed to a multilayer perceptron:

$$\widehat{y}_{y_i} = MLP(z_{y_i})$$

where $\widehat{y}_{y_i}$ is the predicted class distribution or label of territory y_i .

3.4.2 Trajectory similarity

The second task is trajectory similarity. In this case, trajectory embeddings of different territories are compared in the learned embedding space. If two territories have similar semantic and temporal evolution, their embeddings are expected to be close to each other.

In this work, similarity can be computed using the L1 distance between two trajectory embeddings:

$$d(z_{y_i}, z_{y_j}) = |z_{y_i} - z_{y_j}|_1$$

where z_{y_i} and z_{y_j} are the trajectory embeddings of two territories. A smaller distance indicates a higher similarity between the two territory trajectories.

This formulation also allows comparing trajectory embeddings across different periods or thematic contexts. For example, embeddings learned from spring trajectories can be compared with embeddings learned from autumn trajectories using pairwise L1 distance. The closest embedding is then considered the most similar trajectory candidate.

3.4.3 Clustering

The third task is clustering. This task is important because it allows territory trajectories to be grouped according to their learned representations. After the model learns a final embedding for each territory

trajectory, clustering algorithms such as k-means can be applied to these embeddings to identify groups of territories with similar spatiotemporal and semantic behaviors.

In this context, territories that are placed in the same cluster are expected to share similar evolution patterns across their thematic trajectories, such as similar vegetation changes, snow evolution, urbanization behavior, or water-related dynamics. Therefore, clustering can help discover hidden groups of territories without requiring predefined labels.

To visualize the clustering results, dimensionality reduction methods such as PCA, t-SNE, or UMAP [9] can be applied to project the high-dimensional trajectory embeddings into a two-dimensional space. This makes it possible to observe whether territories with similar learned representations are grouped together and to interpret the structure learned by the model.

3.5 Positioning to the State of the Art

The proposed SETT-GNN approach is not designed independently from the state of the art but rather builds on the strengths and limitations identified in the synthesis of Section 2.5. The objective of this section is to position the proposed model with respect to the studied GNN-based approaches and to explain how these models motivated the main design choices of SETT-GNN.

Static GNN models such as GCN and GAT provide the basic principle of neighborhood-based representation learning. However, they are not sufficient for SETT Knowledge Graphs, because SETT snapshots are heterogeneous and contain multiple semantic relation types. Moreover, these models do not directly model temporal evolution between successive episodes. This motivates the use of a relation-aware encoder for each episode snapshot, followed by a temporal model able to capture the evolution of the full territory trajectory.

R-GCN is therefore used as the local graph encoder in the proposed architecture. Compared with a standard GCN, R-GCN is more suitable for SETT because it performs relation-aware message passing. This is important because an episode snapshot may contain several types of semantic relations connecting territories, events, segments, themes, measurements, and temporal or spatial entities. However, R-GCN alone remains a static graph encoder: it can represent the semantic structure of one episode, but it cannot model how the territory evolves from one episode to another.

Temporal KG models such as RE-GCN show the importance of representing graph evolution through a sequence of snapshots. This idea is relevant to SETT, since each territory can also be represented as a sequence of episode-level KG snapshots. Nevertheless, the objective of SETT-GNN is not mainly temporal link prediction, as in RE-GCN, but rather the construction of a trajectory-level representation for each territory. For this reason, the proposed model uses a Transformer encoder to process the sequence of episode embeddings and capture long-range dependencies between distant episodes.

The proposed model is also inspired by THAM, which separates local heterogeneous graph learning from temporal sequence modeling. In THAM, each medical visit is first represented using heterogeneous graph learning, and the sequence of visits is then processed by a time-aware Transformer. SETT-GNN adapts this principle to the SETT context: medical visits are replaced by territory episodes, medical heterogeneous graphs are replaced by SETT episode KG snapshots, and the final patient representation is replaced by a territory trajectory representation.

Therefore, SETT-GNN can be understood as a hybrid architecture combining ideas from several families of models. From GCN-based models, it inherits neighborhood aggregation. From R-GCN, it adopts relation-aware message passing for multi-relational Knowledge Graphs. From temporal KG models such as RE-GCN, it adopts the idea of modeling evolution through graph snapshots. From THAM, it adopts the separation between local graph encoding and temporal sequence encoding. The resulting architecture is adapted to the specific requirements of SETT KGs: preserving the semantic structure of each episode while learning the temporal evolution of a full territory trajectory.

4 Practical Implementation and Experimental Evaluation

In this chapter, we present the practical implementation of the SETT-GNN pipeline, and the experimental evaluation performed on it. After introducing the model from a theoretical point of view in [Chapter 3](#), this chapter focuses on the implementation of the proposed architecture and its application to the SETT dataset. The evaluation focuses on the quality of the learned representations through downstream tasks such as trajectory similarity, trajectory classification, and clustering. The source code developed for this work is provided in the associated project repository (<https://github.com/Al339275/Spatio-Temporal-GNN>).

4.1 Implementation Environment and Tools

Starting with the practical implementation, we introduce the technical environment used to implement the full pipeline. The model is completely implemented in Python [25] which was used for data preprocessing, Knowledge Graph manipulation, model construction, training, and evaluation. Building the graph neural network, time and positional encoders, and transformers has been achieved using the main Python libraries for Deep Learning and graph representation learning, namely PyTorch [26], PyTorch Geometric[27], and Numpy[28].

For Knowledge Graph processing and snapshot generation, several Python libraries have been used. Since SETT KGs are in RDF data, they are parsed and manipulated using rdflib [29], while NetworkX [30] was used for territory-episode snapshot construction, inspection and analysis. To organize the extracted features as time interval, territory attributes, episode index, and territory local index, with each corresponding territory-episode snapshot, we used collections and typing libraries where we can use dictionaries, lists, sets, and other data containers.

The implementation has been executed on a local machine. The code was designed to run on CPU, while GPU acceleration can be used when a CUDA-compatible GPU is available, allowing the same pipeline to be executed in different hardware environments.

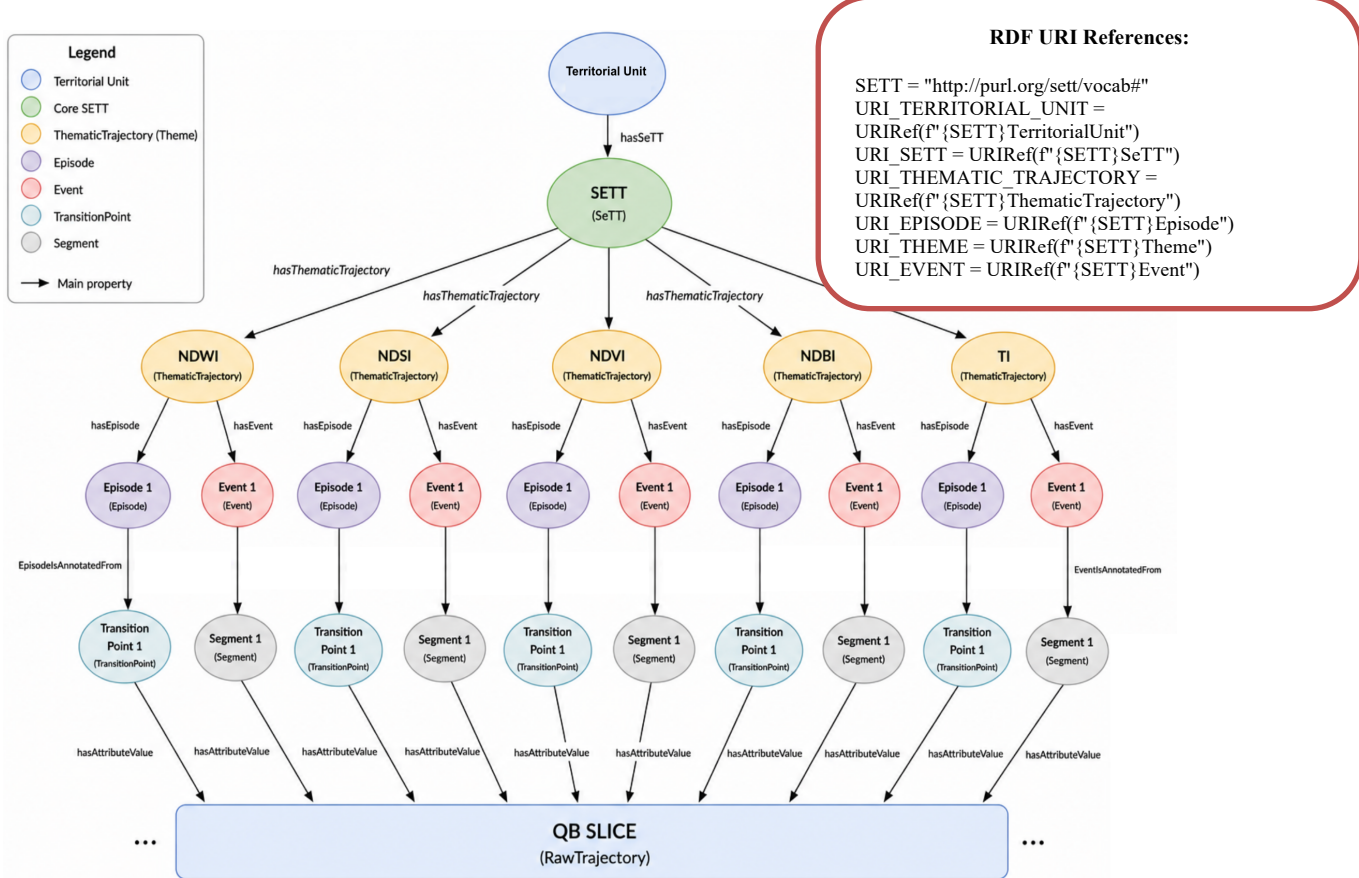


Figure 22 Simplified example of a territory-episode snapshot

4.2 SETT Snapshot Preprocessing

Before applying the SETT-GNN model, we need to structure our SETT knowledge graph into the input representation discussed previously in order to apply it for graph representation learning. The preprocessing step is implemented in *territory_episode_snapshots* module. The main objective of this implementation is to parse turtle RDF files, and extract the ontology-guided trajectory structure, to construct territory-episode snapshots.

The first step in preprocessing is to define the main classes and properties of the SETT ontology that a snapshot should contain. Figure 22 shows an example for the KG structure of the territory-episode snapshot, building it with the main classes defined in the SETT ontology that construct the trajectory are `TerritorialUnit`, `SeTT`, `ThematicTrajectory`, `Episode`, `Theme`, `Event`, `Segment`, `TransitionPoint`, `StructuredTrajectory`, `RawTrajectory`, and `Slice`. While for properties the main ones are `hasSeTT`, `hasThematicTrajectory`, `hasEpisode`, `hasTheme`, `hasEvent`, `hasSegment`, `hasIntrinsicAttribute`, `hasTransitionPoint`, `episodeIsAnnotatedFrom`, `eventIsAnnotatedFrom`, `segmentIsAnnotatedFrom`, and `hasAttributeValue`. These ontology elements are defined in the code in a RDF URI reference as indicated in Figure 22, to be used in extraction process.

The next step consists in loading the RDF SETT KG and parsing it into a graph object using `rdflib`. After parsing the KG, we create two dictionaries to save the incoming and outgoing triples based on the subject and object entities, such that incoming triples are saved with the object entity as the key and outgoing triples are saved with subject entity as the key, for this helps in the snapshot construction. Then, the territorial units are extracted by selecting the subject of all triples whose type is the class `TerritorialUnit`. These territories represent the main entities for which semantic trajectories are constructed.

In addition to the loaded SETT KG, the model also loads two metadata files. The first file contains static features for each territory represented in RDF turtle form. This file contains characteristics of each territory such as `area`, `name`, `elevation` and `slopdegrees`. The second file contains the Corine Land Cover (CLC) [<https://land.copernicus.eu/en/products/corine-land-cover>] indicator measurements organized over five main CLC areas: artificial surfaces, agricultural areas, forest and semi natural areas, wetlands, and water bodies. These metadata are also stored for each territory to be used later as additional features.

After loading the SETT KG and metadata files, the processing follows the SETT ontology structure to extract the episode snapshot nodes for each territory shown in Figure 22. For each territorial unit, the code retrieves the `:SeTT` node which is directly connected to territory node (we recall Figure 3 for SETT ontology), then the `:ThematicTrajectory` node attached to the SETT node defining the theme of this trajectory. For each thematic trajectory, the code retrieves its `:Theme`, `:Episode`, `:Event`, and `:Segment` nodes. After retrieving all these nodes for each territory, they are saved in a tuple such that each tuple represents a territory, one theme, and the episode of the theme.

The time interval for each episode is retrieved by searching the nodes for each episode with type `EpisodeStartDate` and `EpisodeEndDate`, then their values are retrieved through the `hasAttributeValue` property. These start and end values are converted into numerical temporal features composed of the start date, the end date, and the episode duration in days. These temporal features are stored with the snapshot and later used by the temporal part of the model.

Then for each tuple (*territoryURI*, *settURI*, *themeURI*, *episodeURI*) the *collect_snapshot_nodes* function is called, that searches for each tuple all its outgoing and incoming nodes, using the incoming and outgoing lists created previously. An extra expansion has been achieved on the local nodes for the episode, since the episode node is attached to its segment, event, slope value, slop direction, and measurements for the segment. All the retrieved nodes for the local (*territory*, *theme*, *episode*) snapshot are saved in the *selected_nodes* set.

Once the relevant nodes are selected, the function *build_snapshot_graph* constructs the graph using the `NetworkX` library. URI nodes are stored with their URI, label, RDF types, and node type, while literal nodes are also included as graph nodes with their value and datatype. The edges of the snapshot correspond to the

RDF triples whose subject and object both belong to the selected node set and whose predicate is one of the allowed ontology properties. Each edge stores the predicate URI and a predicate label indicating its relation type. The resulting snapshot is shown in [Figure 17](#) for the territory FR74001 and illustrated in a simplified form in Figure 22 to clarify the idea.

The first output of the preprocessing is a set of theme-specific snapshots. Each record corresponds to one territory, one thematic trajectory, and one episode. The record stores the NetworkX graph, the territory identifier, the thematic trajectory identifier, the episode identifier, the theme identifier, the time features, the episode start and end dates, and the territory metadata. These snapshots are organized in a nested container:

$$\text{snapshots}[\text{territory}][\text{trajectory}][\text{episode}]$$

This representation makes it possible to inspect each thematic trajectory independently.

However, the input required by the SETT-GNN model is not a separate graph for each thematic trajectory. Instead, the model needs one snapshot per territory and episode step. Therefore, the preprocessing also builds aggregated episode-level snapshots. For each territory and episode index, all theme-specific snapshots corresponding to that same episode index are merged into a single graph. This produces a territory-episode snapshot that combines the information coming from all thematic trajectories at the same episode step.

The final representation is therefore organized as:

$$G_{y_i} = \{G_{y_i}^{e_1}, G_{y_i}^{e_2}, \dots, G_{y_i}^{e_T}\}$$

where G_{y_i} is the sequence of episode snapshots for territory y_i , and $G_{y_i}^{e_j}$ is the merged Knowledge Graph snapshot of territory y_i at episode step e_j . Each snapshot is represented as:

$$G_{y_i}^{e_j} = (V_{e_j}, R_{e_j}, E_{e_j})$$

where V_{e_j} is the set of nodes, R_{e_j} is the set of relation types, and E_{e_j} is the set of RDF edges preserved in the local snapshot.

During the merging step, the start and end dates of the theme-specific episodes are also combined. The preprocessing takes the minimum start date and the maximum end date among the merged theme-specific snapshots. This creates a boundary time interval covering all thematic components of the same territory-episode snapshot.

Finally, the generated snapshots are sorted according to their episode indices. This ensures that each territory is represented as an ordered sequence of episode snapshots before being passed to the temporal model. The output of this preprocessing step includes the theme-specific snapshots, the merged territory-episode snapshots, the corresponding metadata, and the ordered snapshot sequences used by the SETT-GNN pipeline.

4.3 Input Representation for SETT-GNN

After the preprocessing step, each territory is represented as sequence of ordered episode KG snapshots stored in NetworkX graphs. Before passing it to the RGCN model, we need to convert each snapshot into a numerical representation compatible with PyTorch Geometric. This conversion is implemented in the *graph_to_pyg_featureAware* module.

The first step of the conversion is to create a stable mapping for nodes from graph representation to vector representation. For each snapshot, the nodes are mapped following the one-hot encoding, based on the node type defined in the ontology. The dimension of the node vectors depends on a global node dimension such that all nodes in all snapshots have the same dimension. Similarly, each edge is mapped to a specific identifier

based on its type, according to a global edge-type mapping ensuring that the same RDF edge/relation will always have the same relation identifier through all snapshots.

The next step after mapping nodes is to create the node feature matrix X , this construction is made of several steps. First, each node will receive a one-hot encoding of its RDF type. For example, a node typed as `Episode`, `Segment`, `Event`, or `TerritorialUnit` receives different vectors according to these ontology classes. This allows the model to distinguish between the semantic role of each node in the snapshot.

Second, the implementation adds value-aware features extracted from `hasAttributeValue` relations. Since attribute nodes and literal nodes may contain numerical, and string values. The code therefore checks for each node whether it has such values and encodes these values in the node feature vector. String values are converted using a deterministic hash-based function, that encode each letter with a numerical value resulting in a fixed-size numerical representation for each string. For example, in segment node feature vector, we can now in the embeddings distinguish whether the segment corresponds to an increasing or decreasing slope since each string will have different numerical representation.

Third, an additional event features are also extracted for event nodes. This is based on the SETT ontology where each event is connected to a `TransitionPoint` node which has several attributes such as date, and numerical measure corresponding to each theme. We have created five-values feature vector indicating if the event has *branch* values, if it has *numeric* values, if it has *time* value, if it has *string* value, and the *mean* of its numeric values. This way, we add extra unique features defining the event in each territory-episode snapshot.

Fourth, the static features for each territory are injected to the territory node feature vector only. These features include numerical descriptors extracted during preprocessing, such as *area*, *elevation*, and slope-related attributes. Since these values may have different scales, the code computes dataset-level normalization on each measurement. This ensures that large numerical values do not dominate the node feature representation. In addition to these static territory features, CLC percentage indicators grouped in the five land-cover classes are normalized and injected into the territory node feature vector, resulting in a territory node containing both its semantic graph context and additional spatial/environmental descriptors.

Considering edges, they are represented in two forms respecting the required form for R-GCN input. For this, we consider two data structure for edges. First, an *edge_index* tensor stores the source and target node indices of each edge in the PyTorch Geometric format. A second tensor *edge_attr* stores with an integer identifier the relation type corresponding to each edge. These edge type identifiers are essential for R-GCN since they are the source that allow the model to distinguish between different edge types while learning representations.

To allow the message information to propagate in both directions, for each edge (s, r, o) we add its inverse (o, r, s) in the *edge_index*. This way, the territory node can receive information from its surrounding semantic context and can also pass information back to connected nodes.

Each converted snapshot is stored as a PyTorch Geometric Data object containing the following main elements:

$$Data(X, edge_index, edge_attr)$$

where X is the node feature matrix, *edge_index* is the graph connectivity tensor, and *edge_attr* stores the relation type of each edge. Additional metadata are also stored in the object, including the local index of the territory node, the territory identifier, the episode index, the episode start and end dates, and the time features.

The local territory index in each snapshot is particularly important as in the SETT-GNN model after learning node representation from territory episode snapshot, the model retrieves the embedding of territory only as the representation of the whole episode. Therefore, after applying R-GCN encoder to the territory-episode snapshot, the model retrieves only local territory embedding $h_{y_i}^{e_j}$.

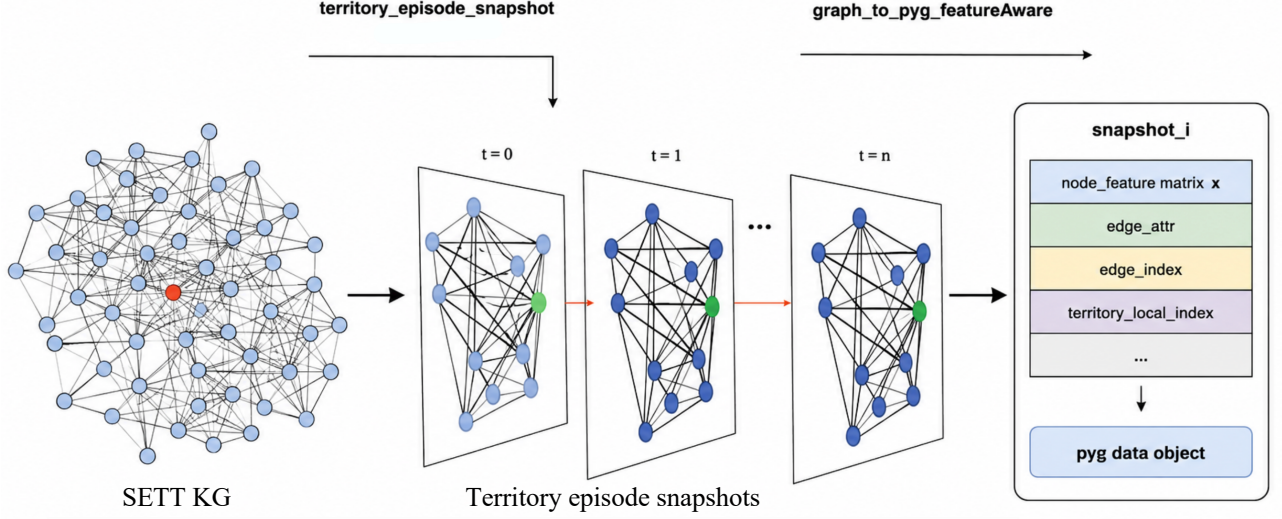


Figure 23 Illustrating SETT KG processing and snapshot representation in PyTorch

Finally, the converted PyTorch Geometric snapshots are organized according to the trajectory structure defined in the preprocessing step. Each territory is represented as a sequence of episode snapshots:

$$G_{y_i} = \{G_{y_i}^{e_1}, G_{y_i}^{e_2}, \dots, G_{y_i}^{e_T}\}$$

Figure 23 show how these 2 steps are related and where each module falls, such that *territory_episode_snapshot* module responsible for the transformation of the SETT KG combining all territories together into snapshots based on each territory and its episodes. Then, *graph_to_pyg_featureAware* module is responsible to transform each snapshot into a PyTorch Geometric Data object saving the input objects for R-GCN.

Each snapshot is first encoded independently by the R-GCN episode encoder. The resulting sequence of territory-episode embeddings is then passed to the temporal part of SETT-GNN, where temporal and positional encodings are added before applying the Transformer encoder.

4.4 Model Implementation

After converting the SETT episode snapshots into PyTorch Geometric Data objects, the next step is implementing the neural network pipeline of the proposed SETT-GNN model. As introduced in Chapter 3, the model follows a two-level representation learning. First level is based on learning territory-episode embedding from its corresponding territory-episode snapshot through R-GCN encoder. Then, the sequence of territory-episode embeddings are processed through a time-aware Transformer to learn the evolution of the full territory trajectory.

The implementation therefore separates the model into several components. The first component is the R-GCN episode encoder, its role is to encode the internal heterogeneous structure of each SETT snapshot by considering both the node features and the relation types between nodes. The second component is the time encoder followed by a positional encoder, injecting temporal features to territory embeddings. Finally, a Transformer encoder is used to learn from the episode representations a final trajectory-level embedding for each territory.

The following subsections describe the implementation of these components, starting with the R-GCN episode encoder.

4.4.1 Episode Encoder Implementation

The episode encoder is implemented using an R-GCN stack. The use of R-GCN is based on the study performed on GNN in [section 2.4](#) and motivated by the heterogeneous structure of each territory-episode snapshot, which is what R-GCN is best suitable for. In each territory-episode snapshot, edges may correspond to different semantic predicates, such as `hasEpisode`, `hasTheme`, `hasSegment`, `hasAttributeValue`, or `episodeIsAnnotatedFrom`. A standard GCN would treat all these edges in the same way, while R-GCN uses relation-specific message passing. This allows the model to distinguish the semantic role of each relation during aggregation.

In the implementation, the episode encoder is defined by the class `RGNNStack`. The model receives as input a territory-episode snapshot represented as a PyTorch Geometric Data object containing the node feature matrix X , the `edge_index` tensor, and the relation-type tensor `edge_attr`. Each R-GCN layer is implemented using `RGCNConv` from PyTorch Geometric. At each layer, the model uses `edge_index` to define the graph connectivity and `edge_attr` as the relation type input. Therefore, messages are aggregated according to both the graph structure and the type of relation connecting the nodes.

The R-GCN encoder can be summarized as:

$$H^{(l+1)} = \text{RGCNConv}(H^{(l)}, \text{edge_index}, \text{edge_type})$$

where $H^{(l)}$ is the node representation at layer l , and `edge_type` corresponds to the relation identifiers stored in `edge_attr`. After each R-GCN layer, a ReLU activation function and dropout are applied. The ReLU function introduces non-linearity, while dropout helps reduce overfitting.

The number of R-GCN layers, hidden dimension, output dimension, dropout rate, and number of relations are passed as model hyperparameters. The first R-GCN layer maps the augmented input features to the hidden dimension, while the following layers preserve the hidden size. After the message-passing layers, a small post-message-passing network is applied. This network is composed of a linear layer, a ReLU activation, dropout, and a final linear projection to obtain the desired output dimension.

The output of the episode encoder is a matrix of node embeddings:

$$H_{y_i}^{e_j} \in \mathbb{R}^{|V_{e_j}| \times d}$$

where $|V_{e_j}|$ is the number of nodes in the episode snapshot and d is the output embedding dimension. Since the goal is to represent the territory at this episode, the model extracts the embedding of the territory node using the stored `territory_local_index`. This embedding is considered the episode-level representation:

$$h_{y_i}^{e_j} = H_{y_i}^{e_j}[\text{territory_local_index}]$$

Thus, for each territory y_i and each episode e_j , the R-GCN encoder produces one embedding $h_{y_i}^{e_j}$. This embedding summarizes the local semantic graph structure of the territory during that episode and is later used as an element in the temporal sequence processed by the Transformer encoder.

4.4.2 Temporal and Positional Encoding Implementation

After encoding the territory episode snapshots with the R-GCN encoder and obtaining the territory-episode embedding $h_{y_i}^{e_j}$, then for each territory trajectory the model outputs a sequence of territory episode embeddings ordered as: $[h_{y_i}^{e_1}, h_{y_i}^{e_2}, \dots, h_{y_i}^{e_T}]$. At this stage, these embeddings encode only the local semantic structure of each episode, and to encode time and position in these embeddings we use both temporal and positional encoders before passing the sequence to the Transformer.

The temporal encoding is implemented using a learnable TimeEncoder. For each episode embedding $h_{y_i}^{e_j}$, the model uses the time interval Δ_t extracted in the preprocessing step. The function *prepare_delta_t* reshapes Δ_t so that it is the same dimension as the episode embedding tensor. Then, the implementation builds two scalar temporal features: the scaled time gap and logarithmic scaled time gap for better stability, as defined before their formulations are respectively:

$$\frac{\Delta_t}{s} \quad \text{and} \quad \frac{\log(1 + \Delta_t)}{\log(1 + s)}$$

These two values create a two-dimensional time feature vector. This time feature vector is passed through a small multi-layer perceptron composed of two linear layers with ReLU activations:

$$h_t = MLP\left(\left[\frac{\Delta_t}{s}, \frac{\log(1 + \Delta_t)}{\log(1 + s)}\right]\right)$$

The two temporal values are passed through an MLP to transform the scalar time information into a learnable temporal representation. The original time feature vector only contains two numerical values, while the episode embedding has a higher dimensional representation space. Therefore, the MLP allows the model to map the time interval into a hidden representation that can be used to generate a temporal contribution compatible with the episode embedding. This also makes the temporal encoding more flexible. Instead of using a fixed transformation of time, the model learns during training how the duration of an episode should influence the embedding. The MLP can capture non-linear effects of time, meaning that short and long episodes do not have to affect the representation in the same way. The output of the MLP is then used to compute the temporal gate and the temporal contribution added to the episode embedding.

The resulting hidden time representation h_t is then used to compute two vectors. The first vector is a gate:

$$g_t = \sigma(W_g h_t)$$

and the second vector is a candidate temporal contribution:

$$\tilde{f}_t = W_f h_t$$

where W_g and W_f are learnable linear projections, and σ is the sigmoid function. The gate controls how much temporal information should be added to each embedding dimension. The final temporal contribution is computed as:

$$f_t = g_t \odot \tilde{f}_t$$

where $\odot$ denotes element-wise multiplication. Finally, the time-aware episode embedding is obtained by adding this gated temporal contribution to the original episode embedding:

$$v_{e_j} = h_{y_i}^{e_j} + f_t$$

The TimeEncoder therefore returns both the time-aware embedding v_t and the temporal contribution f_t . This makes the implementation flexible, because v_t can be used as the input to the temporal encoder, while f_t can be inspected to understand the magnitude of the learned temporal effect.

The second module is the positional encoder, implemented in the class *PositionalEncodingFromEpisodeIndex*. After the temporal encoder representation of the real duration of an episode, the positional encoder represents the order of the episode inside the trajectory. This is necessary because the temporal encoder alone cannot inject the information about the order of the episode in the trajectory.

As discussed previously in Chapter 3, the positional encoder takes as input the episode index, then using the continuous sinusoidal positional encoding, the implementation converts this index into a continuous positional embedding with the same dimension as the episode. The positional encoding follows the standard Transformer formulation:

$$Pos(t, 2i) = \sin\left(t \cdot \exp\left(-\frac{\log(10000)}{D} \cdot 2i\right)\right)$$

$$Pos(t, 2i + 1) = \cos\left(t \cdot \exp\left(-\frac{\log(10000)}{D} \cdot 2i\right)\right)$$

where t is the episode index of episode e_j , i is the dimension index, and D is the embedding dimension. Even dimensions are filled using sine functions, while odd dimensions are filled using cosine functions.

The output of this module is a positional encoding tensor with shape $[T, D]$, where T is the number of territory-episode embeddings, and D is the embedding dimension. This positional vector is then added to the time-aware episode embedding:

$$v'_{e_j} = v_{e_j} + Pos_t$$

where v_{e_j} is the time-aware embedding produced by the TimeEncoder, and Pos_t is the positional encoding of episode t .

To summarize, the implementation distinguishes between two complementary notions of time. The time encoder uses the real temporal duration or time gap of the episode, while the positional encoder uses the episode order in the trajectory. By combining both, the model receives information about how long each episode lasts and where it appears in the sequence before the embeddings are passed to the Transformer encoder.

4.4.3 Transformer Encoder Implementation

After applying the R-GCN episode encoder and adding temporal and positional information, each territory is represented as a sequence of territory-episode time-aware embeddings defined as:

$$[v'_{e_1}, v'_{e_2}, \dots, v'_{e_T}]$$

Where $v'_{e_t} \in R^D$ represents the embedding of the territory y_i at episode e_j and D is the embedding dimension. The Transformer implementation supports both a single trajectory with shape $[T, D]$, indicating T territory-episode trajectory embeddings with D dimension, or a batch of trajectories with shape $[B, T, D]$, where B is the batch size.

The Transformer is implemented using `nn.TransformerEncoder`, a PyTorch module that stacks several Transformer encoder layers to process an input sequence with self-attention and produce contextualized representations for each episode. The number of layers is passed as a hyperparameter through `num_layers`. The Transformer encoder produces an output sequence:

$$O = [h_1, h_2, \dots, h_T]$$

where each h_t is a contextualized episode representation. This means that the representation of each episode is no longer based only on its own local graph embedding, but also on its relation to the other episodes in the same territory trajectory. Through self-attention, each episode can compare itself with the other episodes and learn long-range dependencies across the trajectory.

The output of the module has the same shape as the input episode sequence. If the input has shape [B,T,D], the output also has shape [B,T,D]. If the input was originally a single trajectory with shape [T,D], the temporary batch dimension is removed before returning the result.

This Transformer encoder therefore allows SETT-GNN to move from independent episode embeddings to contextualized trajectory representations, where each episode is interpreted in relation to the full temporal evolution of the territory.

4.4.4 End-to-End Model Integration

After implementing the individual components of the model, they are all combined in the *SpatioTemporalModel* class. The input of the model is a list of PyTorch Geometric Data objects, where each object represents a territory-episode snapshot. Each snapshot Data object contains the node feature matrix, the graph edges, the edge relation types, the local index of the territory node, the episode index, and the start and end dates of the episode.

The first step of the forward pass applies the R-GCN encoder to each territory-episode snapshot independently. This will result an embedding representation for all the nodes in the snapshot. Since the objective is to represent the territory at each episode, the model extracts the embedding of the territory node using the *local_territory_index*. This results with one territory-episode embedding per snapshot $h_{y_i}^{e_j}$, where y_i is the territory and e_j is the episode.

These embeddings are then enriched with temporal and positional information through temporal and positional encoders according to the episode time interval and index respectively. This results with an ordered time-aware episode embedding:

$$[v'_{e_1}, v'_{e_2}, \dots, v'_{e_T}]$$

The time-aware sequence is then passed to the Transformer encoder. The Transformer also receives the episode indices, which are used to add positional encodings before applying self-attention. This allows the model to consider both the real duration of each episode and its order inside the trajectory.

The Transformer outputs a contextualized sequence of episode embeddings, where each episode representation has been updated according to the other episodes of the same territory trajectory. In the current implementation, the final territory trajectory representation is obtained by taking the last Transformer output:

$$z_{y_i} = h_T$$

where h_T is the contextualized representation of the last episode. This vector is considered the final embedding of the full territory trajectory.

The model returns the final embeddings for all territories, together with their identifiers and the intermediate grouped episode sequences. These trajectory embeddings can then be used outside the model for downstream tasks such as trajectory classification, trajectory similarity, or clustering.

4.5 Evaluation Tasks and Metrics

The objective of the evaluation is to study the quality of the trajectory embedding produced by the SETT-GNN. After processing the sequence of episode snapshots for each territory, the model outputs a final trajectory embedding z_{y_i} . This embedding is expected to hold the semantic and temporal evolution of each territory. Therefore, the evaluation focuses on downstream tasks that use these embeddings for: trajectory classification, trajectory similarity, and clustering.

4.5.1 Trajectory Similarity

Trajectory similarity evaluates whether territories with similar evolution patterns are close in the learned embedding space. Given two territory embeddings z_{y_i} and z_{y_j} , their distance can be computed using measures such as L1 distance, L2 distance, or cosine similarity [22]. For example, the L1 distance is defined as:

$$d(y_i, y_j) = \|z_{y_i} - z_{y_j}\|_1$$

A small distance indicates that the two territories have similar learned representations, while a large distance indicates that their trajectories are less similar. This task can be used to retrieve territories that present similar semantic and spatiotemporal behaviors. If reference pairs of similar territories are available, ranking metrics such as *Hits@K* or *Mean Reciprocal Rank MRR* can also be used [22]. *Hits@K* measures whether the correct answer appears in the top K results returned by the model. While *MRR* measures how high the correct answer is ranked on average.

4.5.2 Clustering

Clustering evaluates the structure of the embedding space without requiring predefined labels. After obtaining the final trajectory embeddings for all territories, a clustering algorithm such as K-means can be applied to group territories with similar representations:

$$\{C_1, C_2, \dots, C_K\} = KMeans(\{z_{y_1}, z_{y_2}, \dots, z_{y_N}\})$$

If the learned embeddings are meaningful, territories with similar thematic and temporal evolution should be grouped in the same cluster. In addition, dimensionality reduction methods such as PCA, t-SNE, or UMAP [9] can be used to project the trajectory embeddings into two dimensions. This allows a visual inspection of whether the territories form meaningful groups according to their learned representations.

4.5.3 Trajectory Classification

Trajectory classification evaluates whether the learned trajectory embeddings carry enough information that allow to predict the class of each territory trajectory. In this setting, the trajectory embedding is passed to a classifier, such as a multilayer perceptron, in order to predict the target label:

$$\hat{y}_i = MLP(z_{y_i})$$

This task requires a ground-truth labels to be available. In our case, there was no such ground-truth for these datasets. If ground-truth labels are available, the performance can be measured using standard classification metrics. Accuracy measures the percentage of correctly classified trajectories. Precision measures how many predicted elements of a class are correct, while recall measures how many true elements of a class are correctly retrieved. The F1-score combines precision and recall into one metric.

4.6 Results and Ablation Studies

This section presents the experimental results obtained with the SETT-GNN pipeline. Since the main objective of the model is to learn meaningful territory embeddings, the evaluation focuses mainly on self-supervised trajectory similarity. The goal is to verify whether the model can learn embeddings that place related, or corresponding territory trajectories close to each other in the embedding space.

The experiments were mainly conducted on the Fribourg dataset. This dataset contains 127 territories (municipalities), where each territory is described through five thematic trajectories. Each thematic trajectory is composed of two episodes, resulting in a total of 254 territory-episode snapshots after preprocessing. The generated graph snapshots contain 11 relation types, which are used by the R-GCN encoder during relation-aware message passing. In addition to the graph structure, each territory is enriched with 11 static territorial features and 5 grouped CLC land-cover features. These characteristics make the Fribourg dataset suitable for testing whether the proposed pipeline can learn meaningful territory trajectory representations from both semantic graph structure and territory-level attributes.

Several configurations were tested to study the effect of adding territory-level static features, land cover features, and feature masking during training. The evaluation was performed using ranking metrics, namely Hits@1, Hits@5, and Mean Reciprocal Rank (MRR). Hits@1 measures whether the correct territory is ranked first, Hits@5 measures whether it appears among the top five retrieved candidates, and MRR evaluates the average ranking position of the correct answer.

4.6.1 Classification Results

Trajectory classification is one of the possible downstream tasks that could be applied on the learned territory embedding. In this task the final trajectory embedding z_{y_i} is passed to a classifier head to predict the class associated with the territory trajectory. However, in the current stage our dataset does not have territory classes or labels to do this supervised task.

Therefore, classification is considered as a possible downstream application of the learned embeddings, but it was not the main evaluation task in the current experiments. A complete classification evaluation would require reliable labels for territory trajectories, such as categories describing specific evolution patterns, territorial behaviors, or environmental classes. Once such labels are available, the embeddings produced by SETT-GNN can be evaluated using metrics such as accuracy, precision, recall, F1-score.

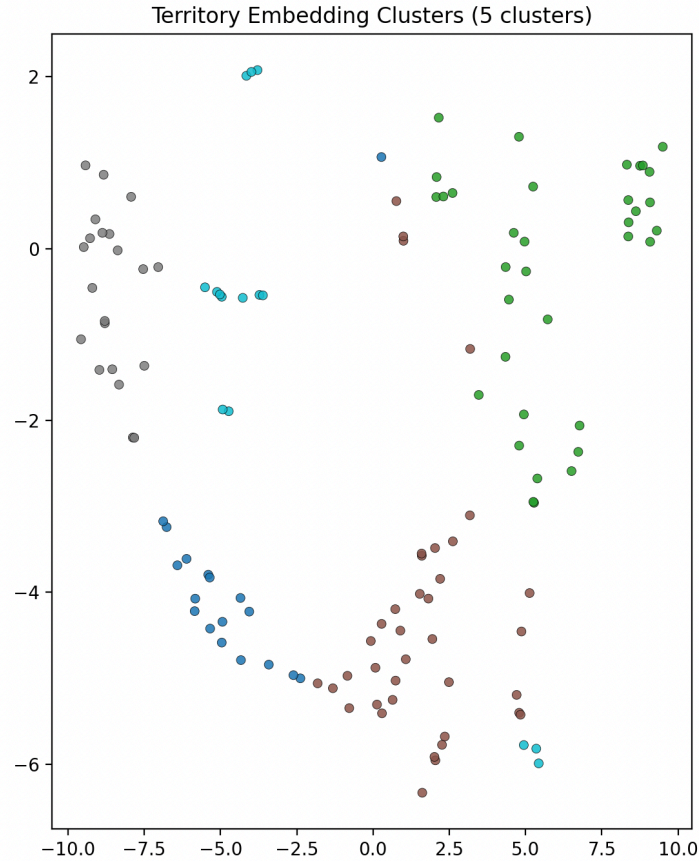


Figure 24 Fribourg dataset clustered over five groups

4.6.2 Clustering Results

Clustering was also defined as a possible downstream task for the trajectory embeddings. The objective of clustering is to group similar territory trajectory embeddings close to each other's, without using predefined labels.

In the current stage of the experiments, clustering is mainly considered as a qualitative evaluation direction. The final embeddings produced by SETT-GNN can be passed to clustering algorithms such as K-means. Then, dimensionality reduction methods such as PCA, t-SNE, or UMAP can be used to visualize the embeddings in two dimensions. This would allow checking whether territories form meaningful groups according to their region, land-cover profile, or trajectory behavior.

Figure 24 shows the clustering of five groups applied on Fribourg dataset composed of 127 territory trajectories encoded with static and CLC features. We can take from these clusters the territories that are classified in each group and launch a reversible process predicting the territory labels based on this clustering.

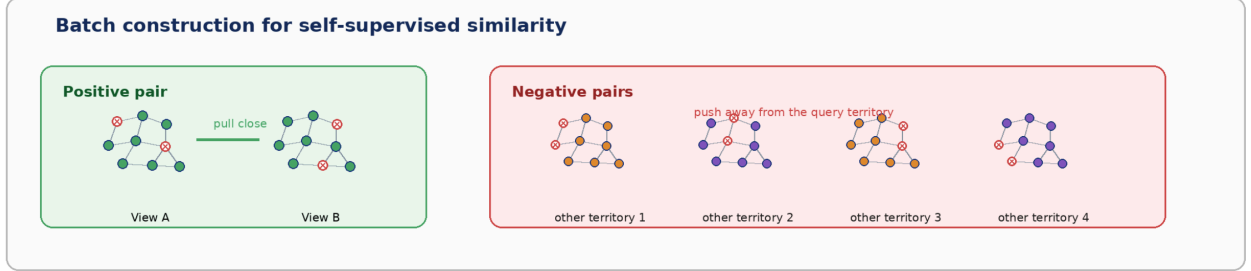


Figure 25 Batch construction for self-supervised similarity

4.6.3 Trajectory Similarity Results

Trajectory similarity is the main downstream task on the learned territory trajectory embeddings. In this task, the learned embeddings are used to retrieve the most similar SETT territory trajectories. Given a group of territory embeddings, the model ranks candidate territory embeddings according to their distance or similarity in the embedding space.

Since in our dataset we do not have any information about the similarity of such trajectories, we have followed a self-supervised training based on masking a percentage of nodes in each territory snapshot. This masking is done directly on the node feature matrix, so that a specified number of nodes randomly chosen are replaced with zero vector. Then, in each batch, we have two positive (masked) pairs belonging to the same territory and the others are different territories (negatives). As shown in Figure 25, each batch include two augmented snapshots belonging to the same territory, and other augmented snapshots for different territories. The positive pair refers to the two augmented snapshots of the same territory, while negative pairs indicate two different territory augmented snapshots. The role of the self-supervised training is to keep the positive pairs always close to each other's, meaning that there is always a small distance between their embeddings, and to increase the distance with the negative pairs.

Following this strategy, the first experiments were conducted on the Fribourg dataset without adding additional static or CLC features. Across three runs, the model obtained as shown in Table 2 Hits@1 values between 0.5769 and 0.6154, Hits@5 values between 0.8462 and 0.8846, and MRR values between 0.7052 and 0.7242. These results show that the model was already able to retrieve the correct trajectory within the top five candidates in most cases, but the correct trajectory was not always ranked first.

Configuration	Run	Minimum loss	Hits@1	Hits@5	MRR
Fribourg without additional features	1	0.5625	0.5769	0.8846	0.7052
Fribourg without additional features	2	0.5858	0.6154	0.8846	0.7242
Fribourg without additional features	3	0.5407	0.5769	0.8462	0.7177

Table 2 Self-supervised results on Fribourg dataset without static and CLC features

After adding territory-level static (LIS) features, the results improved significantly. Without feature masking, the model reached a minimum loss of 0.1505, Hits@1 of 0.8846, Hits@5 of 1.0000, and MRR of 0.9423. This indicates that the static territory descriptors provide important information for learning more discriminative territory embeddings.

Configuration	Run	Feature Masking	Minimum loss	Hits@1	Hits@5	MRR
Fribourg with LIS features	1	No masking	0.1505	0.8846	1.0000	0.9423
Fribourg with LIS features	2	30% masking	0.3032	0.7308	1.0000	0.8590
Fribourg with LIS features	3	50% masking	0.3940	0.6154	0.8462	0.7177

Table 3 Self-supervised results on Fribourg dataset with static features only

A similar experiment was conducted using only CLC land-cover features. Without masking, the model obtained Hits@1 of 0.8077, Hits@5 of 1.0000, and MRR of 0.8974. This confirms that land-cover information also contributes positively to the learned territory embeddings. However, when 50% of the CLC features were masked, the performance dropped to Hits@1 of 0.5769 and MRR of 0.7007, showing that excessive masking reduces the amount of useful information available to the model.

Configuration	Run	Feature Masking	Minimum loss	Hits@1	Hits@5	MRR
Fribourg with CLC only	1	No masking	0.2715	0.8077	1.0000	0.8947
Fribourg with CLC only	2	30% masking	0.4609	0.7308	1.0000	0.8333
Fribourg with CLC only	3	50% masking	0.4996	0.5769	0.8846	0.7007

Table 4 Self-supervised results on Fribourg dataset with CLC features only

The best results were obtained when both LIS and CLC features were added. Without feature masking, the model reached Hits@1 of 0.8846, Hits@5 of 1.0000, and MRR values between 0.9359 and 0.9423 across two runs. With 10% feature masking, the model still achieved strong performance, with Hits@1 of 0.8462 and MRR of 0.9231. This suggests that light masking can act as a useful self-supervised regularization mechanism, while strong masking makes the retrieval task harder.

Configuration	Run	Feature Masking	Minimum loss	Hits@1	Hits@5	MRR
Fribourg with LIS + CLC	1	No masking	0.1318	0.8846	1.0000	0.9423
Fribourg with LIS + CLC	2	30% masking	0.2947	0.8077	1.0000	0.8974
Fribourg with LIS + CLC	3	50% masking	0.3703	0.7308	0.9615	0.8218

Table 5 Self-supervised results on Fribourg dataset with LIS and CLC features

Overall, the trajectory similarity results show that the learned embeddings become more effective when additional territory-level features are included. The baseline model without these features already retrieves the correct trajectory in the top five candidates in most cases, but adding LIS and CLC information improves the ranking quality and increases the probability that the correct trajectory is ranked first.

4.6.4 Ablation Studies

Configuration	Minimum Loss	Hits@1	Hits@5	MRR
SETT-model	0.1743	0.9231	1.0000	0.9615
SETT-model without time and positional encoders	0.197580	0.8462	1.0000	0.9231
SETT-model without Transformer	0.295416	0.7308	1.0000	0.8442
SETT-model without Transformer taking the mean of episode embeddings	0.215948	0.8846	1.0000	0.9263

Table 6 Ablation results on Fribourg dataset with LIS and CLC features

- **Model Ablation Study**

Since SETT-GNN is composed of several architectural components, an ablation study was conducted to evaluate the contribution of each module to the self-supervised trajectory similarity task. The full SETT-GNN model was compared with three reduced variants: one without the temporal and positional encoders, one without the Transformer encoder using only the last episode embedding as the trajectory representation, and one without the Transformer encoder using mean pooling over all episode embeddings.

Table 6 reports the results of the model ablation study. The full SETT-GNN model achieves the best overall performance, with a minimum loss of 0.1743, Hits@1 of 0.9231, Hits@5 of 1.0000, and an MRR of 0.9615. This indicates that the complete architecture is the most effective configuration for learning discriminative territory trajectory embeddings. In particular, the Hits@5 score of 1.0000 shows that the correct positive pair is always ranked among the top five retrieved candidates.

Removing the temporal and positional encoders leads to a decrease in performance. Hits@1 drops from 0.9231 to 0.8462, while MRR decreases from 0.9615 to 0.9231. Although Hits@5 remains equal to 1.0000, the reduction in Hits@1 and MRR shows that temporal information contributes to more precise ranking. This suggests that episode duration and episode order provide useful signals for representing the evolution of a territory trajectory.

A stronger performance drop is observed when the Transformer encoder is removed and the last episode embedding is used as the final territory trajectory representation. In this configuration, Hits@1 decreases to 0.7308 and MRR drops to 0.8442. This confirms the importance of sequence-level reasoning between episodes. Without the Transformer, the model loses the ability to explicitly integrate information across the whole sequence of episode embeddings, and the final representation becomes less effective for trajectory similarity retrieval.

The variant without the Transformer but using the mean of all episode embeddings performs better than using only the last episode embedding. It achieves a Hits@1 of 0.8846 and an MRR of 0.9263. This shows that aggregating all episode embeddings is more informative than relying only on the last episode. However, this variant still remains less effective than the full Transformer-based model. Therefore, mean pooling provides a competitive but simpler aggregation strategy, while the Transformer remains the most effective approach because it models the sequential dependencies between episodes.

Overall, the ablation results show that each component contributes to the final performance. The temporal and positional encoders help the model capture episode duration and order, while the Transformer encoder plays a central role in modeling the trajectory as a sequence of evolving episode representations.

- **Data Ablation Study**

The data ablation study was designed to understand the effect of different features added on the territory snapshots from static and CLC metadata. Evaluating how these features will affect self-supervised training and will they keep helping the model to find the correct territory after masking.

Masking was done on the node feature matrix of the territory-episode snapshot as indicated before, and after the addition of static and CLC features we also masked with the same percentage these features in the territory feature vector.

The first observation is that adding LIS static features significantly improves the results compared with the baseline configuration without additional features. This suggests that static territory descriptors such as area, elevation, and slope-related attributes provide useful information for distinguishing territories.

The second observation is that CLC features also improve the learned representations. Since CLC features describe land-cover composition, they provide complementary information about the environmental structure of each territory. This helps the model learn embeddings that better reflect territorial characteristics.

The third observation concerns feature masking. Masking a small percentage of features can still preserve strong performance, especially when both LIS and CLC features are used. For example, with 10% masking, the model remains close to the no-mask performance. However, stronger masking, such as 50%, leads to a clear performance decrease. This indicates that the model benefits from self-supervised masking, but only when enough informative features remain available.

4.7 Discussion

The experimental results obtained show that SETT-GNN pipeline can learn useful territory trajectory embeddings, especially for the trajectory similarity task. The initial evaluation of Fribourg dataset without any additional features showed that the model can learn the special semantics of each territory and use it to distinguish between positive and negative pairs. This indicates that the graph-based representation already captures useful information from the SETT episode snapshots.

As the results showed after adding territory-level static features, the quality of the learned embedding was improved. The injection of territory-level static features led to a clear improvement in Hits@1, Hits@5, and MRR. This suggests that static territorial descriptors, such as area, elevation, and slope-related information, provide useful complementary information to the semantic graph structure. Similarly, the use of CLC land-cover features improved the retrieval results, showing that land-cover composition is also relevant for distinguishing territories.

The best results were obtained when combining LIS and CLC features. This confirms that the two groups of features provide complementary information. While LIS features describe physical and topographical characteristics of the territory, CLC features describe its land-cover composition. Their combination helps the model build more discriminative territory embeddings.

The feature masking experiments show that the model remains able to learn meaningful embeddings even when part of the territory feature vector is masked. With small masking percentages, the model preserves strong performance, indicating that the learned embeddings are robust and do not depend on a single feature source. However, stronger masking, such as 50%, leads to a performance decrease, which is expected because a larger part of the input information is removed. Importantly, the performance does not collapse, suggesting that the model benefits from both the graph structure and the injected territory features.

The model ablation study further confirms the importance of the different components of SETT-GNN. The full SETT-GNN model achieves the best performance, showing that the combination of relation-aware graph encoding, temporal and positional encoding, and Transformer-based trajectory modeling is the most effective architecture. Removing the temporal and positional encoders decreases the results, which shows that episode duration and episode order contribute to the construction of the trajectory representation. Removing the Transformer causes a stronger performance drop, confirming the importance of sequence-level reasoning

between episodes. Mean pooling over episode embeddings remains competitive, but it is less effective than the full Transformer model, indicating that simple aggregation can capture part of the trajectory information but cannot fully replace explicit sequence modeling.

Overall, the results validate the usefulness of SETT-GNN embeddings for trajectory similarity. The model is able to learn territory-level representations that are useful for retrieving similar trajectories in a self-supervised setting. Classification and clustering were also identified as possible downstream tasks, but the strongest quantitative evaluation in this work concerns self-supervised similarity retrieval. Future work could extend the evaluation by using reliable trajectory labels for classification and by analyzing discovered clusters in relation to known territorial behaviors.

5 Conclusions and Future Work

This chapter concludes the report by summarizing the outcomes of the work done on Graph Neural Networks for Semantic Environmental Trajectories of Territories (SETT). The objective of this thesis was to study the possibility to apply GNNs on SETT trajectories and obtain meaningful vector representations for territory trajectories.

The work was guided by several research questions about the existing GNN architectures applied on KGs, their ability to fit out SETT dataset, the representation of SETT dataset for GNN processing, the ability to have an end-to-end pipeline learning meaningful territory trajectory representations, and the possible downstream tasks that can be performed using these representations.

The chapter is organized into four sections. Section 5.1 presents the main research questions and how they are addressed. Section 5.2 presents the main lessons learned from the study. Section 5.3 presents the limitations faced in this work. Finally, Section 5.4 presents possible future research directions for improving and extending the proposed approach.

5.1 Contributions

This report has addressed the problem of learning representations from SETT represented as Spatiotemporal Knowledge Graphs. The work was guided by five main research questions:

1. What are the existing Graph Neural Network approaches for Knowledge Graph representation learning?
2. Which of these approaches we analyzed could be applied to or adapted for SETT Knowledge Graphs?
3. How can SETT KG be exploited within a learning pipeline for trajectory representation learning?
4. Could the learned territory trajectory representations capture meaningful information about territory trajectories?
5. What are the possible downstream tasks that can be performed using these learned trajectory representations?

These questions were addressed through four main contributions. The first contribution was a state-of-the-art study of GNN approaches for Knowledge Graphs and semantic trajectories, starting from traditional GCNs and moving toward spatial GNNs where we analyzed R-GCN, then we explored temporal GNN approaches and we discussed temporal KG representation for GNN input, reaching spatiotemporal GNNs where we analyzed two important frameworks RE-GCN and THAM which we are inspired from.

The second contribution was the proposal of an adapted architecture, SETT-GNN, designed to represent SETT spatiotemporal Knowledge Graphs. Inspired by THAM architecture, we adapted their framework to represent out SETT KGs which is different than patient diagnosis trajectories they used.

The third contribution was the implementation of a complete end-to-end pipeline, from RDF graph preprocessing and episode-snapshot construction to model implementation and representation learning. The pipeline first parses the original SETT KG and extracts the main ontology elements to create snapshots based on each territory episode, such that each territory trajectory is represented as a sequence of territory-episode snapshots. Then, it converts each snapshot into a PyTorch Geometric data object containing the required input for R-GCN. Finally, the implemented model processes these inputs to produce one vector representation for each territory trajectory.

The fourth contribution was an experimental evaluation of the learned representations, mainly through trajectory similarity, data ablation studies, and model ablation studies. Since no strong supervised ground truth was available, the evaluation focused on whether the model could learn meaningful embeddings in a self-

supervised setting. Trajectory similarity was evaluated using ranking metrics such as Hits@1, Hits@5, and MRR, to measure whether similar or corresponding territory representations were ranked close to each other. Several data ablation studies were also performed to analyze the impact of adding LIS static features, CLC land-cover features, and feature masking during training. The model ablation study showed that the full SETT-GNN architecture achieves the best performance, while removing the temporal and positional encoders or the Transformer encoder reduces the quality of the learned trajectory representations. These experiments showed that the learned representations become more informative when additional territory-level features are included, and that masking can be used as a self-supervised strategy to evaluate the robustness of the embeddings.

5.2 Lessons Learned

The first lesson learned from this work is that a study of the existing GNN approaches is needed before proposing a model for SETT Knowledge Graphs. This study is necessary because SETT data combines several characteristics that are not usually handled together by standard graph learning models, such as heterogeneous relations, temporal evolution, and trajectory-level representation. Without this preliminary analysis, it would be difficult to justify which model components are suitable for SETT and which ones require adaptation. Traditional GNN models, such as GCN and GAT, provide a strong basis for understanding message passing and node representation learning. However, they are not directly sufficient for SETT data because they do not explicitly model heterogeneous relation types or temporal evolution. Relational models, such as R-GCN, are more suitable for Knowledge Graphs because they distinguish between different predicates during message passing. Temporal and spatiotemporal models also provide important ideas for representing evolution over time, but most existing models are designed for tasks that are different from the SETT setting.

The second lesson is that representing the data correctly is a fundamental step before applying any graph learning framework. SETT trajectories are not simple graphs. They contain territories, thematic trajectories, episodes, events, segments, transition points, attributes, values, and temporal intervals. Therefore, before applying any GNN model, it has been necessary to transform the original SETT Knowledge Graph into a structure that can be processed by graph neural networks. The construction of territory-episode snapshots was a key step in this work, because it allowed each episode of a territory trajectory to be represented as a local semantic graph.

The third lesson is that SETT requires both graph-level and sequence-level reasoning. Each episode-snapshot contains local semantic information that can be captured by a relation-aware GNN. However, the full territory trajectory is not limited to one episode. It is composed of several ordered episodes with different temporal intervals. For this reason, the proposed architecture combines an R-GCN episode encoder with temporal and positional encodings and a Transformer-based sequence encoder. This combination makes it possible to represent both the local semantic structure of each episode and the temporal evolution of the territory trajectory. The model ablation results confirm this observation, since removing the temporal and positional encoders decreases retrieval performance, and removing the Transformer encoder leads to a stronger performance drop.

The fourth lesson is related to the evaluation phase. Since reliable supervised labels are not always available for territory trajectories, evaluating the quality of the learned representations is not straightforward. In this work, trajectory similarity is the main evaluation task, using ranking metrics such as Hits@K and MRR. The results show that the learned embeddings can retrieve similar or corresponding territory trajectories, especially when territory-level features such as LIS and CLC features are included. This suggests that the model can capture meaningful information about territories, even if further validation is still needed.

Finally, this work has demonstrated that adapting a model inspired by THAM [24] to SETT trajectories is a valuable first step. THAM was originally designed for medical trajectories, where each patient is represented through a sequence of visits. In this work, the same general idea has been adapted to territories, where each territory is represented through a sequence of episode-snapshots. This adaptation shows that the idea of combining local graph encoding with temporal sequence modeling can be useful beyond the medical domain, especially for semantic trajectory representation.

5.3 Limitations

Despite the value of this first implementation, several limitations remain. The first limitation concerns the structure of the available SETT datasets. In the current datasets, many territory trajectories contain only two or three episodes. This makes the temporal sequence relatively short. As a result, it is difficult to fully evaluate whether the Transformer-based temporal encoder remains stable and useful when the number of episodes becomes larger. Testing the model on trajectories with longer sequences would provide a better understanding of its ability to capture long-term temporal evolution.

The second limitation is the lack of ground-truth labels. Because no strong supervised labels were available for the territory trajectories, the model could not be fully evaluated on supervised prediction tasks. This limits the possibility of testing classification or prediction in a rigorous way. For this reason, the evaluation mainly relied on self-supervised similarity tasks and data ablation studies. These results are useful, but they do not fully prove that the learned representations correspond to expert-defined territorial behaviours.

The third limitation is that the model treats each territory independently. For each territory, the model collects its episode-snapshots and processes them as an independent sequence. This design is coherent with the objective of learning one trajectory embedding per territory, but it does not explicitly model interactions between neighboring territories. In a real spatiotemporal setting, the evolution of one territory may be related to the evolution of nearby territories. Integrating spatial neighborhood information between territories, and their multilevel spatial inclusion relations, could therefore improve the representation.

Another limitation is related to scalability and parallelization. Since the current model builds and processes a sequence of episode-snapshots for each territory, the pipeline can become less parallelizable when the number of territories, episodes, or graph variables increases. This issue may become more important when applying the model to larger datasets or more complex semantic trajectories.

An additional limitation concerns the number and type of variables included in each episode. In the current implementation, the model considers the available node features and additional territory-level features, but it does not explicitly learn which variables are more important than others. If future SETT datasets include a larger number of variables of different types, the model may need a mechanism to assign different importance weights to variables inside each episode.

Finally, the current model treats thematic trajectories in a relatively uniform way. However, in real environmental analysis, some themes may be more important than others depending on the region, the task, or the studied phenomenon. For example, vegetation may be more important in one context, while snow or urbanization may be more relevant in another. The model does not yet include a mechanism to learn weighted thematic importance.

5.4 Future Work

The limitations identified before opens several avenues of research to explore. The first direction is to improve the evaluation of the learned representations. Since ground truth is currently limited, one possible approach is to perform an additional validation process based on clustering and pseudo-label prediction. First, territories can be clustered using the embeddings learned by SETT-GNN. Then, the resulting clusters can be used as pseudo-labels for a prediction task. If the model can predict these cluster-based classes in a stable way, this would provide an additional indication that the learned embeddings capture meaningful structure. However, such pseudo-labels should be interpreted carefully, since they are derived from the model itself and not from external expert annotations.

A second direction is to evaluate the model on longer and more complex trajectories. Semantic Life Trajectories are another type of Semantic Trajectories about individuals' life courses that follow a trajectory pattern very similar to SETT with thematic trajectories, episodes and events. In particular, the 3B life course datasets contain biographical data on 5,000 individuals in France collected by INSEE and INED in 1981. The life courses in this dataset are organized around three thematic trajectories (family, career and residential), each of which comprises, on average, a far greater number of episodes than the SETT. Applying the proposed architecture to these life trajectories would make it possible to test whether the model remains stable when the

temporal sequence is longer and more complex. This would also open the possibility of extending the model beyond territorial trajectories toward other forms of semantic trajectories.

A third direction is to improve clustering-based analysis. One possible idea is to perform two-level clustering. First, territories can be clustered according to their CLC land-cover profiles. Then, inside each land-cover cluster, the learned SETT-GNN embeddings can be used to identify territories with similar temporal behaviors. This would make the analysis more interpretable, because territories would first be grouped according to their physical or land-cover characteristics and then compared according to their semantic trajectories.

Finally, future work could also introduce thematic weighting. Since SETT trajectories are composed of several thematic trajectories, such as vegetation, water, snow, temperature, and urbanization, the model could learn which themes are more important for a given territory or task. This would make the representation more flexible and more interpretable, especially when some themes are more informative than others.

Overall, this work represents a first step toward learning neural representations from Semantic Trajectories of Territories. It shows that SETT Knowledge Graphs can be transformed into episode-snapshot sequences and processed using a hybrid architecture combining relation-aware graph encoding and temporal sequence modeling. Although further evaluation and extensions are still needed, the proposed framework provides a useful foundation for future research on SETT and, more generally, on semantic trajectory representation learning.

6 References

- [1] Knowledge Graphs, Aidan Hogan et al. [CrossRef]
- [2] Spatio-Temporal Graph Neural Networks: A Survey, Zahraa Al Sahili et al. 2023 [CrossRef]
- [3] Edward W. Schneider. 1973. Course Modularization Applied: The Interface System and Its Implications For Sequence Control and Data Analysis. In Association for the Development of Instructional Systems (ADIS), Chicago, Illinois, April 1972.
- [4] Berners-Lee, T.; Hendler, J.; Lassila, O. The Semantic Web. *Sci. Am.* 2001, 284, 28–37. [CrossRef]
- [5] Shadbolt, N.; Berners-Lee, T.; Hall, W. The Semantic Web Revisited. *IEEE Intell. Syst.* 2006, 21, 96–101. [CrossRef]
- [6] Heath, T.; Bizer, C. *Linked Data: Evolving the Web into a Global Data Space; Synthesis Lectures on Data, Semantics, and Knowledge (SLDSK) Series*; Springer: Cham, Switzerland, 2011. [CrossRef]
- [7] Dbpedia: <https://dbpedia.org>
- [8] WikiData: <https://www.wikidata.org/wiki>
- [9] McInnes, L., Healy, J., & Saul, N. (2018). UMAP: Uniform Manifold Approximation and Projection. *Journal of Open Source Software*, 3(29), 861.
- [10] H. S. Al Khatib, S. Neupane, H. K. Manchukonda, N. A. Golilarz, S. Mittal, A. Amirlatifi, and S. Rahimi, “Patient-Centric Knowledge Graphs: A Survey of Current Methods, Challenges, and Applications,” *arXiv preprint arXiv:2402.12608*, 2024. doi: 10.48550/arXiv.2402.12608.
- [11] K. E. Yuksel and S. Uskudarli, “Incorporating Knowledge Graph Embeddings into Graph Neural Networks for Sequential Recommender Systems,” in *2024 9th International Conference on Computer Science and Engineering (UBMK)*, 2024, pp. 517–522, doi: 10.1109/UBMK63289.2024.10773537.
- [12] Y. Tang, Z. Zhao, W. Deng, S. Lei, Y. Liang, and Z. Ma, “RouteKG: A knowledge graph-based framework for route prediction on road networks,” *IEEE Transactions on Intelligent Transportation Systems*, 2025. arXiv:2310.03617.
- [13] C. Mishra, “A real-time evolution of security knowledge graphs using temporal and agentic framework,” Ph.D. dissertation, Dept. Computer Science, Indian Institute of Information Technology, Sri City, Chittoor, India, 2025.
- [14] D. F. Milon Flores, “OFF-SETT: An ontology-driven framework for semantic environmental trajectories of territories,” Ph.D. dissertation, Université Grenoble Alpes, Grenoble, France, 2025.
- [15] C. Bernard, J. Gensel, D. F. Milon-Flores, G. Giuliani, and M. Villanova, “OFF-SETT: A semantic framework for annotating trends in spatiotemporal data,” *ISPRS International Journal of Geo-Information*, vol. 15, no. 3, Art. no. 132, 2026, doi: 10.3390/ijgi15030132.
- [16] Zhao, K.; Wulder, M.A.; Hu, T.; Bright, R.; Wu, Q.; Qin, H.; Li, Y.; Toman, E.; Mallick, B.; Zhang, X.; et al. Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm. *Remote Sens. Environ.* 2019, 232, 111181. [CrossRef]
- [17] Zhu, Z.; Woodcock, C.E. Continuous change detection and classification of land cover using all available Landsat data. *Remote Sens. Environ.* 2014, 144, 152–171. [CrossRef]
- [18] Truong, C.; Oudre, L.; Vayatis, N. Selective review of offline change point detection methods. *Signal Process.* 2020, 167, 107299. [CrossRef]
- [19] Ermshaus, A.; Schäfer, P.; Leser, U. ClaSP: Parameter-free time series segmentation. *Data Min. Knowl. Discov.* 2023, 37, 1262–1300. [CrossRef]
- [20] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” in *Proc. International Conference on Learning Representations (ICLR)*, 2018.
- [21] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, “Modeling relational data with graph convolutional networks,” in *Proc. Extended Semantic Web Conference (ESWC)*, 2018, pp. 593–607.
- [22] C. Xu, F. Su, and J. Lehmann, “Time-aware graph neural networks for entity alignment between temporal knowledge graphs,” in *Proc. 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021, pp. 8999–9010.

- [23] Z. Li, X. Jin, W. Li, S. Guan, J. Guo, H. Shen, Y. Wang, and X. Cheng, “Temporal knowledge graph reasoning based on evolutionary representation learning,” in *Proc. 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2021, pp. 408–417, doi: 10.1145/3404835.3462963.
- [24] S. Li, H. Cheng, R. Li, and W. Li, “Time-aware heterogeneous graph transformer with adaptive attention merging for health event prediction,” *arXiv preprint arXiv:2404.14815*, 2024.
- [25] Python Software Foundation, “Python Language Reference, version 3.x,” Python Software Foundation. [Online]. Available: <https://www.python.org>
- [26] A. Paszke *et al.*, “PyTorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 8024–8035.
- [27] M. Fey and J. E. Lenssen, “Fast graph representation learning with PyTorch Geometric,” *arXiv preprint arXiv:1903.02428*, 2019.
- [28] C. R. Harris *et al.*, “Array programming with NumPy,” *Nature*, vol. 585, pp. 357–362, 2020, doi: 10.1038/s41586-020-2649-2
- [29] RDFLib Team, “RDFLib: A Python library for working with RDF,” 2024. [Online]. Available: <https://rdflib.readthedocs.io/>
- [30] A. A. Hagberg, D. A. Schult, and P. J. Swart, “Exploring network structure, dynamics, and function using NetworkX,” in *Proc. 7th Python in Science Conference (SciPy 2008)*, Pasadena, CA, USA, 2008, pp. 11–15.
- [31] Semi-Supervised Classification With Graph Convolutional Networks, DOI: [10.13140/RG.2.2.22993.71526](https://doi.org/10.13140/RG.2.2.22993.71526)
- [32] CS224W: Machine Learning with Graphs, <https://web.stanford.edu/class/cs224w/>
- [33] A Graph Neural Network with Spatio-Temporal Attention for Multi-Sources Time Series Data: An Application to Frost Forecast, DOI: [10.3390/s22041486](https://doi.org/10.3390/s22041486)
- [34] <https://traces-anr-fns.imag.fr>