



Università di Pisa

Consiglio Nazionale delle Ricerche

National Ph.D. in Artificial Intelligence

XXXVIII cycle

Efficient Neural Community Detection on Time-Varying Attributed Graphs

Candidate
Nelson Aloysio Reis de Almeida Passos

Supervisors
Dr. Emanuele Carlini
Dr. Salvatore Trani

Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Artificial Intelligence

To shared dreams

Summary

The compendium of community detection techniques has been significantly enriched in recent years — including by the introduction of *neural* strategies for *learning* on graphs.

This research investigates these techniques for relational systems whose topology evolves over time and whose nodes or edges may carry informative features. Although neural models have demonstrated state-of-the-art performance for node-, link- and graph-level prediction tasks, in applications such as recommendation systems, anomaly detection, and protein design, their utility for node-level clustering — i.e., detecting communities — warrants further investigation, particularly in temporal graph settings.

The motivation for this work stems from the need from diverse domains for robust and scalable methods for mesoscale graph analysis, where community structure can provide insight into the organization and dynamics of complex systems. Departing from a simple comparative question — *can neural strategies outperform established baselines?* — it evolved into two complementary strands: a (i) mature algorithmic route, relying on GPU-accelerated implementations of spectral and modularity-based algorithms; and (ii) an ongoing neural route, focused on scalable architectures for node clustering in graphs with high-dimensional features through differentiable graph-theoretic objectives.

This thesis is organized around the theoretical foundations and methodological contributions needed to address such problem. Part I (Chapters 2 and 3) introduces foundational temporal graph concepts and presents NetworkX-Temporal, a software library for building, manipulating, and analyzing such graphs through a familiar and extensible API. Part II (Chapters 4 and 5) addresses community detection and model evaluation, comparing static and dynamic methods from descriptive and inferential perspectives and presenting two methodological contributions: high-performance implementations of spectral and modularity-based algorithms, integrated into the software library; and TADC-SBM, a time-varying, attributed, degree-corrected stochastic block model generator for temporal graphs with known ground truths and community-aligned node and edge features, used for principled benchmarking under controlled experimental settings. Part III (Chapters 6 and 7) turns to machine learning on graphs, discussing neural community detection and presenting LMoN, a Longitudinal Modularity Network that optimizes a continuous-time variant of modularity as an end-to-end differentiable training objective, avoiding the post-processing clustering stage required by many methods.

The software and datasets introduced in this work have been made publicly available to support reproducibility and further research in the area. Together, these contributions advance the study of community detection in temporal graphs, combining software infrastructure, scalable algorithms, and neural methods grounded in graph-theoretic principles and rigorous evaluation frameworks. This thesis concludes with a discussion of the identified limitations and an outline of potential directions for future research.

Acknowledgements

Time to thank those who made this possible: in my case, family, friends, and mentors.

Science is, arguably so, the collection of knowledge we accumulate about the world and ourselves. It is through the support and encouragement of those close to me that I have been able to pursue this research, aiming to contribute to such an endeavor.

I am grateful to my family, who instilled in me the value of intelligence and persistence; my friends, for their companionship and camaraderie, constant and motivating sources of joy throughout life; my supervisors and colleagues, for their guidance and stimulating discussions, which enriched this research and helped me navigate academia; and all those who believed in or challenged me — sometimes both at once — contributing to my growth as a human and a researcher. Thinking of whom to mention makes numerous people from different circles come to mind: my parents Solange and Francisco, with their unconditional love and support; my brother Thales, with his iron-clad will; my accomplice Guilherme, with his sharp wit; my mentors Emanuele and Salvatore, with their patient wisdom; and many more, all part of this journey in a way or another. Including *you*, who came this far to read my solipsisms (*why?*) — I hope I am able to reciprocate the kindness and support I have received from all my dear ones, now and ever.

An early version of this research was discussed with Maurizio Tesconi (CNR-IIT) and Ricardo Guidotti (UNIPI), toward the end of its second year. The thesis manuscript was reviewed ahead of defense by Amilcar Soares (Linnaeus University) and Konstantinos Tserpes (National Technical University of Athens). I thank them for their time, insightful feedback, and valuable encouragement, which helped improve the quality of this work.

Part of this study was carried out at the Department of Network and Data Science, Central European University in Vienna, Austria, and the Interdisciplinary Transformation University in Linz, Austria, under the supervision of Tiago Peixoto. I thank them for the stimulating environments and fruitful discussions during the spring of 2024 and the winter of 2026, which greatly contributed to this research, especially Chapter 5.

This research was supported by a grant funded by CNR-ISTI (Pisa) and CNR/FOE, within the scope of the Italian National PhD in Artificial Intelligence, executive provision no. 141068 of October 24, 2022. I thank the institutions for the opportunity and support.

Contents

Summary i

Acknowledgements iii

List of Figures 1

List of Tables 3

List of Symbols 5

1 Introduction 7
- 1.1 Challenging motivations . . . 9
  - 1.1.1 Machines learning on graphs . . . 11
  - 1.1.2 Do electric sheep dream of communities? . . . 12
- 1.2 Motivating advancements . . . 12
  - 1.2.1 On temporal graph structures and data . . . 13
  - 1.2.2 On algorithmic clustering and benchmarks . . . 14
  - 1.2.3 On neural community detection . . . 15
- 1.3 Publications . . . 16

I Temporal Networks 19

2 Temporal Graphs 21
- 2.1 Temporal graph representations . . . 22
  - 2.1.1 Discrete-time temporal graphs . . . 22
  - 2.1.2 Continuous-time temporal graphs . . . 23
  - 2.1.3 Higher-order temporal graphs . . . 24
- 2.2 Spectral graph properties . . . 25
  - 2.2.1 Spectral operators and clustering . . . 25
  - 2.2.2 Temporal graph spectra . . . 27

3 Dynamic Graph Modeling 31
- 3.1 Software advancements for temporal networks . . . 32
  - 3.1.1 The NetworkX-Temporal software library . . . 32
  - 3.1.2 The PubMed-Temporal graph dataset . . . 41
- 3.2 Applications to the study of graph dynamics . . . 45
  - 3.2.1 Reconstruction and clustering with deep autoencoders . . . 45

3.2.2 Entropy and information processing in temporal graphs 49

II Community Detection 51

4 Detecting Communities in Networks 53
4.1 Descriptive and inferential methods 54
4.2 Static and dynamic community detection 55
4.2.1 Spectral decomposition and matrix factorization 56
4.2.2 Heuristic optimization of quality functions 59
4.2.3 Inference and generative models 63

5 Algorithms and Benchmarks 67
5.1 The TADC-SBM generative model 68
5.1.1 Edge sampling procedure 68
5.1.2 Feature attribution 70
5.1.3 Benchmarking protocols 71
5.2 Scalable temporal community detection 74
5.2.1 Implementation details 74
5.2.2 Experimental evaluation and discussion 77

III Learning on Graphs 81

6 Machine Learning on Graphs 83
6.1 Neural networks for graphs 84
6.1.1 Graph neural network architectures 86
6.2 Community detection with neural networks 92
6.2.1 Neural methods for community detection 94
6.2.2 Experimental evaluation of models 96
6.2.3 Ground truths on neural community detection 104

7 Neural Community Detection 107
7.1 The LMoN graph neural network 108
7.1.1 Optimization objective 109
7.1.2 Encoding strategy 111
7.1.3 Implementation details 113
7.2 Experimental evaluation and discussion 115
7.2.1 Real-world temporal graphs 116
7.2.2 Synthetic temporal graphs 118
7.2.3 Ablation study and comparison with spectral baseline 119

IV Final Remarks 123

Bibliography 127

List of Figures

1.1 Research schematic 13

2.1 Temporal graph represented as three snapshots 22
2.2 Network representation models 24

3.1 NetworkX-Temporal software architecture 33
3.2 Community detection by modularity optimization 35
3.3 Temporal graph representations 36
3.4 Temporal node and edge matrices for CollegeMsg 37
3.5 Comparison of degree-sequence generators 38
3.6 PubMed-Temporal graph characteristics 41
3.7 Transductive and inductive learning settings 43
3.8 Degree distribution of the PubMed graph 44
3.9 Node and edge set overlap in PubMed-Temporal 44
3.10 Graph augmentation for temporal node clustering 46
3.11 DAEGC architecture for attributed graph clustering 47
3.12 Augmented PubMed-Temporal clustering results 48
3.13 Complex coordination networks 49

4.1 Example graph and possible community partition 56

5.1 TADC-SBM model illustration 69
5.2 TADC-SBM feature type comparison 70
5.3 TADC-SBM baseline evaluation results 73
5.4 GPU acceleration of NetworkX-Temporal 75
5.5 Temporal Leiden CPU/GPU runtimes 77

6.1 Comparison of message-passing layers 86
6.2 Color refinement of decalin and bicyclopentyl 90
6.3 Real-world graph baselines 98
6.4 Synthetic graph baselines 101
6.5 SBM log-likelihood for Zachary's Karate Club 105

7.1 LMoN architecture 108

List of Tables

3.1 NetworkX-Temporal software modules 32
3.2 Temporal graph representations 39
3.3 NetworkX-Temporal conversion functions 40
3.4 PubMed temporal graph dataset statistics 42
3.5 Augmented PubMed-Temporal clustering results 48

4.1 Descriptive versus inferential approaches 54

5.1 TADC-SBM feature type comparison 71
5.2 TADC-SBM baseline graphs 72
5.3 TADC-SBM baseline evaluation results 73
5.4 Temporal Leiden CPU/GPU runtimes 78
5.5 Evaluated dataset properties 79

6.1 Evaluation baselines on real-world graphs 99
6.2 Evaluation baselines on SBM graphs 102
6.3 Leiden baselines on SBM graphs 103

7.1 LMoN versus DMoN on real-world temporal graphs 117
7.2 LMoN versus DMoN on synthetic TADC-SBM graphs 117
7.3 LMoN ablation on synthetic TADC-SBM graphs 120
7.4 Neural and spectral clustering on real-world and synthetic graphs 121

List of Symbols

G	Static graph
$\mathcal{G}$	Temporal graph
$\mathcal{G}_C$	Continuous-time temporal graph
$\mathcal{G}_D$	Discrete-time temporal graph
v	Node (vertex)
V	Set of nodes (vertices)
$\mathcal{V}$	Set of temporal nodes
$\mathcal{V}_T$	Set of temporal node-time pairs
e	Edge (arc)
E	Set of edges
$\mathcal{E}$	Set of temporal edges
$\mathcal{E}^V$	Set of node events
$\mathcal{E}^E$	Set of edge events
ε	Event (edge-level)
$\mathcal{C}$	Set of communities
C_i	Community (node subset)
c_i	Community label of node i
$\mathbf{A}$	Adjacency matrix
$\tilde{\mathbf{A}}$	Normalized adjacency matrix
$\hat{\mathbf{A}}$	Temporally decayed adjacency matrix
$\mathbf{A}^{\text{supra}}$	Supra-adjacency matrix
$\mathbf{B}$	Block (affinity) matrix
$\mathbf{B}^{\text{nb}}$	Non-backtracking operator
$\mathbf{C}$	Community assignment matrix
$\mathbf{D}$	Degree matrix
$\mathbf{H}$	Latent (hidden) feature matrix
$\mathbf{H}(r)$	Bethe-Hessian operator
$\mathbf{I}$	Identity matrix
$\mathbf{J}$	All-ones matrix
$\mathbf{L}$	Laplacian matrix
$\mathbf{M}$	Modularity matrix
$\mathbf{P}$	Null model (expected degree) matrix
$\mathbf{X}$	Node feature matrix
$\mathbf{Z}$	Output (embedding) matrix
$\boldsymbol{\tau}$	Community transition matrix
$\boldsymbol{\Omega}$	Inter-slice coupling matrix
$\mathbf{x}$	Feature vector
$\mathbf{h}$	Latent (hidden) vector
$\mathbf{z}$	Output (community) vector
$\mathbf{d}$	Degree vector
$\mathbf{p}$	Temporal presence vector
$\mathbf{a}$	Attention vector
$\boldsymbol{\mu}_c$	Community centroid (feature space)
$\boldsymbol{\theta}$	Degree-correction parameters
n	Number of nodes
m	Number of edges
k	Number of communities
t	Time (timestamp, snapshot index)
T	Number of snapshots
d_i	Degree of node i
d_v	Node feature dimension
d_e	Edge attribute dimension
$F^{(\ell)}$	Embedding dimension at layer ℓ
Q	Modularity score
γ	Resolution parameter of modularity
ω	Temporal smoothness penalty weight
ω_{jsr}	Inter-slice coupling
ϑ	Collapse regularization weight
β	Hawkes temporal decay rate
$\rho \in [0,1]$	Hawkes temporal retention rate
α	Hawkes excitation amplitude
α_0	Power-law exponent
α_{ij}	Attention coefficient
λ	Community structural signal strength
λ_k	k-th eigenvalue
λ_{uv}	Poisson edge rate
$\lambda_{ij}(t)$	Hawkes intensity
η	Community temporal signal strength/prob.
χ_i	Number of community switches of i
$\xi \in \{0,1\}$	Current (0) vs. initial (1) membership
$\zeta \in [0,1]$	Sampled edge observation probability
μ_i	Hawkes baseline intensity of node i
r	Bethe-Hessian regularizer
c	Average node degree
$c_{\text{in}}, c_{\text{out}}$	Within/between community edge rates
σ	Nonlinear activation function
σ^2	Within-community variance
σ_c^2	Centroid feature variance
ϵ	Small positive constant ($\epsilon \ll 1$)
$z_u^{(t)}$	Community label of node u at time t
$\mathcal{N}$	Neighborhood function
$\delta(\cdot,\cdot)$	Kronecker delta
δ_v	Node event duration/indicator
δ_e	Edge event duration/indicator
Δ	Interval (change), e.g. Δt
ψ	Local (update) function
Ψ	Global (readout) function
ϕ	Local (message) function
Φ	Global (readout) transform
κ	Hawkes excitation kernel
$\sum$	Summation
$\bigoplus$	Aggregation operator (e.g., mean)
$\odot$	Hadamard (elementwise) product
Tr	Matrix trace (sum of diagonals)
$\mathbb{N}$	Set of natural numbers
$\mathbb{R}$	Set of real numbers

Chapter 1

Introduction

> *You think that because you understand 'one' that you must therefore understand 'two', because one and one make two. But you forget that you must also understand 'and'.*
>
> Sufi story, apud Donella Meadows (2008)

Systems of interacting objects — *networks* — are often represented as *graphs*: mathematical structures composed of nodes (vertices), connected among themselves by edges (arcs). Numerous real-world structures can be depicted as graphs, both naturally occurring — from simple molecules, with atoms as nodes and their chemical bonds as edges; to entire food webs, with biological species linked by their predation relations — and artificially formed — such as in railroad transportation, where train stations are linked by rail tracks; or the World Wide Web, in which pages are connected by hyperlinks.

The mathematical study of such structures is the subject of graph theory, which provides a powerful framework for modeling and analyzing a wide range of phenomena. Though the earliest recorded use of the word *graph* can be traced back to the 19th century, the conception of the field is widely attributed to the 18th century instead, with Euler's seminal work on the Seven Bridges of Königsberg (now Kaliningrad, Russia). By modeling the city's landmasses as nodes and the bridges connecting them as edges, Euler discovered that no walk crossing each bridge exactly once could exist, a problem later formalized as that of Eulerian paths and cycles [37], i.e., trails and circuits. This is considered the founding event of graph theory as a mathematical discipline, paving the way for network science to arise as the interdisciplinary study of *complex systems* and their properties in various domains [114]. The terms system, graph and network will hereon be used interchangeably without loss of generality, unless ambiguity arises.

Given a sufficiently complex system, understanding its function by analyzing individual components may prove an unfeasible task: unforeseen properties may emerge from the interactions among distinct sets, which may not be readily apparent when considering elements in isolation. Likewise, its different parts may behave and evolve in varied and unexpected ways: groups of nodes in the same network may display differing connectivity patterns, both assortative (homophilic) and disassortative (heterophilic), forming a number of *communities* characterized by distinct topological and/or attribute-based properties, which may influence their generative mechanisms and evolutionary dynam-

ics. Uncovering such mesoscale structures becomes an increasingly onerous task as the network grows in complexity, often requiring specialized methods and algorithms — an often-elusive challenge that has spurred significant efforts across disciplines [42].

Such a task is commonly referred to as *community detection* in the network science literature, and has formal connections to that of *clustering* on transactional (relational) data [62]. Arguably one of its most central problems, whose lineage can be traced to early matrix-based sociometry of the mid-20th century [41, 89], it has since thrived as a research topic and been continuously enriched by contributions from various fields, including computer science, statistics, physics, and biology [53, 74, 81]. During this time, a plethora of methods aiming to tackle the issue were introduced: from matrix decomposition to greedy heuristics and inferential techniques, and more recently, artificial neural networks optimized for modeling relational data — that is, graphs [44, 171].

However, *static* graph representations often fall short in modeling complex systems, as they do not account for the temporal dynamics inherent to many real-world processes. In fact, most natural and artificial systems are *dynamic* in nature: social interactions evolve, transportation routes are modified, and biological processes unfold over time. Further complicating matters, such networks are often *attributed* with node and edge-level information, which provides additional context for understanding their inner workings, but also poses additional challenges for their modeling and analysis. Without accounting for such data, crucial insights into the underlying mechanisms governing the behavior of complex systems may be overlooked, potentially leading to oversimplified or misleading descriptions that may hinder a model's ability to capture relevant patterns and induce suboptimal performance in downstream tasks, e.g., forecasting future interactions, detecting anomalous behavior, or recovering its hidden substructures.

This thesis mainly occupies itself with the latter — i.e., *how to detect communities in time-varying attributed graphs*, relational systems where connections between elements are not fixed over time, and whose elements are (optionally) associated with diverse feature sets. This problem is particularly challenging due to the intricate interplay between temporal dynamics, topological structure, and attribute information, which are now central to modeling complex systems across many real-world applications [102]. To tackle it, this research followed a multi-step study (see Figure 1.1 toward the end of this chapter for a schematic overview), leveraging recent advancements in GPU-based computing and deep learning to introduce scalable strategies to extract community structure from such systems at scale, together with software contributions to temporal graph modeling and systematic evaluation on downstream tasks — altogether pursued in order to enable answering the original question on how to improve the state of the art in this domain.

A 'neural' community is here defined as a set of nodes in a real-dimensional space, *clustered*[1] together by a neural model optimizing a graph-theoretic objective — so as to disambiguate it from groups obtained by algorithms applied directly to relational data, e.g., spectral clustering, or by statistical methods that infer posterior distributions under explicit generative assumptions, e.g., stochastic blockmodeling. Both strategies are nonetheless employed throughout this study for several purposes, such as comparison baselines, evaluation metrics, and model benchmarking. By addressing the described problem, the goal is to contribute to the understanding of complex networks and their dynamics, providing insights that may inform future research and applications.

[1] The terms cluster, community, module, and partition are often used interchangeably in the literature.

This work is structured as follows. In the remainder of this introduction, the motivations and challenges behind this research are first substantiated, and the contributions made throughout it are outlined. Even-numbered chapters (2, 4 and 6) present the background and key theoretical concepts this thesis engages with, particularly on temporal graphs, community detection, and graph representation learning, which summarize the state of the art from which this research departed. Odd-numbered chapters (3, 5 and 7) present the novel contributions advancing each topic, including a software library for building and manipulating temporal graphs and a temporal graph dataset with node-level features; high-performance clustering algorithms and principled benchmarking with synthetic graphs; and a neural network architecture for end-to-end differentiable community detection on attributed graphs, respectively. Each part is connected by theory and application, and remains sufficiently self-contained to be read independently. This thesis then concludes by summarizing the main findings and limitations, connecting it to previous discussions, and outlining potential avenues for future research.

1.1 Challenging motivations

Partitioning a network's components into groups of nodes is not a trivial task. Although a well-studied problem, with a rich body of literature exploring various methods and algorithms, the definition of a *community* is itself contentious and context-dependent. The task of their detection is often considered an 'ill-defined' or 'ill-posed' problem [44, 170], as it does not have a unique, universal solution, and competing methods may encode distinct notions of community structure, not always comparable to each other [158].

In general, communities are frequently defined as groups of nodes more densely connected among themselves than with the rest of the network — 'birds of a feather flock together' [108] — a definition that assumes assortative connectivity patterns, where nodes preferentially connect to others with shared characteristics. However, it does not hold in several instances: disassortative networks, where nodes preferentially connect to those with differing features; overlapping (mixed-membership) communities, where nodes may simultaneously take part in multiple groups with different degrees of affiliation; or hierarchical structures, where communities are nested within larger groups, forming a multi-level organization. In such cases, the simple definition of a community is void, warranting alternative definitions and, consequently, more specialized methods for their detection. Moreover, the presence of node and edge attributes increases the complexity of the task, as communities may also be characterized by shared feature sets, rather than solely by their connectivity patterns — while such features may not necessarily align with the network's topology and mesoscale patterns of connectivity, further complicating the endeavor [131]. Traditional methods for community detection, including widely used ones such as modularity optimization and spectral clustering, struggle to capture these nuances or simply disregard them altogether [135]. They also have inherent limitations, such as the resolution limit [43] and sensitivity to disconnected components [184], respectively, which hinder their applicability to many real-world scenarios.

Such challenges become even more pronounced in the dynamic setting, where communities can grow *or* shrink, merge *and* split, or even vanish (and resurge) partially or entirely at a later stage, either randomly or in a coordinated manner [38, 174]. Oftentimes, the objective may include tracking how these groups behaved or how their

attributes changed over time, in which case a model may be required to identify and preserve coherent community assignments throughout the network's temporal *evolution*. This is often done in an ad hoc manner, e.g., by means of set similarity (Jaccard, Dice, or intersection-over-union) [38], optimization heuristics [153], or mutual information [39] — introducing additional complexity to the study of their dynamics, especially in more fine-grained temporal scales. Modeling a network as discrete-time graph snapshots, or as continuous-time interaction events, leads to distinct algorithmic solutions and methodological hardships [102] — in the former, intervals may be chosen based on external knowledge, but arbitrary choices of window size and alignment must be made; in the latter, event-based representations avoid arbitrary discretization, but may result in less intuitive visualization and interpretation than snapshot-based representations, and require more sophisticated techniques to handle the continuous nature of the data.

In both modeling choices, it is clear that the temporal domain introduces additional analytical hurdles — for instance, to consistently identify and track communities, especially when changes occur at different paces. This is further compounded in graphs where both topology and attributes are dynamically defined, requiring models to effectively integrate structural, temporal, and node- and edge-level data. Model selection principles and the balance between performance, scalability, and computational costs remain central challenges when dealing with large networks — the latter causing models to fall short in practical applications if unaddressed [100], and the former leading to theoretical inadequacy and analytical inconsistency if ignored [135]. This is especially important for high-risk and critical applications, where interpretability and explainability are paramount, and measuring uncertainty and robustness is crucial for drawing reliable conclusions from data and supporting informed decision-making processes [61, 132].

From network theory's standpoint, then, there is 'no free lunch' [131] when it comes to community detection, as is true for many optimization problems [189]. In other words, there is no single method that can be considered universally optimal (for all domains) — as soon as one method is shown to outperform another in a specific setting, it is often at the expense of performance in other settings, making the choice of which model to employ and the metrics to evaluate it best guided by the specific characteristics of the data and the research problem at hand. Therefore, community detection is better understood as a generalizable goal that may find correspondence with solutions from different domains, fortunately resulting in a continuously enriched and interdisciplinary compendium of techniques accrued from diverse research and application needs [45].

Among these, machine learning models quickly achieved state-of-the-art performance in diverse downstream graph tasks, encouraging the exploration of their potential and expressive power [200] for unsupervised clustering. Bridging the two literatures, then — that of network science and artificial intelligence — seemed like the appropriate step forward for scientific enquiry. The defining question of this research thus becomes not merely whether and how novel neural approaches may be designed for node clustering, but under which assumptions, evaluation protocols, and computational constraints their usefulness may be meaningfully assessed with respect to established methods. In graph-theoretic terms, this amounts to asking what signed property should be assigned to the edge connecting learning on graphs and community detection: positive, if both fields benefit from their interplay; negative, if their goals are mutually exclusive; or neither, if orthogonal and unrelated in practice — a matter explored here throughout.

1.1.1 Machines learning on graphs

The technical and technological leap of the last decade has allowed for the development of sophisticated architectures and mechanisms for dealing with high-dimensional data, particularly through the use of graphics processing units (GPUs), optimized for parallelized computations and matrix operations. This has proven an effective strategy for tasks such as node regression, link prediction, and graph generation — e.g., image-guided brain disease diagnosis [203], identifying fraudulent actors in financial networks [20], and discovering molecular compounds modeled as graphs, respectively [52].

Considering these advances, specialized architectures for learning on relational, non-Euclidean data have made graph neural networks (GNNs) an avenue worthy of exploration for node clustering tasks [97]. Such models obtain parametric mappings that encode nodes, edges, or entire graphs into latent vector representations — *embeddings* — where elements with similar structural or attribute-based properties are typically mapped closer together in a real-valued space, according to an objective function. Neural communities may then be obtained either by clustering such embeddings, or by constraining the model's output dimension to a predefined number of units and producing assignment probabilities, usually through a normalized exponential function ($\mathrm{softmax}$) that maps real-valued vectors to a probability simplex [179]. Several families of methods have been proposed to capture information from relational data, including message passing, attention-based and autoencoder-based models, physics-informed approaches, and other architectural paradigms [207], as well as temporal extensions [102], which can be adapted for community detection by optimizing graph-theoretic objectives such as modularity or conductance, among other strategies [97]. In principle, these models can jointly exploit topology, attributes, and temporal dynamics, potentially improving the detection of communities when these sources of information are aligned; their applicability to temporal community detection, however, remains underexplored compared to other downstream tasks where they constitute the state of the art [100, 102].

Despite their potential, GNNs face several unique challenges. In multiple-layer 'deep' networks, propagated node representations can become indistinguishable, leading to a loss of meaningful distinctions (oversmoothing); while in graphs with long-range dependencies or bottlenecked connectivity, information from distant nodes may be compressed into fixed-size representations, limiting the model's ability to capture distant interactions effectively (oversquashing) [211]. Their highly non-convex loss landscapes [179] make training challenging, often resulting in suboptimal solutions: incorporating temporal information introduces further complexity and interpretability barriers, and commonly mechanisms such as memory, recurrent gates, or attention can increase expressivity, but do so at the cost of additional parameters and computational overhead, which may hinder scalability on large graphs. Moreover, in contrast to settings where large models can be pretrained on broad corpora and transferred across tasks, graph models for community detection are often trained in transductive or dataset-specific settings, making generalization across domains much less straightforward — and the lack of standardized benchmarks and evaluation protocols further complicates their assessment and comparison [124]. Assessing their limits remains nonetheless a crucial step to understand their applicability, informing proper methodological design choices, and identify potential areas of improvement — making the analysis of their performance a central concern of this research and motivating a broader conceptualization question.

1.1.2 Do electric sheep dream of communities?[2]

Despite the aforementioned challenges, the potential of *neural community detection* for capturing complex patterns in temporal graphs remains substantial: it provides a flexible framework for incorporating distinct information signals and domain-specific constraints, including patterns that established methods may struggle to model at scale.

Within this scope, neural community detection refers to uncovering (disjoint or overlapping) node sets through an unsupervised[3] machine learning model for relational data trained to optimize a graph-theoretic objective. Particular importance is given here to the temporal case, where the underlying system topology varies over time, and the model is required to account for such dynamics in order to produce coherent community assignments. This approach is appealing for real-world clustering tasks involving large relational datasets — as exploiting structural, attribute-based, and temporal information simultaneously may improve community detectability if the signals align [51, 112, 117].

While the lack of interpretability for high-dimensional relational data remains a shortcoming for high-risk and critical sectors, explainable-by-design architectural choices may help mitigate it. This motivated the neural strand of this research to remain encoder-agnostic: the presented architecture can be integrated with various embedding strategies, including message-passing, attention-based, autoencoder-based, physics-informed, and other architectural paradigms. It may therefore be potentially used in conjunction with interpretability-enhancing mechanisms for GNNs [202], to provide insight into the underlying community structures and their dynamics. Insofar as many resulting partitions can be interpreted through latent block-model assumptions, however, the advantages of neural strategies over principled approaches grounded in statistical theory remain unclear from a model-selection standpoint. Their appeal is instead more directly connected to scalability on large-scale attributed graphs, where differentiable objectives and GPU-oriented implementations may provide practical advantages. This also means that their gains in scientific insight and interpretability may be limited, which motivates this research to include principled approaches not only as baselines for comparison, but also as alternative solutions, briefly discussed in the next section.

1.2 Motivating advancements

The central question of this research was whether neural networks can recover ground-truth communities from dynamic graphs with node- and edge-level features as accurately as established baseline methods. The answer proved to be negative under most evaluated scenarios [145] — warranting, in turn, a more nuanced investigation into the conditions under which neural methods are useful for community detection in temporal graphs, and how their performance compares to that of algorithmic approaches — the latter particularly relevant both from efficiency and principled evaluation standpoints.

Addressing this revised goal required tackling three interconnected challenges: (i) software infrastructure for modeling and representing dynamic graphs; (ii) scalable al-

[2] Of what kind? — Artificial neural networks are often compared metaphorically to biological brains due to their ability to *learn* from data, sometimes also displaying 'hallucinatory' or 'dream-like' states.

[3] Although this research focuses on unsupervised methods, ML literature includes supervised methods with neural network architectures under the guise of community discovery, community search, &c.

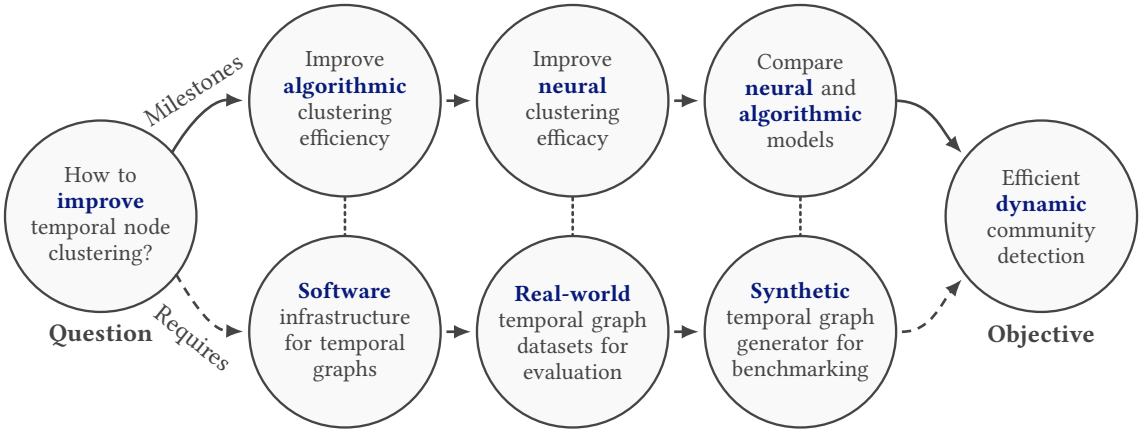


Figure 1.1: Research schematic, with methodological milestones and contributions. The upper trajectory summarizes the algorithmic and neural objectives, while the lower trajectory shows the software and evaluation infrastructure developed to support them. Revised from [148].

gorithms for community detection, together with principled benchmarks for their evaluation; and (iii) the design and implementation of a neural community detection model for continuous-time temporal graphs. Each corresponds to a self-contained part of this thesis, connected by theory and application. Their presentation order prioritizes narrative flow; nonetheless, care was taken so that each chapter remains self-contained and contextualized within preceding work, so they may be read independently. Next, each contribution is briefly outlined, and discussed in detail in the respective chapters.

1.2.1 On temporal graph structures and data

The first challenge concerned new software infrastructure for temporal graph modeling.

Initial experimentation highlighted the need for a standardized way to handle and convert datasets across diverse formats, eventually leading to the development of NetworkX-Temporal [128]: a library extending the widely used NetworkX framework [65] to dynamic graph structures. The library supports discrete- and continuous-time representations, provides functions to build, generate, slice, analyze, draw, and convert temporal graphs, and is interoperable with several graph-based libraries, enabling integration with existing workflows. It has seen encouraging adoption since its first public release, subsequently employed to support research on, e.g., information flow in physical production systems [180] and cross-species ecological interactions [14] — as well as on graph learning and downstream node clustering with time-augmented topologies, the latter of which is a direct application of this research, discussed in Section 3.2.

Alongside the library, this research extends a graph dataset derived from the PubMed citation network [113], a common benchmark in graph representation learning, by adding edge-level temporal data while preserving node-level features [146]. This allows the evaluation of community detection models to be extended to a real-world setting with temporal dynamics. It was used in the initial stages of this research to test whether augmenting static graph learning models with topological temporal information was beneficial for graph clustering, which showed limited gains in performance at higher computational cost and motivated the study of a more scalable approach.

The theoretical background on temporal graph theory supporting these contributions is presented in detail in Chapter 2. The software library and temporal graph dataset are discussed in Chapter 3, together with usage examples in two applications: a machine learning pipeline based on deep autoencoders (Section 3.2.1) and a simulation of dynamic processes in production systems (Section 3.2.2) modeled as temporal graphs.

1.2.2 On algorithmic clustering and benchmarks

The second challenge concerned the efficiency and efficacy of community detection methods for temporal graphs, together with the means for their principled evaluation.

Addressing the former, this research produced high-performance GPU implementations of spectral clustering and modularity optimization on unattributed temporal graphs. The presented methods allow but do not require coarse snapshot discretization — avoiding the need for predefined aggregation windows that may introduce information loss or analytical artifacts through arbitrary choices of window size, unless desired for specific purposes (e.g., for daily or weekly aggregation of interactions). The implementations rely on CuPy and the NVIDIA RAPIDS ecosystem, and are based on an asymptotically optimal method for community detection on graph spectra [152], implemented for single-GPU execution, and on a fast state-of-the-art greedy algorithm for supra-graph modularity optimization with multi-GPU support [111, 176], respectively.

The resulting implementations are integrated with NetworkX-Temporal through a zero-code-change interface, allowing easy switching between CPU and GPU backends. Empirically, speedups of up to three orders of magnitude over the temporal CPU-based implementation were observed, depending on graph density and number of snapshots. For large-scale networks, with millions of nodes and/or edges, this represents a change in tractability: while CPU-bound temporal modularity maximization [176] incurred prohibitive runtime costs until convergence to locally optimal solutions during tests, the GPU-based implementation completed the same task within reasonable runtime limits, enabling the clustering of large temporal graphs that were previously impractical to handle. Such implementations may support applications in recommendation systems, environmental science, trajectory and mobility analysis, and other domains where large volumes of relational temporal data must be explored efficiently [1, 57, 206]. Another envisioned application is to obtain initial node assignments for algorithmic or neural models, e.g., based on label propagation or semi-supervised learning — while still allowing community evolution to be tracked over time, since community indices are preserved across time steps as a natural consequence of constructing a temporal supra-graph [111].

Assessing such methods, in turn, calls for suitable benchmarks. The lack of appropriate benchmarks and evaluation protocols has long been a research hurdle for the evaluation of models for learning on graphs, and is particularly pronounced in dynamic settings [99]. Without standardized datasets and controlled experimental settings, it is challenging to compare the performance of different methods or assess their generalizability. This difficulty is further compounded by the scarcity of real-world temporal graphs with ground-truth structures and community-aligned attribute features [124].

To address this gap, this thesis introduces TADC-SBM [129], a time-varying, attributed, degree-corrected stochastic block model generator for synthetic graphs with community-aligned node and edge attributes. It allows the generation of synthetic

graphs with known community structures and hierarchical feature sets, where edges are sampled from a distribution parameterized by node degrees and community affiliations, and nodes are assigned feature vectors sampled from multivariate Gaussian distributions centered around community-specific means. Temporal dynamics are modeled through a Markovian process, where nodes may transition between communities over time based on their initial hidden affiliations or their current memberships. The implementation is flexible enough to accommodate network properties such as degree heterogeneity, nested structures, community stability (slowly evolving, rapidly changing, or stationary communities), and the alignment of structural, attribute-based, and temporal signals.

Chapters 4 and 5 respectively present the theory and methodological advancements here outlined. Throughout this thesis, synthetic graphs generated with TADC-SBM are used to evaluate models for community detection under controlled experimental settings, with properties commonly observed in real-world networks, such as scale-free (power-law) and Poisson-like degree distributions — providing a principled framework for assessing models for community detection and graph learning tasks, which addresses an identified lack of appropriate datasets [99, 123, 131] for model benchmarking.

1.2.3 On neural community detection

The third and final challenge concerned the design of a neural model for community detection in temporal graphs, returning to the question from which this research departed.

Toward this end, this research introduces the Longitudinal Modularity Network (LMoN), a GNN architecture that integrates structural, attribute-based, and temporal information by optimizing a recently introduced variant of temporal modularity [10]. LMoN builds on Deep Modularity Networks (DMoN), an end-to-end differentiable strategy for community detection in static graphs, while extending its objective to the temporal setting [179]. Following the same principles, the model is designed to preserve the 'best-in-class' scalability properties reported for DMoN, while integrating temporal dynamics in an encoder-agnostic manner to allow the use of various embedding strategies. Interactions are modeled as a sequence of time-stamped edges, and temporal decay is modeled by a Hawkes-inspired function [72] that reweights pairwise interaction strengths based on their recency and historical influence, allowing the model to distinguish persistent structure from transient events and obtain coherent community assignments for last observations. LMoN also optimizes a differentiable graph-theoretic objective directly, producing community assignments without an additional post-processing clustering stage, and recovers the static modularity-based objective in the absence of temporal coupling. A smoothness and regularization terms in its loss function encourage temporally stable community assignments and balanced cluster sizes during training, while a separate optimizer considers the negative log-likelihood of the event sequence under the Hawkes process, so it remains architecturally decoupled from the community-based objective for flexibility with respect to other temporal encoding strategies.

A separate contribution incorporated into LMoN is the spectral relaxation of longitudinal modularity with mean-membership link expectation, optimized end-to-end through sparse matrix operations and rank-one normalization. The original formulation [10] is algorithmic and does not provide an end-to-end differentiable learning objective; its differentiable relaxation is therefore a key step toward avoiding the post-processing

clustering stage required by many neural methods for temporal community detection. Beyond its use within the neural model, it may be of interest to other applications requiring a descriptive [135] mesoscale view of structure in continuous-time temporal graphs.

The current implementation allows multi-GPU training and relies on the TensorFlow/Keras ecosystem. Comparability is prioritized by keeping architectural and training choices close to the baseline, so that observed differences can be more directly attributed to the temporal extension. Planned extensions include a PyTorch implementation, following recent work extending DMoN to this framework [101, 154], and further experimentation with distinct encoders for more expressive embedding strategies.

The final Chapters 6 and 7 discuss the theory and methodological advancements in this neural strand. Evaluation focuses on static-versus-temporal comparison and is performed on several datasets: consistent with the no free lunch principle, its advantage is limited to cases where spatio-temporal and attribute signals align and scalability requirements dominate, rather than as a universal gain. On static graphs, LMoN recovers DMoN as a special case, and attains better or comparable performance on the temporal graphs examined, on both label-based and structural graph metrics. It therefore yields a consistent clustering objective under assumptions of assortativity [116, 205], with the added benefit of temporal coherence, which may be relevant for applications where the detection of dynamic or evolving community structures is of particular interest.

1.3 Publications

The following is a list of contributions made throughout this research, separated in journal articles, conference papers, preprints, and others (posters, talks, tutorials, datasets).

Journal articles

Passos, N.A.R.A.; Carlini, E.; Trani, S. (2025). *NetworkX-Temporal: Building, manipulating, and analyzing dynamic graph structures.* SoftwareX 31, 102277. doi: 10.1016/j.softx.2025.102277.

Merode, F. v.; Boersma, H.; Tournois, F.; Winasti, W.; Passos, N.A.R.A.; Ham, A. v.d. (2025). *Using Entropy Metrics to Analyze Information Processing Within Production Systems: The Role of Organizational Constraints.* Logistics 9(2), 46. doi: 10.3390/logistics9020046.

Conference papers

Passos, N.A.R.A.; Carlini, E.; Trani, S. (2025). *TADC-SBM: a Time-varying, Attributed, Degree-Corrected Stochastic Block Model.* In Proceedings of the 30th IEEE Symposium on Computers and Communications (ISCC '25), IEEE Computer Society, Los Alamitos, CA, USA, 1–6. doi: 10.1109/ISCC65549.2025.11326334.

Passos, N.A.R.A.; Carlini, E.; Trani, S. (2024). *Deep Community Detection in Attributed Temporal Graphs: Experimental Evaluation of Current Approaches.* In Proceedings of the 3rd Graph Neural Networking Workshop (GNNet '24), co-located with: 20th International Conference on emerging Networking EXperiments and

Technologies (CoNEXT '24), Association for Computing Machinery, New York, NY, USA, 1–6. doi: 10.1145/3694811.3697822.

Preprints and accepted papers

Passos, N.A.R.A.; Carlini, E.; Trani, S. (2026). *Learning and Clustering on Temporal Graphs: Principles, Primitives, and Pooling.* European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD '26, Nectar T.), Sep. 7–11, 2026, Naples, Italy. arXiv: 2608.03696 [cs.LG].

Passos, N.A.R.A.; Carlini, E.; Trani, S. (2026). *Accelerating Dynamic Graph Clustering on GPU Architectures with cuGraph.* Accepted at: the 6th Workshop on Flexible Resource and Application Management on the Edge (FRAME '26), co-located with: 32nd International European Conference on Parallel and Distributed Computing (Euro-Par '26), 24–28 August, 2026, Pisa, Italy. arXiv: 2608.03695 [cs.DC].

Passos, N.A.R.A.; Carlini, E.; Trani, S. (2026). *Toward Efficient Community Detection on Time-Varying Attributed Graphs.* Accepted at: PhD Symposium at the 32nd International European Conference on Parallel and Distributed Computing (Euro-Par '26), 24–28 August, 2026, Pisa, Italy. Preprint at repository: `https://nelsonaloysio.github.io/files/preprint/europar2026symposium.pdf`.

Caligiuri, A.; Emery, E.; Ferreira, L.; García-Castillo, J.; Lindner, S.D.; Molina-Hernández, J.; Passos, N.A.R.A.; Ribeiro, V.H.; Sartore, M.; Wang, B.; Calleja-Solanas, V. (2025). *Time-varying ecological interactions characterise equilibrium and stability* Preprint at arXiv: 2506.22123 [q-bio.PE] (To be submitted to a journal).

Other contributions

Passos, N.A.R.A. (2026). *Dynamic Graph Clustering with Graph Neural Networks.* Poster at Learning on Graphs (LoG) Italian Meetup, Jun. 9–11, 2026, Pisa, Italy.

Passos, N.A.R.A. (2025). *Efficient 'Neural' Community Detection on Temporal Graphs: Challenges, Advancements and Opportunities.* Poster at Future Artificial Intelligence Research General Conference (FAIR-GC '25), Dec. 10–12, 2025, Rome, Italy.

Passos, N.A.R.A. (2025). *Manipulating temporal networks using NetworkX-Temporal.* Tutorial at Complexity 72h Workshop, Jun. 24, 2025, Madrid, Spain.

Passos, N.A.R.A. (2025). *Benchmarking Algorithmic and Neural Approaches for Temporal Community Detection.* Lightning Talk at International School and Conference on Network Science (NetSci '25), Jun. 2–6, 2025, Maastricht, Netherlands.

Passos, N.A.R.A.; Carlini, E.; Trani, S. (2024). *PubMed-Temporal: A temporal graph dataset with node-level features.* Zenodo, Dec. 9, 2024, Genève, Switzerland.

Passos, N.A.R.A. (2024). *Deep Community Detection on Attributed Dynamic Graphs with a Spatio-Temporal Graph Neural Network.* Poster at International School and Conference on Network Science (NetSciX '24), Jan. 22–25, 2024, Venice, Italy.

Part I
Temporal Networks

Oh, would that my mind could let fall its
dead ideas, as the tree does its withered leaves.

André Gide, apud Judea Pearl (1988)

Chapter 2

Temporal Graphs

A dynamic or temporal graph $\mathcal{G} \coloneqq (\mathcal{V}, \mathcal{E})$ consists of temporal node and edge sets $\mathcal{V}$ and $\mathcal{E}$, whose elements may be associated with static or dynamic attributes [75]. Formally,

$$\mathcal{V} \coloneqq \{\ (v, x_v)_t \mid v \in V,\ x_v(t) \in \mathbb{R}^{d_v},\ t \in \mathbb{R}^+\ \}, \tag{2.1a}$$

$$\mathcal{E} \coloneqq \{\ (v_i, v_j, x_e)_t \mid v_i, v_j \in V,\ x_e(t) \in \mathbb{R}^{d_e},\ t \in \mathbb{R}^+\ \}, \tag{2.1b}$$

where v_i and v_j are source and target node identities, and t is the interaction time within the temporal domain $\mathbb{R}$, generally assumed as non-negative. The optional node and edge attributes x_v and x_e may be time-dependent, with dimensions d_v and d_e, respectively.

This broad definition includes various types of temporal graphs, such as contact-sequence graphs, where edges represent instantaneous interactions at specific timestamps, and interval graphs, where edges carry attributes like travel time along with explicit start and end times [79, 106]. The order $|\mathcal{V}|$ and size $|\mathcal{E}|$ of a temporal graph correspond to the cardinality of temporal node and edge occurrences over the entire temporal domain. Unless otherwise specified, the degree of a node is here understood in its aggregated sense, i.e., as the total number of its incident connections across time.

Edges in a graph may be directed or undirected — in the former case, $(v_i, v_j, x_e) \in \mathcal{E}$ does not imply $(v_j, v_i, x_e) \in \mathcal{E}$. Graphs with multiple parallel edges between the same node pairs are referred to as *multigraphs*, such as those with time-stamped repeated interactions. Graphs whose nodes or edges are partitioned into subsets representing distinct types of relationships are known as *multilayer* graphs, with *multiplex* graphs as a common special case; snapshot-based temporal graphs may also be represented in this way by treating each time step as a layer. A mapping from multigraph to a multilayer representation is obtained by assigning each edge type, timestamp, or interval to a distinct layer, and vice versa; while the union of all layers is understood as the *supra-graph*.

This work focuses on graphs of the attributed and temporal kind, also referred to as *time-varying attributed graphs* [16]. Modeling them requires careful consideration of how time is represented and how it impacts the graph's structure and dynamics. The underlying data structure used to store temporal information in computer memory significantly influences data processing, algorithmic complexity, and neural architecture design. These representations are typically categorized into two broad families: snapshot-based or discrete-time and event-based or continuous-time temporal graphs.

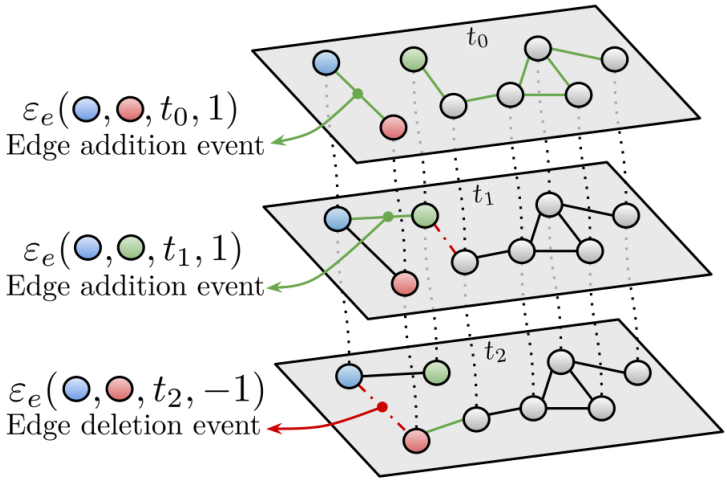


Figure 2.1: A temporal graph represented as three snapshots, with arrows indicating edge addition (in green) and deletion (in red) events. The neighborhood of the blue node changes over time, illustrating how aggregation over an interval of $\Delta t = 2$ merges distinct interaction states and induces information loss, which event-based representations such as adjacency lists avoid.

2.1 Temporal graph representations

To represent dynamic systems, it is common to distinguish between discrete-time and continuous-time models, each with its own advantages and trade-offs regarding data handling, computational efficiency, and the ability to capture temporal dynamics [85].

Although both representations aim to model evolving networks, the ways in which they encode temporal information differ, as well as the granularity with which they capture changes in the graph structure. Figure 2.1 illustrates their differences. The two methods require careful consideration of the temporal resolution, as it directly impacts the granularity of the representation and the model's ability to capture relevant temporal patterns, introducing further complexity to model design and implementation. Details and notable implications for temporal graph learning tasks are briefly discussed next.

2.1.1 Discrete-time temporal graphs

A discrete-time temporal graph (DTTG), or snapshot-based graph, represents a network as a sequence of static graphs, typically obtained by aggregating temporal events, i.e.,

$$f_d : \mathcal{G} \to \mathcal{G}_D = \{\, G_t = (V_t, E_t) \mid t \in \mathbb{N} \,\}, \tag{2.2a}$$

$$V_t = \{\, (v, x_v) \mid (v, x_v)_t \in \mathcal{V} \,\}, \tag{2.2b}$$

$$E_t = \{\, (v_i, v_j, x_e) \mid (v_i, v_j, x_e)_t \in \mathcal{E} \,\}. \tag{2.2c}$$

Each snapshot $G_t \in \mathcal{G}_D$ consists of the node and edge sets V_t and E_t, whose elements may be attributed with node-specific and edge-specific features x_v and x_e, respectively.

In this representation, time progresses in discrete steps, $t_{i+1} = t_i + 1$, where graph snapshots capture the system's state over a predefined period or aggregation interval $[t_i, t_{i+1})$. Snapshots are typically generated either by dividing the temporal domain into fixed interval windows, or by choosing an arbitrary number of bins to slice the graph.

While discrete-time representations allow for an intuitive analysis of the temporal domain, they require additional preprocessing choices to determine an appropriate binning strategy, with trade-offs [85] involving window size, event alignment, and whether intervals should be defined by fixed duration or comparable levels of node/edge activity. Fixed-duration windows may result in unevenly distributed data across intervals, while fixed-count binning may obscure temporal dynamics if bins do not align with the empirical distribution of events. This may occur, for instance, when data collection procedures or seasonal effects introduce periodic patterns that are missed by arbitrary binning.

In DTTGs, adjacency matrices are commonly used to represent connections in each snapshot, facilitating the application of traditional graph algorithms and neural network architectures for static graphs. However, this approach can lead to information loss due to the aggregation of events within fixed time intervals, which may obscure the precise timing and sequence of interactions (Figure 2.1). Despite this limitation, DTTGs remain widely used in temporal graph research, particularly in community detection [149], due to their compatibility with established methods and overall ease of implementation.

2.1.2 Continuous-time temporal graphs

A continuous-time temporal graph (CTTG), also referred to as an event-based graph, captures the precise timing of node interactions by recording individual events as they occur over time. It corresponds to the output of a function f_c mapping the temporal graph $\mathcal{G}$ to a set of time-stamped node- and edge-level events $\mathcal{E}^V$ and $\mathcal{E}^E$, respectively:

$$f_c : \mathcal{G} \to \mathcal{G}_C \coloneqq (\mathcal{E}^V, \mathcal{E}^E), \tag{2.3a}$$

$$\mathcal{E}^V \coloneqq \{\, \varepsilon_v \coloneqq (v, x_v, t_v, \delta_v) \mid v \in V,\ x_v \in \mathbb{R}^{d_v},\ t_v \in \mathbb{R}^+ \,\}, \tag{2.3b}$$

$$\mathcal{E}^E \coloneqq \{\, \varepsilon_e \coloneqq (v_i, v_j, x_e, t_e, \delta_e) \mid v_i, v_j \in V,\ x_e \in \mathbb{R}^{d_e},\ t_e \in \mathbb{R}^+ \,\}. \tag{2.3c}$$

Here, ε_v and ε_e represent node and edge events, respectively, active from start times t_v, t_e for intervals $\delta_v, \delta_e \geq 0$. Alternatively, δ may be used as a signed indicator $\delta \in \{+1, -1\}$ to denote addition $(+1)$ or deletion (-1) events, or omitted depending on the application.

A fine-grained temporal resolution allows CTTGs to capture temporal patterns that may be lost in discrete-time representations, for instance, when aggregating events into fixed intervals removes information about the order and timing of interactions [85]. When the event stream preserves all node and edge events, a mapping $f_e : \mathcal{G}_C \to \mathcal{G}$ can reconstruct the corresponding temporal graph structure without information loss.

Due to their event-based nature, this representation foregoes dense adjacency matrices in favor of interaction sequences or time-indexed adjacency lists. This enables batch processing in which computational cost scales with the number of edge-level events, $|\mathcal{E}^E|$, rather than with all (recurring) edges $\sum_{t=1}^{T} |E_t|$ (Eq. 2.2), depending on the algorithm. As a result, CTTGs offer greater scalability, particularly for large sparse networks. Such representations are increasingly found in the temporal graph learning literature [63, 151, 191], and are especially relevant for higher-order or causally informed temporal modeling tasks, such as those involving higher-order De Bruijn graphs [32, 142].

Overall, while DTTGs are a convenient and widely used representation, CTTGs provide a powerful framework for working with temporal graphs — although not without modeling and implementation challenges, as maintaining fine-grained temporal resolution increases the complexity of data handling and some algorithmic implementations.

2.1.3 Higher-order temporal graphs

Additional representations have been used to model time-varying relational data, some extending beyond dyadic graph topologies to capture higher-order (n-ary) relations.

Resembling a DTTG-like representation, supra-adjacency matrices encode temporal information by stacking per-snapshot adjacency matrices into a single data structure, with off-diagonal blocks capturing inter-snapshot connections. Although not a higher-order structure by definition, this representation allows the application of spectral methods (thoroughly discussed in the next section) and graph learning models to temporal data by encoding temporal dependencies as inter-snapshot couplings [51, 111], with applications to clustering (see Chapters 5 and 7) in algorithmic and neural contexts.

Distinct from these, De Bruijn graphs [32] encode sequences of symbols or events by representing length-k subsequences (or k-mers) as nodes, with directed edges connecting pairs of nodes depending on whether the suffix of one matches the prefix of the other over $k-1$ symbols (e.g., ABC $\rightarrow$ BCD for $k=3$ through their shared substring BC). Such representation provides an expressive graph-based way to model higher-order dependencies in temporal sequences, and has been used by graph neural networks to capture sequential patterns and infer causal relationships in temporal data [142].

Beyond dyadic structures, collective behavior in temporal data may be modeled using, e.g., simplicial complexes[1] and hypergraphs [6]. Temporal hypergraphs extend the latter by allowing hyperedges to connect multiple nodes in a time-resolved manner, providing explicit and potentially parsimonious topological encodings of group interactions that evolve over time [49]. At the same time, more recent work emphasizes that higher-order behavior should not be inferred from hypergraph parametrizations alone, since graph-based models may also define multivariate, potentially nonlinear interaction functions over node neighborhoods [136]. As in the distinction between CTTGs and DTTGs, the choice between graph-based, hypergraph-based, and related representations should be justified by the observed event structure, modeling assumptions, or empirical gains in parsimony and predictive performance [204]. Although they fall outside the scope of this thesis, these structures provide rich frameworks for modeling multi-body interactions: their study remains an active research topic in network science and machine learning [98], with applications ranging from recommendation systems to complex system modeling, and potential for further exploration in temporal graph clustering tasks.

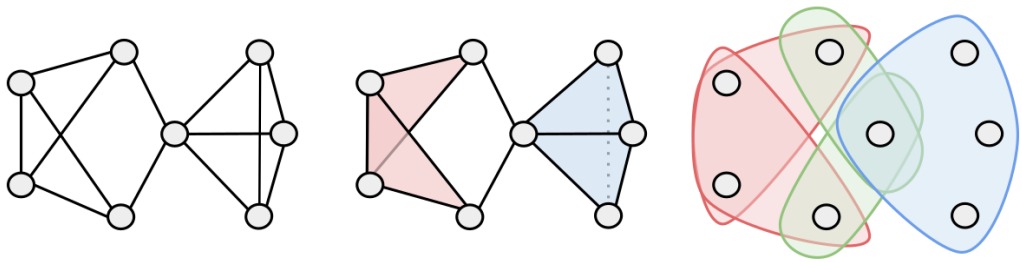


Figure 2.2: Network representation models. From left to right: a pairwise graph, a graph and its simplicial k-complexes (k=2 in red and k=3 in blue), and a hypergraph are shown.

[1] See Figure 2.2. — A simplicial complex is a collection of simplices closed under inclusion: if a simplex belongs to the complex, then all of its faces also belong to it. A k-simplex contains $k+1$ vertices. It is a stricter structure than a hypergraph, as hyperedges do not imply the presence of all lower-order faces.

2.2 Spectral graph properties

Spectral theory studies the eigenvalues and eigenvectors of matrices associated with a graph — most commonly the adjacency matrix $\mathbf{A}$ and its Laplacian $\mathbf{L}$. In static networks, these spectra underpin a broad family of clustering and embedding methods, and characterize diffusion, synchronization, and information propagation dynamics [24, 116].

For temporal networks, the time dimension may be incorporated into spectral representations in several ways. For instance, a supra-adjacency matrix may be obtained from a set of per-snapshot matrices, a common construction for temporal community detection [111]. More generally, distinct operators may be defined to encode spatial and temporal dependencies, with the spectrum of the resulting matrix providing insight into the detectability thresholds of the community structure in the temporal setting [51].

This section introduces spectral operators (Section 2.2.1), extends them to the temporal setting through supra-graph constructions (Section 2.2.2), and discusses the limits when community structure can be recovered in known sparse regimes (Section 2.2.2), providing the theoretical foundations for the clustering methods in Chapters 5 and 7.

2.2.1 Spectral operators and clustering

Let $\mathbf{A} = (A_{ij}) \in \{0,1\}^{|V|\times|V|}$ be the adjacency matrix of a simple, undirected, unweighted graph $G = (V, E)$, and let $\mathbf{D} = (D_{ij})$ be its diagonal degree matrix, where

$$D_{ij} \coloneqq \begin{cases} \sum_{v_k \in V} A_{ik} & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{and} \qquad \mathbf{L} \coloneqq \mathbf{D} - \mathbf{A}, \tag{2.4}$$

that is, the combinatorial Laplacian $\mathbf{L}$ of a graph is equal to the degree matrix minus the adjacency matrix. Note that, for directed graphs, $\mathbf{A}$ is not necessarily symmetric, and $\mathbf{D}$ may be defined from in-degrees, out-degrees, or its symmetrized version; while, for weighted graphs, entries of $\mathbf{A}$ encode pairwise interaction strengths, i.e., edge weights.

Because node degrees vary widely in real-world networks, $\mathbf{L}$ is often normalized to reduce the influence of degree heterogeneity and to bound its eigenvalues within a fixed range, avoiding the localization of eigenvectors on high-degree nodes, which generally improves the stability and performance of spectral clustering methods [24, 184].

For simple graphs without isolated nodes, the normalized Laplacian is defined as

$$\tilde{\mathbf{L}} \coloneqq \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} = \begin{cases} 1 & \text{if } i = j \text{ and } d_i \neq 0, \\ -\dfrac{1}{\sqrt{d_i d_j}} & \text{if } i \neq j \text{ and } (v_i, v_j) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

This scaling transformation can also be applied to adjacency matrices, yielding degree-normalized operators with balanced spectral properties [22, 188]. Similar definitions exist for graphs with different properties, such as directed or weighted graphs. The related random-walk Laplacian, $\mathbf{L}_{rw} = \mathbf{D}^{-1}\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$, is often used when a Markov-chain interpretation is desired [184]. For undirected graphs without isolated nodes, $\tilde{\mathbf{L}}$ and $\mathbf{L}_{rw}$ are similar matrices and have corresponding eigenvalues, with eigenvectors related by degree scaling. For spectral clustering, the resulting embeddings are often row-normalized post hoc, particularly in the presence of degree heterogeneity [184].

Clustering on graph spectra

Spectral clustering embeds nodes through the k eigenvectors associated with the smallest eigenvalues of $\tilde{\mathbf{L}}$, stacked as the rows of a matrix $\mathbf{U} \in \mathbb{R}^{n \times k}$. The rows of $\mathbf{U}$ are then typically normalized and treated as points in $\mathbb{R}^k$, where a clustering algorithm, such as k-means, groups nodes based on proximity in the embedding space [118].

Cheeger's inequality relates the second-smallest eigenvalue of $\tilde{\mathbf{L}}$ to the conductance of a graph, a measure of the edge boundary of a subset relative to its volume [24, 196]. It is closely related to the normalized cut objective, whose spectral relaxation is minimized by the same family of Laplacian eigenvectors. Through the eigendecomposition $\tilde{\mathbf{L}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, the eigenvector associated with the second-smallest eigenvalue provides a relaxed solution to the bipartitioning problem; thresholding or sweeping over this vector yields candidate cuts with low conductance, e.g., by sorting its entries and evaluating the conductance of each prefix. In contrast, spectral modularity methods use the leading eigenvectors of the modularity matrix to optimize (by maximizing) a relaxation of the modularity objective[2], yielding partitions governed by a different quality criterion.

A common strategy for spectral clustering is to minimize normalized cut [162], which favors partitions with small boundary size relative to community volume, defined as

$$\mathrm{NCut}(C_1, \ldots, C_k) = \sum_{i=1}^{k} \frac{\mathrm{cut}(C_i, \bar{C}_i)}{\mathrm{vol}(C_i)}, \quad \mathrm{cut}(C_i, \bar{C}_i) = \sum_{u \in C_i,\, v \notin C_i} A_{uv}, \tag{2.6}$$

where $\mathrm{vol}(C_i) = \sum_{u \in C_i} d_u$, that is, the sum of degrees of nodes in the community C_i, and $\mathrm{cut}(C_i, \bar{C}_i)$ is the number of edges between C_i and its complement $\bar{C}_i$.

The k smallest eigenvectors of $\tilde{\mathbf{L}}$ provide a relaxed solution to the normalized cut problem [162]. The symmetric normalized Laplacian $\tilde{\mathbf{L}}$ has its spectrum contained in $[0, 2]$, while the random-walk Laplacian $\mathbf{L}_{rw}$ shares the eigenvalues of $\tilde{\mathbf{L}}$ and admits a Markov-chain interpretation. For undirected graphs without isolated nodes, the eigenvectors of $\mathbf{L}_{rw}$ correspond, up to degree scaling, to the relaxed cluster-indicator vectors used in normalized spectral clustering [184]. Analogously, the leading eigenvectors of the modularity matrix optimize a relaxation of the modularity objective as well [116].

Minimizing the ratio cut[3] [66] instead weighs partitions by node set cardinality $|C_i|$,

$$\mathrm{RCut}(C_1, \ldots, C_k) = \sum_{i=1}^{k} \frac{\mathrm{cut}(C_i, \bar{C}_i)}{|C_i|}. \tag{2.7}$$

thereby favoring partitions that are balanced in terms of node count. Its spectral relaxation is solved using the eigenvectors of the unnormalized Laplacian $\mathbf{L}$. For graphs with substantial degree heterogeneity, normalized objectives may be preferred because they account for node volume rather than only node cardinality; equivalently, one may work with the symmetric normalized Laplacian $\tilde{\mathbf{L}}$ or the random-walk Laplacian $\mathbf{L}_{rw}$. For graphs without isolates, $\tilde{\mathbf{L}}$ and $\mathbf{L}_{rw}$ are similar matrices, $\mathbf{L}_{rw} = \mathbf{D}^{-1/2}\tilde{\mathbf{L}}\mathbf{D}^{1/2}$, and therefore share the same eigenvalues, with eigenvectors related by degree scaling [184].

[2] See Chapters 4 and 5 for discussions on spectral modularity and its temporal extensions [10, 111, 116].

[3] Sometimes referred to as a balanced cut. Note that *ratio cut* is biased toward partitions with *balanced community sizes*, while *normalized cut* is biased toward partitions with *balanced edge volume* instead.

Both ratio cut and normalized cut are non-deterministic polynomial-time (NP)-hard in their discrete forms, as many other graph partitioning problems [115]. Their continuous relaxations reduce to eigenvector problems on the unnormalized ($\mathbf{L}$) and normalized ($\tilde{\mathbf{L}}$ or $\mathbf{L}_{rw}$) Laplacians, respectively. These problems can be approximated using iterative eigensolvers, such as Lanczos or related Krylov subspace methods, which scale well to large sparse[4] graphs [184]. However, convergence depends on the relevant eigengap, e.g., $\gamma_k = |\lambda_{k+1} - \lambda_k|$, where λ_k and λ_{k+1} are consecutive eigenvalues of the Laplacian. Small eigengaps can slow convergence and make the resulting eigenspaces less stable, yielding poor approximations and suboptimal partitions. Moreover, disconnected components require additional care, since the multiplicity of the zero eigenvalue equals the number of connected components; in practice, spectral clustering is often applied separately to connected components or after preprocessing the graph accordingly [184].

Lastly, note that the same eigenbasis underlies spectral graph learning. A node signal[5] $\mathbf{x}$ can be projected to the spectral domain as $\hat{\mathbf{x}} = \mathbf{U}^\top \mathbf{x}$, so a spectral filter $g(\cdot)$ acts as

$$\mathbf{y} = \mathbf{U}\, g(\mathbf{\Lambda})\, \mathbf{U}^\top \mathbf{x}. \tag{2.8}$$

Since explicitly forming $\mathbf{U}$ is costly, practical models often approximate the filter, as in

$$g_\theta(\tilde{\mathbf{L}}) \approx \sum_{k=0}^{K} w_k\, \tilde{\mathbf{L}}^k, \tag{2.9}$$

which is K-localized and aggregates information from K-hop neighborhoods without an explicit eigendecomposition, corresponding to a degree-K polynomial of the Laplacian. Chebyshev polynomial filters [33] and Lanczos approximations [95], as well as graph convolutional networks (GCNs) [84], are examples of such localized spectral constructions. These provide part of the background for the graph neural networks revisited in Chapter 6, and extended to the attribute-aware temporal setting in Chapter 7.

2.2.2 Temporal graph spectra

For temporal (multilayer or multiplex) graphs, block-structured supra-adjacency or supra-Laplacian matrices may be assembled from per-layer adjacency matrices [27, 85].

This produces a single supra-graph with temporal couplings, enabling community detection and identity-preserving tracking within a joint decomposition [111]. Assuming a common set of n node identities across snapshots, this construction stacks the snapshot matrices on the block diagonal and links each node to its time-adjacent copies through off-diagonal identity blocks weighted by an inter-slice coupling parameter ω,

$$\mathbf{A}^{\text{supra}} = \begin{pmatrix} \mathbf{A}^{(1)} & \omega\mathbf{I} & \mathbf{0} & \cdots & \mathbf{0} \\ \omega\mathbf{I} & \mathbf{A}^{(2)} & \omega\mathbf{I} & \cdots & \mathbf{0} \\ \mathbf{0} & \omega\mathbf{I} & \mathbf{A}^{(3)} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \omega\mathbf{I} \\ \mathbf{0} & \mathbf{0} & \cdots & \omega\mathbf{I} & \mathbf{A}^{(T)} \end{pmatrix}, \tag{2.10}$$

[4] See Chapter 5 for an example with spectral modularity. — Because they require only matrix-vector products and can exploit the sparsity of $\mathbf{A}$ or $\mathbf{L}$, without computing a full eigendecomposition [184].

[5] This operation is the graph analog of Fourier-domain filtering: eigenvectors define graph frequencies, while $g(\mathbf{\Lambda})$ determines how strongly each frequency component contributes to the output signal.

where $\mathbf{A}^{(t)}$ is the adjacency matrix of snapshot t, $\mathbf{I} \in \mathbb{R}^{n\times n}$ links each node to its temporal copies, and ω controls the strength of temporal coupling between adjacent slices. The corresponding supra-Laplacian $\mathbf{L}^{\text{supra}}$ follows from the usual degree-minus-adjacency construction of $\mathbf{A}^{\text{supra}}$ — employed, e.g., for multislice modularity optimization [111].

This construction is general but costly: $\mathbf{A}^{\text{supra}}$ has dimension $nT \times nT$, so the cost of spectral decomposition grows with both the number of nodes and the number of snapshots. Although the matrix remains sparse, as nonzero entries arise only from intra-slice edges and time-adjacent inter-slice couplings, sparse storage and iterative eigensolvers are essential for scalability. However, Laplacian-based spectral methods are known to fail in networks where spatial and temporal signals are weakly correlated [51].

A principled alternative is obtained by linearizing belief propagation around the uninformative fixed point [51]. In static graphs, this leads to the non-backtracking (Hashimoto-type) operators that propagate information along directed edges while suppressing immediate backtracking (i.e., returning to the previous node) paths. The temporal extension couples spatial and temporal propagation in a block operator, yielding

$$\mathbf{B}^{\text{nb}} = \begin{pmatrix} \lambda\mathbf{A}^{(\text{s})} & -\lambda\mathbf{I} & \lambda\mathbf{A}^{(\text{s})} & \mathbf{0} \\ \lambda(\mathbf{D}^{(\text{s})} - \mathbf{I}) & \mathbf{0} & \lambda\mathbf{D}^{(\text{s})} & \mathbf{0} \\ \eta\mathbf{A}^{(\text{t})} & \mathbf{0} & \eta\mathbf{A}^{(\text{t})} & -\eta\mathbf{I} \\ \eta\mathbf{D}^{(\text{t})} & \mathbf{0} & \eta(\mathbf{D}^{(\text{t})} - \mathbf{I}) & \mathbf{0} \end{pmatrix}, \tag{2.11}$$

where each of $\mathbf{A}^{(\text{s})}$, $\mathbf{A}^{(\text{t})}$, $\mathbf{D}^{(\text{s})}$, and $\mathbf{D}^{(\text{t})}$ is an $nT\times nT$ matrix. Here, $\mathbf{A}^{(\text{s})} = \bigoplus_t \mathbf{A}^{(t)}$ is the block-diagonal intra-slice adjacency matrix, $\mathbf{A}^{(\text{t})}$ is the inter-slice adjacency matrix connecting each node to its time-adjacent copies (with block-diagonal identities weighted by η), $\mathbf{D}^{(\text{s})}$ and $\mathbf{D}^{(\text{t})}$ are the corresponding diagonal degree matrices, and $\mathbf{I}$ is the nT-dimensional identity. The scalar $\lambda = (c_{\text{in}} - c_{\text{out}})/(k \cdot c)$ is the second eigenvalue of the spatial label-transition matrix, with c_{in} and c_{out} denoting the average within- and between-community connection rates, k the number of communities, and c the average degree. It controls attenuation of community information along spatial edges, while $\eta \in [0, 1]$ controls attenuation along temporal edges by governing the stability of node labels across consecutive snapshots. Together, λ and η yield a richer temporal model than one based solely on an inter-slice coupling ω, with the effective signal-to-noise ratio $c\lambda^2$ and the temporal stability η determining the detectability transition [51].

Although the resulting matrix $\mathbf{B}^{\text{nb}}$ has dimension $4nT \times 4nT$, it is also critically sparse, since many of its blocks are degree, identity, or zero-valued matrices. The construction is intentionally asymmetric (direction-aware), and its non-backtracking structure mitigates the localization effects that affect Laplacian eigenvectors in sparse graphs, allowing its eigenvectors to remain informative closer to the detectability threshold. The $k - 1$ eigenvectors associated with the largest-magnitude eigenvalues outside the uninformative bulk can be used to obtain embeddings for temporal nodes, clustered in $\mathbb{R}^{k-1}$.

More fundamentally, Laplacian-based operators often fail in sparse regimes where an effective signal-to-noise ratio approaches a critical threshold, while non-backtracking operators remain informative closer to this limit [51]. Because this non-backtracking operator is generally non-symmetric, practical implementations often rely on symmetric alternatives with related spectral information, such as the Bethe-Hessian [152], as discussed in the contributions of Chapter 5. The next section formalizes the detectability of communities in temporal graphs and introduces the operators that attain it.

Symmetrization and the Bethe-Hessian operator

In sparse stochastic block models (SBMs), i.e., generative models with (planted) node groups whose edge probabilities depend on group memberships, there is a fundamental detectability threshold below which no algorithm can recover community structure better than chance in the large-graph limit ($n \to \infty$) [51]. In the static case, this limit is governed by the community signal $\lambda = (c_{\text{in}} - c_{\text{out}})/(k \cdot c)$, where c_{in} and c_{out} are the average within- and between-community connection rates, k is the number of communities, and c is the average degree. The corresponding effective signal-to-noise ratio is $c\lambda^2$, where λ is the second eigenvalue of the label-transition matrix; and the detectability transition occurs at the Kesten-Stigum, or Almeida-Thouless, bound $c\lambda^2 = 1$ [51].

In the temporal case, a second parameter enters: the community stability rate $\eta \in [0, 1]$, which dictates the probability with which a node retains its community label (stays in the same community) between consecutive snapshots. When η is high, temporal persistence reinforces the community signal across time; as η decreases, temporal noise weakens this persistence and increases the amount of spatial signal required for reliable recovery. When labels become temporally uncorrelated, temporal coupling no longer provides useful information beyond what can be recovered from individual snapshots. The detectability of dynamic communities therefore depends jointly on λ and η [51], the same parameters that control the synthetic benchmarks introduced in Chapter 5.

Ordinary Laplacian spectra generally do not attain this limit in sparse regimes, since their informative eigenvectors may localize around high-degree nodes before the detectability threshold is reached. Non-backtracking operators avoid this localization and propagate information along directed edges while suppressing immediate backtracking paths, and their eigenvectors remaining informative closer to the threshold. Linearizing belief propagation around the uninformative fixed point yields a non-backtracking (Hashimoto-type) operator, and in temporal graphs the corresponding block construction introduces spatial and temporal attenuation parameters governed by λ and η.

However, the non-backtracking operator is by construction non-symmetric, which complicates its numerical treatment and limits the applicability of standard eigensolvers. In the static case, a symmetric alternative is given by the Bethe-Hessian operator,

$$\mathbf{H}(r) = (r^2 - 1)\mathbf{I} - r\mathbf{A} + \mathbf{D}, \tag{2.12}$$

whose informative eigenvectors are associated with negative eigenvalues and correspond to informative directions of the non-backtracking matrix (Eq. 2.11) [152]. At $r = \sqrt{\langle c^2 \rangle / \langle c \rangle - 1} \approx \sqrt{c}$, the number of negative eigenvalues of $\mathbf{H}(r)$ estimates the number of communities [88] in SBM regimes, and the corresponding eigenvectors may be used to recover community structure if $c\lambda^2 > 1$, yielding a computationally efficient method that attains the detectability limit in sparse graphs with planted communities.

Beyond detection, inferential temporal block models can estimate or control the parameters governing community signal and temporal stability, thereby generating temporal graphs with planted communities that evolve at prescribed rates. This principle underlies the synthetic benchmarks used throughout this thesis. Such benchmarks make the detectability transition empirically observable: across the experiments reported in Chapters 5 and 7, well-tuned spectral and inferential methods were not consistently outperformed by neural models on real-world or synthetic temporal graphs [129, 145] — thereby motivating efficiency, rather than accuracy alone, as a central design objective.

Chapter 3

Dynamic Graph Modeling

In real-world networks, interactions between entities are rarely fixed — rather, they undergo continuous changes as the system *functions*, due to distinct temporal *dynamics*.

For instance, relationships in a social network form and dissolve over time; information flow in communication networks fluctuates depending on time of day and day of week; and biological responses vary depending on endogenous and exogenous mechanisms, such as circadian rhythms and environmental factors. This added complexity poses significant analytical challenges, as such systems may exhibit intricate and non-trivial patterns that are not adequately captured by static graph representations — and so are often modeled as *dynamic graphs*, with time-varying node and edge elements [16].

Driven by the increasing availability of large-scale temporal relational data, as well as by the development of new algorithms and computational tools, research on dynamic graphs has gained traction in recent years [75, 76]. Yet software support has remained limited and fragmented [96]: datasets were scarce, standardized benchmarks for model evaluation were limited, and algorithm implementations were often scattered across repositories and formats, typically as isolated pieces of code when publicly available. This fragmentation demands substantial expertise from users and hinders adoption outside specialized research settings, particularly in the absence of a common framework for building, transforming, analyzing, and visualizing temporal networks. In the context of this thesis, these challenges motivated the development of an integrated software toolset for dynamic graph modeling that interfaces with established graph analytics workflows.

This chapter presents NetworkX-Temporal [128], a software library for the creation, manipulation, analysis, and visualization of dynamic graphs. It extends the widely used NetworkX framework [65] to support time-evolving systems. Alongside the software, the chapter introduces PubMed-Temporal [146], a graph dataset representing scientific publications and citations over time, with node-level attributes and time-stamped edges. Usage examples illustrate the library's main functionalities, including slicing and visualizing relational data over time, transforming graphs between temporal representations and data formats, and computing temporal metrics and properties, such as graph centralization [46]. Two applications are then discussed: an initial experiment employing NetworkX-Temporal on the preprocessing stage of a machine learning model, and its use in a co-authored study of dynamic processes in production systems modeled as graphs [180]. This chapter lays the groundwork for subsequent contributions, including high-performance implementations of spectral clustering and modularity optimization algorithms for temporal graphs, integrated into the library and introduced in later chapters.

3.1 Software advancements for temporal networks

This thesis advances dynamic graph modeling through the development of free open-source software [128] and the extension of a real-world dataset to the temporal domain [146]. The next sections present both contributions in detail, followed by an application of each to deep learning on graphs and the study of real-world system dynamics [180].

3.1.1 The NetworkX-Temporal software library[1]

NetworkX-Temporal is a Python package that extends the popular NetworkX library to dynamic graphs, enabling the modeling and analysis of time-evolving complex systems.

Network science provides a robust framework for representing and analyzing relational systems across domains, from communication and transportation infrastructures to biological components and chemical interactions [52, 64]. In these representations, entities are modeled as nodes and their interactions as edges; however, most real-world systems are not static, and exhibit temporal dynamics such as the dissolution, formation, and recurrence of connections [17]. While traditional network analysis has primarily focused on static graphs, there is a growing recognition of the need to incorporate temporal dynamics to more accurately model these systems — which, in turn, demands the development of specialized tools and methodologies [75].

To address the practical challenges posed by these systems for their computational analysis, especially in the context of this thesis, the software presented here took shape.

Module	Description
`algorithms`	Implements algorithms and metrics adapted for dynamic systems, such as temporal centrality, graph centralization, and multislice modularity optimization.
`classes`	Defines core temporal graph structures, supporting directed/undirected edges and multiple pairwise interactions, inheriting from main NetworkX classes.
`drawing`	Tools for drawing and visualizing temporal graphs and dynamics, leveraging Matplotlib and custom layouts for, e.g., unrolled multislice representations.
`generators`	Temporal graph datasets, including generators based on stochastic block models for controlled simulations. Also provides loaders for real-world graphs.
`readwrite`	Exposes NetworkX-compatible I/O functions to import and export compressed graph snapshots in standard formats, e.g., CSV, GraphML, GEXF, and JSON.
`transform`	Provides conversion functions between discrete-time snapshots, continuous-time events, and unrolled multislice representations of temporal graphs.
`typing`	Package-specific type hints to ensure consistent function signatures and improve developer workflow, e.g., for static and temporal graph objects.
`utils`	Miscellaneous helper functions for data manipulation and masking. Also implements conversion functions for external graph and ML frameworks.

Table 3.1: Overview of modules in the NetworkX-Temporal software library.

[1]Part of the content in this section was published in: *NetworkX-Temporal: Building, manipulating, and analyzing dynamic graph structures.* Passos, N.A.R.A.; Carlini, E.; Trani, S. (2025). SoftwareX 31, 102277.

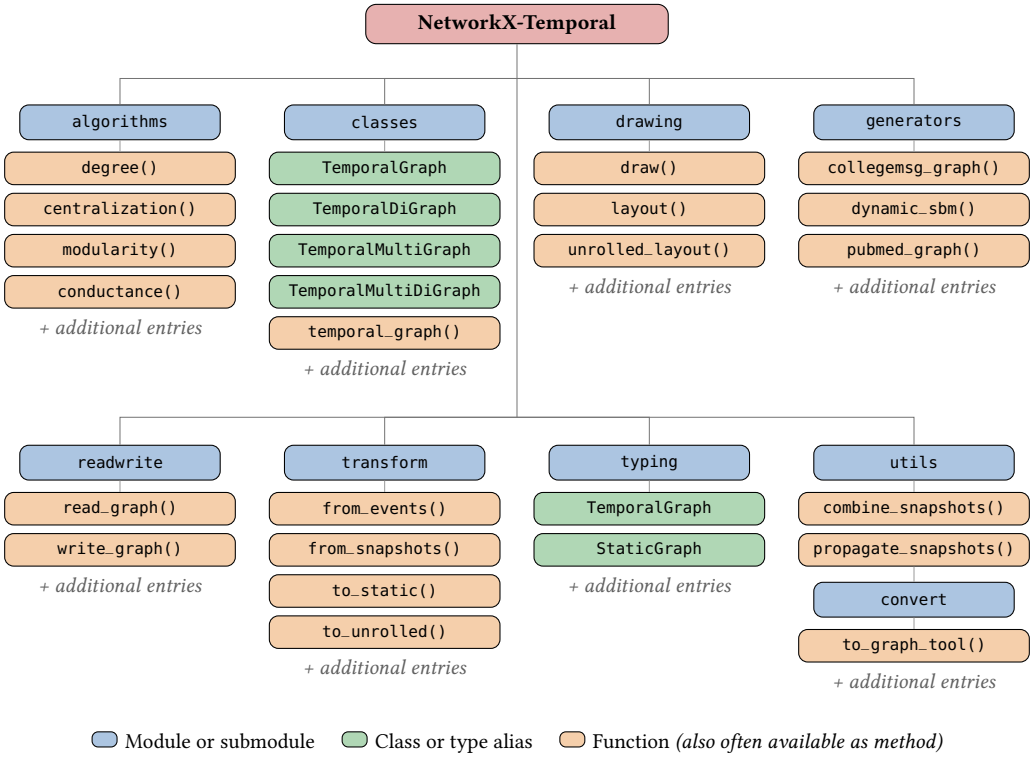


Figure 3.1: Software architecture diagram. For brevity, only some classes and functions exposed on package import are shown. Full documentation is available online. Revised from [128].

The data structures and algorithms implemented in NetworkX-Temporal support the representation, manipulation, and study of dynamic graphs as native objects, including discrete-time snapshots, continuous-time events, and 'unrolled' multislice representations, as well as utility functions that facilitate interoperability with other libraries in the Python ecosystem [96]. Developed as a modular library with an emphasis on extensibility, it adheres to NetworkX's API conventions [65] and acts as a bridge between the well-established framework and the emerging needs of temporal network analysis.

A recent review on software support for temporal networks [96] found that, while several solutions are available, overall support remains limited and fragmented, far from consolidated around a single library. The review identified a lack of tools integrating data handling, algorithm implementations, and visualization, with most reviewed options tailored to specific tasks. In that context, an early version of NetworkX-Temporal was found to display 'balanced performance with no significant drawbacks' and was described as a 'reliable general-purpose choice'. It has since been extended to support additional functionalities, including graph simulation by stochastic block models [129], community detection by spectral and modularity-based methods [147], and loaders for existing and novel temporal graph datasets [146]. Moreover, it was also employed in a study of production systems logistics, where manufacturing and healthcare workflows were modeled as dynamic graphs and analyzed through graph-entropy measures [180].

The software architecture and its main functionalities are discussed next. The library is free and open source, distributed[2] through the Python Package Index (PyPI). At

[2] Available at: `https://pypi.org/p/networkx-temporal`. Distributed under the BSD-3-Clause License.

the time of writing, an average[3] of approximately 2,000 monthly package downloads is reported. Its source code is publicly available, accompanied by online documentation, tutorials, and usage examples with references to their respective implementations.

Software architecture and functionalities

Following object-oriented development principles, the software is organized into modules that encapsulate related functionalities, promoting code reusability and maintainability. The module layout follows the organizational structure of NetworkX, while extending it with temporal graph classes, algorithms, dataset loaders, and interoperability utilities. Table 3.1 provides a list of each module and a high-level description of their functionalities, and Figure 3.1 provides a tree-like overview of the code structure.

Algorithms — Temporal metrics and properties

Enables computing topological properties of dynamic graphs, including centrality, centralization, and conductance, and implements temporal community detection methods.

The `algorithms` module represents the main analytical component of the software, providing functions to quantify node- and graph-level topological properties as the network evolves over time, with particular emphasis on dynamic community detection. Spectral clustering techniques and greedy optimization of multislice modularity [111] are implemented employing supra-graph representations (Equation 2.10), providing a way to reduce information loss from static aggregation and supporting temporally consistent community assignments. Execution is supported on both CPU and CUDA-enabled GPU hardware, with multi-GPU parallelization available for greedy optimization, enabling the analysis of large-scale temporal graph data. Device selection is automated by passing a `backend` argument to the supported functions, or by setting a global environment variable, allowing to seamlessly switch between CPU and GPU execution. Figure 3.2 (next page) illustrates the contrast between static aggregation, isolated snapshot analysis, and temporal representations in the context of modularity optimization.

Lastly, as temporal graphs may be sliced into static subgraphs, NetworkX algorithms for static graphs may be applied on a per-snapshot basis and compared against aggregated periods of interest. The module also provides graph-level metrics such as centralization [46], spectral modularity scores for evaluating mixed-membership partitions, and selected node-level measures not available in NetworkX, including bridgeness and brokering centrality [13, 78]. These measures help quantify the role of nodes in connecting communities and mediating information flow across temporal networks.

Classes — Core temporal graph structures

Defines the core temporal graph structures of the library, extending NetworkX classes to support directed, undirected, and multigraph representations with time-stamped edges.

The `classes` module establishes the data-model foundation of NetworkX-Temporal by extending the static graph classes provided by NetworkX. This design preserves compatibility with established graph-theoretical functionality while introducing abstractions for

[3]Source: https://clickpy.clickhouse.com/dashboard/networkx-temporal. Accessed on: 2026-06-01.

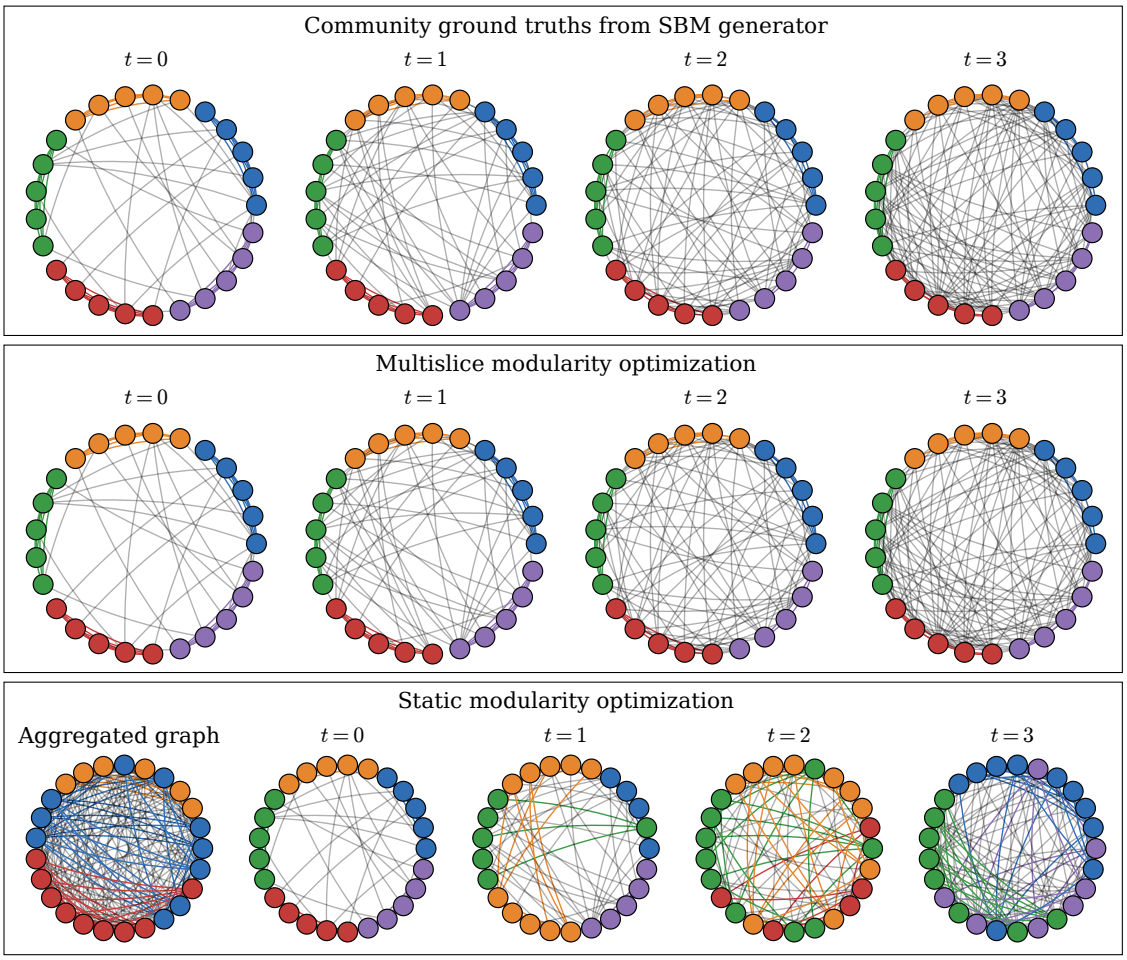


Figure 3.2: Community detection on synthetic temporal graphs with decreasing assortativeness. Rows show planted SBM communities [74], Leiden optimization of multislice modularity [111, 176], or of static modularity on the aggregated graph or single snapshots. Node positions are fixed; node colors indicate communities, label-aligned for comparison with ground truths [86]; colored/gray edges denote intra- and inter-community edges, respectively. Revised from [128].

time-varying relational data. Temporal graph objects therefore retain the familiar NetworkX interface for adding, removing, and modifying nodes, edges, and attributes, while enabling memory-efficient snapshot slicing through subgraph views whenever data duplication is not required, i.e., when attributes are not dynamically updated. Following NetworkX nomenclature, the module defines four primary temporal graph classes, covering directed and undirected graphs with either single or multiple pairwise interactions.

The module supports the temporal representations introduced in Chapter 2. Discrete-time temporal graphs are represented as ordered sequences of static subgraphs (Equation 2.2), whereas continuous-time temporal graphs are represented as sets of time-stamped edge events, optionally associated with an interval parameter (Equation 2.3). The former is convenient when the analysis naturally depends on snapshots, repeated observations, or compatibility with static graph algorithms; the latter is preferable when interactions are irregular, high-frequency, or more compactly expressed as events. A third, 'unrolled' multislice representation stores temporal evolution in a single graph object by introducing time-adjacent node copies and inter-slice couplings, making it

suitable for methods based on supra-adjacency constructions and temporal community detection. Figure 3.3 compares these representations. Core operations needed to move among these structures, including combining snapshots, propagating nodes or edges across time, are available through the `transform` and `utils` modules, discussed next.

Drawing — Visualization of temporal graphs

Provides tools for visualizing temporal graphs and their dynamics, leveraging Matplotlib [77] and implementing helper functions for, e.g., unrolled multislice representations.

An important component of network analysis is visualization, particularly when temporal dynamics introduce additional complexity not easily captured by numerical metrics alone. The `drawing` module extends NetworkX-style visualization routines to temporal graphs, allowing users to render static snapshots, aggregated intervals, and unified representations with different layouts and visual encodings. Node positions may be computed using, e.g., force-directed layout algorithms, or provided manually to preserve spatial consistency across time steps. Matplotlib [77] is used as the native rendering backend, enabling fine-grained customization of node and edge colors, sizes, labels, and other graphical attributes. These routines are intended primarily for exploratory

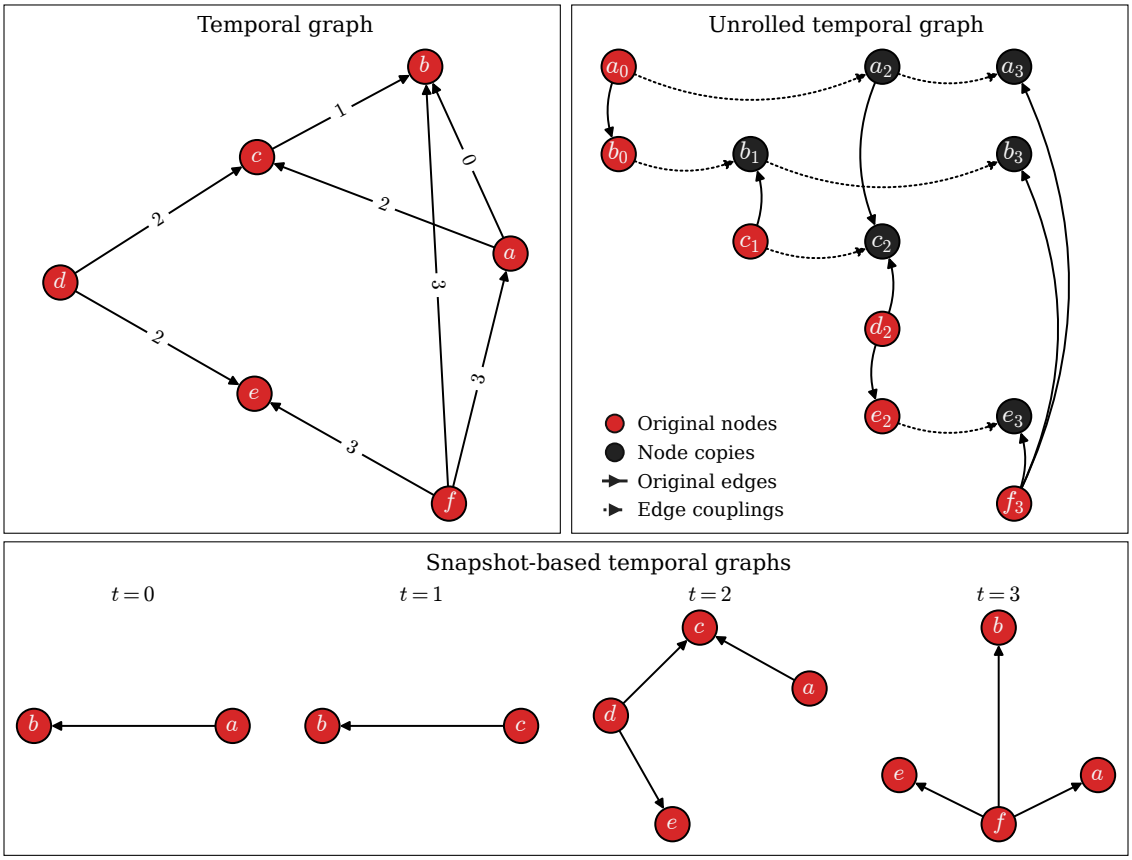


Figure 3.3: Temporal graph representations. A directed graph with 6 nodes and 8 time-stamped edges is shown as a temporal graph (top left), an unrolled temporal graph (top right), and a snapshot-based temporal graph (bottom). In the unrolled representation, dotted edges denote inter-slice couplings added between temporally ordered node copies. All representations are created from the same temporal graph data and drawn with the software. Revised from [128].

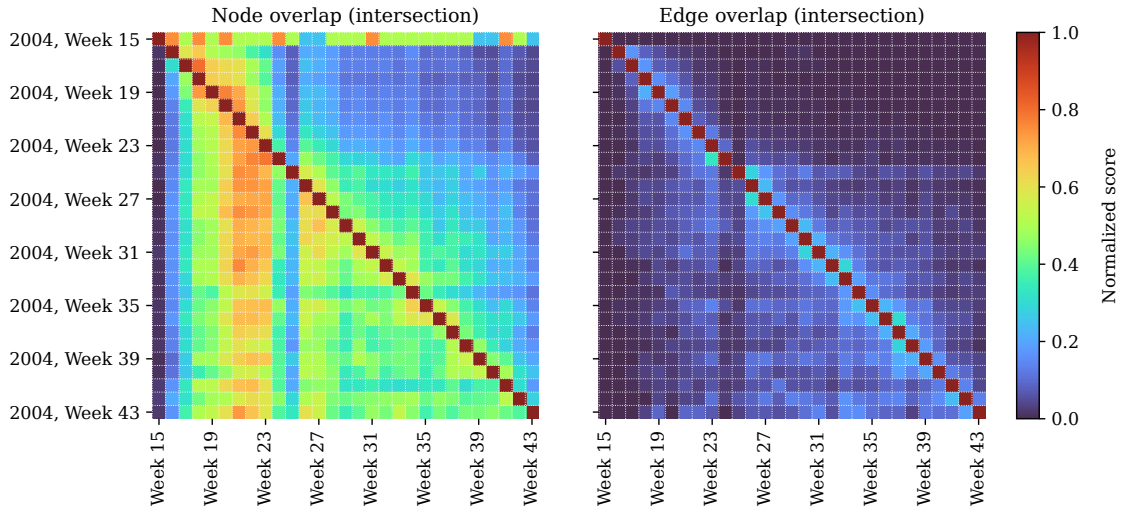


Figure 3.4: Node and edge overlap matrices for the CollegeMsg dataset [125], sliced with NetworkX-Temporal into 29 weekly snapshots. Entries report pairwise snapshot intersections normalized by the size of the first set; larger values indicate stronger node or edge persistence.

analysis and for producing compact illustrations of temporal structure, such as the aggregation of interactions within intervals or the filtering of subgraphs by activity.

However, visualizing larger temporal graphs remains challenging, as layout choices, temporal discretization, and visual encodings can substantially affect interpretability. Although subgraph filtering may improve rendering performance, identifying useful visual configurations remains an iterative process that often requires domain knowledge. For interactive exploration, specialized platforms such as Gephi [5] and Cytoscape [160] offer more advanced capabilities, including limited support for temporal networks through plugin extensions [140] and automated workflow execution with Python [122]. A comparison among these tools for general network visualization is provided in [130].

Alternatively, external libraries such as graph-tool [133] implements efficient routines for community-centric visualization, while Teneto [175] provides additional support for temporal graph structures. General-purpose solutions such as HoloViews and Datashader may also be used to construct more scalable visualization routines [58, 169], albeit requiring custom data parsing and pipeline implementation. NetworkX-Temporal therefore supports such workflows by exporting generated snapshots in standard graph formats and providing conversion functions for external libraries, enabling its use as a modeling and preprocessing layer before exploratory analysis with external software.

Generators — Artificial and real-world datasets

Creates synthetic graphs with temporal dynamics for controlled simulations and provides loaders for real-world temporal graph datasets included with the library.

The `generators` module supports two complementary needs in temporal graph analysis: controlled simulation and access to empirical datasets. For synthetic data, the module implements model-based generators, including a temporal extension of the degree-corrected stochastic block model (DC-SBM, Equation 4.17) — allowing users to create dynamic graphs with prescribed community structure, heterogeneous degree distribu-

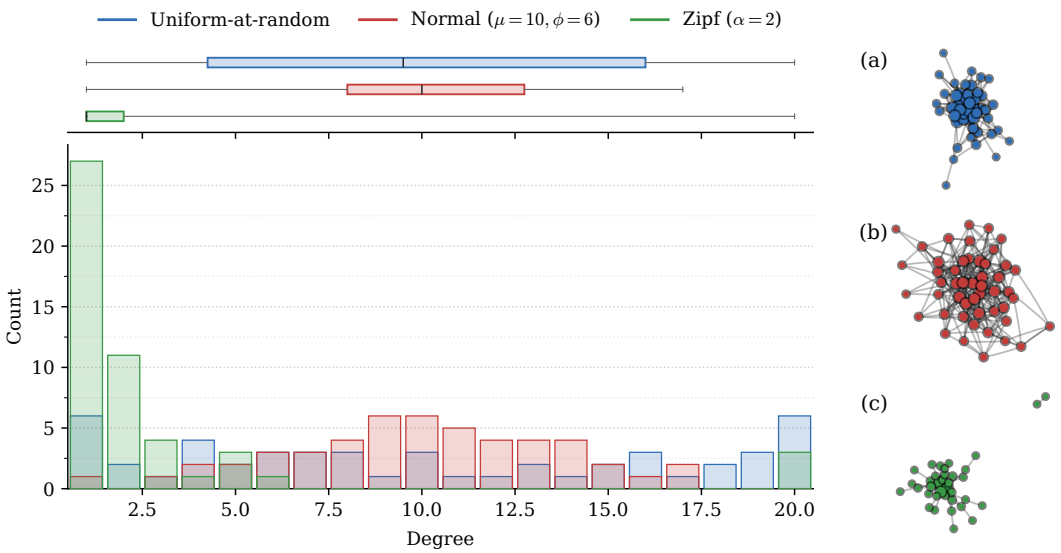


Figure 3.5: Empirical degree vectors sampled from uniform (i.i.d.), Gaussian (normal), and Zipf (discrete power-law) mechanisms. The upper panel summarizes each vector by min-max whiskers, interquartile range, and median, while the histogram reports sampled degree counts. Panels (a), (b), and (c) show graph realizations generated from each vector with random wiring.

tions, and node-level transitions over time, providing controlled settings for algorithm evaluation and methodological development. Such functionality is particularly relevant in the context of this thesis, where synthetic benchmarks are later used to evaluate temporal community detection methods under known ground truths and varying temporal stability; the attributed extension of this model is introduced in Chapter 4. Figure 3.2 illustrates one such use case, comparing ground-truth communities with those detected through static and temporal modularity optimization. The module also includes degree-vector generators, as illustrated in Figure 3.5 for graphs drawn with the library from uniform, Gaussian, and Zipf distributions, using the Fruchterman–Reingold layout [47].

In addition to synthetic graph generation, the module includes loaders for real-world temporal datasets, including communication, citation, and social interaction networks. These loaders return temporal graph objects directly compatible with the rest of the library, allowing datasets to be sliced into snapshots, transformed between temporal representations, or converted to external graph libraries. For example, the CollegeMsg dataset [125], consisting of private message exchanges between users of an online social network at a U.S. university, is included as a built-in loader. The module also provides access to PubMed-Temporal [146], an extended version of the widely used dataset for evaluation benchmarks [113, 197] with time-stamped edges, introduced in Section 3.1.2. By following the same interface, additional datasets can be incorporated in the library, making it a reusable entry point for diverse needs in temporal graph analysis.

Read and write — Input/output operations

Handles data persistence by importing and exporting temporal graphs in standard formats, including CSV, GraphML, GEXF, and JSON, with support for file compression.

The `readwrite` module provides the main interface for saving and loading temporal

graph objects to/from disk and buffered streams. It extends NetworkX internal functions, allowing time-varying relational data to be exchanged with external analysis tools through established file formats. This functionality enables reproducible workflows by making temporal graphs and resulting snapshots serializable, portable, and shareable.

A distinctive feature of the module is its support for compressed archives. Instead of storing each temporal snapshot as a separate file, a sequence of snapshots can be stored as a single file using ZIP, BZIP2, or LZMA compression, reducing storage overhead and simplifying dataset management. Within these archives, each snapshot is preserved as an individual graph object, retaining node-, edge-, and graph-level metadata when supported by the chosen file format. The compression level may be adjusted to balance storage efficiency and processing speed, and archived temporal graphs may be loaded back into the library with their snapshot structure preserved, in order to standardize the exchange of temporal graph datasets across software environments and workflows.

The module additionally integrates support to HIF (Hierarchical Interchange Format) [25], a JSON-based format for hypergraph data, and GEXF (Graph Data Format), a legacy, but highly compressible tabular format supported by external tools such as Gephi [5].

Transform — Temporal graph structures and representations

The `transform` module provides functions to reconfigure temporal graphs among snapshot-based, event-based, static, and unrolled representations, as seen in Figure 3.3.

Unlike conversions to external library formats (Table 3.3), these transformations modify the object data structures, affecting the graph's order, size, attribute semantics, and implementation logic of downstream algorithms. Table 3.2 summarizes the supported representations by order, size, and support for dynamic node and edge attributes.

Snapshot-based representations model a temporal graph as an ordered sequence of static graphs, whereas event-based representations store interactions as time-stamped events, optionally with duration or edge-addition/removal semantics (Equations 2.2 and 2.3). Static aggregation collapses the temporal graph into a single object with time-stamped edges, while the unrolled representation stores network evolution by adding time-indexed node copies and inter-slice edges coupling them. These representations are interconvertible, and the appropriate choice depends on the requirements of the analysis or algorithm: snapshot-based representations, stored as NetworkX subgraph views, are intuitive and compatible with static graph algorithms, whereas event-based representations can be more compact and better suited to continuous-time data.

	Order	Size	Dynamic node attributes	Dynamic edge attributes
Static graph	$V = V$	$E = E_T$		✓
Snapshot-based	$V \geq V_T$	$E = E_T$	✓	✓
Event-based	$V = V_T$	$E = E_T$	✓	✓
Unrolled graph	$V \geq V_T$	$E \geq E_T$	✓	✓

Table 3.2: Temporal graph representations implemented in the `transform` module. Expected order and size changes is indicated, along with support for storing dynamic node and edge attributes. By default, snapshots (NetworkX subgraph views) are internally used to store data.

Library	$\mathcal{G}$	Parameter	Description
cuGraph [119]		`'cugraph'`	Graph suite from NVIDIA RAPIDS.
CuPy [121]		`'cupy'`	Sparse GPU array representation.
Deep Graph Library [186]		`'dgl'`	Deep learning on graphs.
DyNetX [150]	✓	`'dynetx'`	Dynamic network analysis.
graph-tool [133]		`'graph_tool'`	High-performance graph operations.
igraph [28]		`'igraph'`	General-purpose network analysis.
NetworKit [167]		`'networkit'`	Scalable graph algorithms.
NumPy [71]		`'numpy'`	Dense vector representations.
PyTorch Geometric [40]		`'torch_geometric'`	Deep learning on graphs.
SciPy [183]		`'scipy'`	Sparse matrix representations.
SNAP [91]		`'snap'`	Large-scale network analysis.
StellarGraph [31]		`'stellargraph'`	Machine learning on graphs.
Teneto [175]	✓	`'teneto'`	Temporal network analysis.

Table 3.3: Conversion functions for external libraries in the `utils` module. Column $\mathcal{G}$ indicates whether temporal graph structures are natively supported by the library; if not, conversion produces supra-graphs or static (snapshot-based) graph representations with time-stamped edges.

Together, these transformations provide the preprocessing layer required by temporal graph algorithms and machine learning pipelines with representation-specific inputs.

Typing — Package-specific type definitions

Defines package-specific type aliases for static and temporal graph objects, as well as Matplotlib figure and axis objects, to improve codebase maintainability and readability.

The `typing` module centralizes type definitions used across the library, providing consistent annotations for temporal graph objects, accepted input formats, and common return types. These aliases enable explicit annotation of function signatures, both in the source code and in the API documentation generated by Sphinx. They also support modern Python development by making type expectations easier to check and maintain as the codebase evolves, reducing the likelihood of runtime errors due to type mismatches.

Utility functions — Data manipulation and conversion

Provides helper functions for temporal data manipulation, masking, similarity computation, and conversion to external graph analysis and machine learning libraries.

The `utils` module provides auxiliary routines for manipulating temporal graph data and converting it to external libraries and frameworks. Temporal node and edge indices may be obtained and subgraphs filtered according to different criteria, while additional utilities support combining and propagating snapshots over time, node- and edge-level mapping, and matrix-based comparisons of temporal activity. Measures such as intersection-over-union (Jaccard) and Dice coefficients quantify overlap across inter-

vals based on either temporal node or edge sets, facilitating the analysis of persistence, turnover, and structural similarity in dynamic graphs. Figure 3.4 displays such an example for the CollegeMsg graph [125], where node/edge overlap considers the intersection of elements normalized by the size of the first set (horizontal axis), i.e., $|A \cap B|/|A|$.

Additional functions in the `convert` submodule support interoperability with external graph libraries and machine learning frameworks, as summarized in Table 3.3. For frameworks that do not natively support temporal relational data, these functions generate compatible representations, such as snapshot sequences or aggregated graphs with time-stamped edges, depending on the current graph representation. External dependencies are dynamically imported when the corresponding functions are called to keep the core library lightweight while allowing NetworkX-Temporal to be used as a modeling and preprocessing layer for downstream analysis tasks with specialized libraries.

3.1.2 The PubMed-Temporal graph dataset

The PubMed-Temporal dataset [146] extends the widely used PubMed citation network [113, 197] with edge-level timestamps and chronological train/validation/test splits.

Extending the PubMed graph dataset to the temporal domain

The dataset was constructed[4] by augmenting the PubMed citation graph with publication-year metadata obtained from the PubMed database. Nodes represent scien-

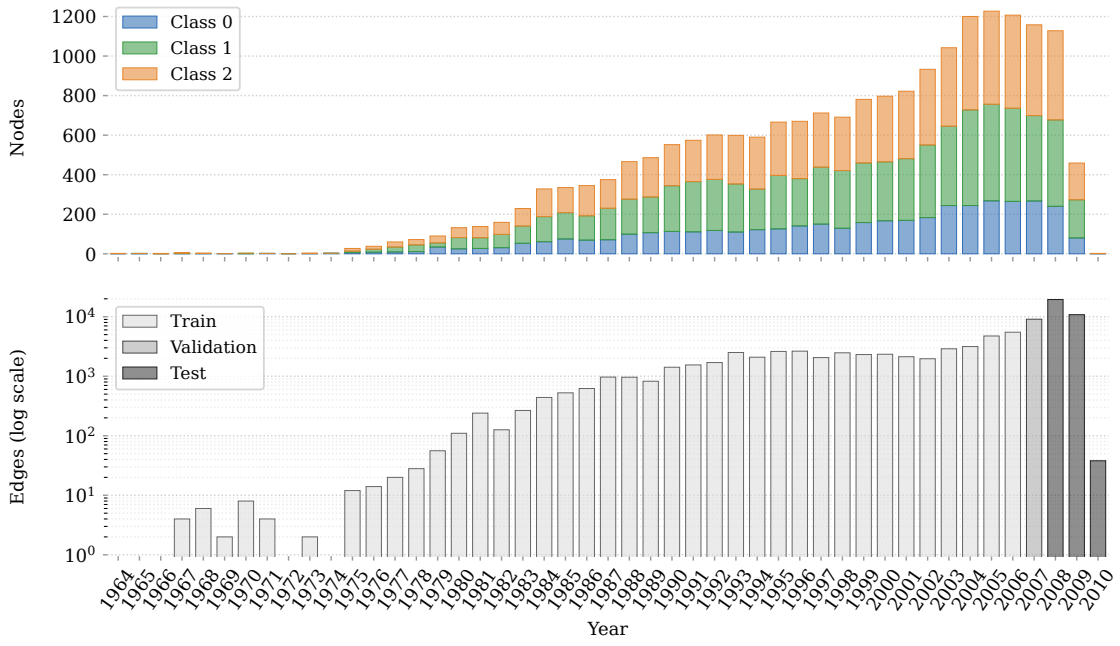


Figure 3.6: PubMed-Temporal graph characteristics. Top: number of active nodes (publications) per class. Bottom (log-scaled): number of edges in time-respecting train, validation, and test sets. Note that some years (1964, 1965, 1966, 1972, 1974) show no citation (edge) activity in the dataset.

[4]Capsule available for reproducibility: *PubMed-Temporal: A temporal graph dataset with node-level features.* Passos, N.A.R.A.; Carlini, E.; Trani, S. (2024). Zenodo. Dec. 9, Genève, Switzerland.

Graph	Split	Nodes	Edges	Class 0	Class 1	Class 2	Time steps	Interval (Years)
Full	None	19,717	44,324	4,103	7,739	7,875	42	1967 - 2010
Transd.	Train	11,664	24,645	2,964	3,508	5,192	38	1967 - 2006
	Val.	3,697	4,535	524	1,803	1,370	1	2007 - 2007
	Test	9,810	15,144	1,372	4,795	3,643	3	2008 - 2010
Inductive	Train	11,664	24,645	2,964	3,508	5,192	38	1967 - 2006
	Val.	2,093	2,113	297	1,123	673	1	2007 - 2007
	Test	5,960	6,928	842	3,108	2,010	3	2008 - 2010

Table 3.4: Summary statistics of the PubMed temporal graph. The dataset is divided into transductive and inductive splits for machine learning tasks, with time-respecting training, validation, and test sets. Classes are research subtopics in diabetes mellitus (Type 1, Type 2, Experiments).

tific publications, and directed edges represent citation relationships. Temporal information is encoded at the edge level: each citation is assigned the publication year of the citing paper. Yearly resolution was adopted because exact publication dates are not consistently available for all records, whereas publication years are. The resulting temporal graph spans 1967–2010, yielding $T = 42$ temporal intervals. Table 3.4 summarizes its main statistics, including node and edge counts, class distribution, and the temporal intervals used for training, validation, and testing. NetworkX-Temporal provides the graph together with node-level features and time-respecting splits for transductive and inductive learning settings, also available in PyTorch Geometric format [40, 146].

Real-world temporal graph datasets with node-level features and labels for node classification or clustering remain limited, although recent releases have improved coverage [99]. In several such datasets, node features are obtained by pretraining Node2Vec embeddings [100], which restricts evaluation to a transductive setting, as random walk sampling during pretraining may expose node representations to nodes later assigned to validation or test sets. The PubMed citation graph, released by [113] and later used for graph-based node classification in [197], is widely adopted in benchmark studies and was therefore selected for temporal augmentation. The graph contains $|V| = 19\,717$ nodes, representing papers, and $|E| = 44\,335$ directed edges (11 of which are bidirectional and removed in PyTorch Geometric), representing citations among them, plus node-level features and class labels for downstream node classification tasks. Features are bag-of-words representations extracted from paper abstracts using the 500 highest-scoring terms by term frequency–inverse document frequency (TF-IDF), while labels correspond to three diabetes mellitus subtopics in the PubMed database: Type 1, Type 2, and experimental studies. The time-respecting split supports both transductive and inductive learning settings, as detailed in Table 3.4; their distribution is illustrated in Figure 3.6.

To obtain temporal metadata, the official PubMed API was queried for each paper's publication date and cross-verified against the original data released by [113], with records disambiguated to match the implementation derived from [197] and currently available in PyTorch Geometric [40]. Because citing and cited papers may belong to different publication years, temporal information was assigned as an edge attribute; as

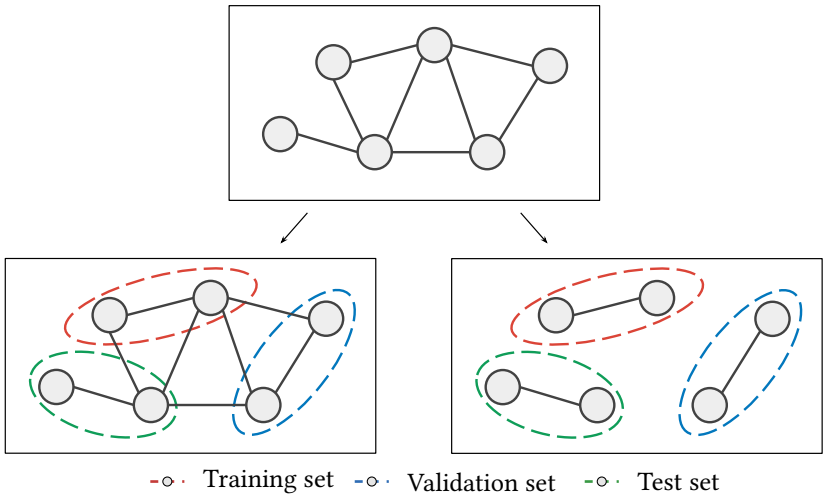


Figure 3.7: Comparison between transductive and inductive learning settings. On transductive (bottom left), validation and test edges are seen during training; while on inductive (bottom right), all node and edge sets are disjoint, requiring models to generalize to unseen graph regions.

monthly or daily resolution was not consistently available for all records, a yearly resolution was adopted. The resulting dataset preserves the nodes, edges, features, and labels of the original PubMed graph while adding edge-level timestamps over $T = 42$ time steps, spanning 1967–2010. By adding edge-level timestamps while preserving the original nodes, features, and labels, temporal graph learning experiments are made possible, while retaining comparability with previous baselines. Figure 3.7 illustrates the difference between transductive and inductive graph learning settings, respectively.

The dataset is publicly available [146] and included as a built-in loader in NetworkX-Temporal with node features, class labels, edge timestamps, and time-respecting training, validation and test splits. It is used in this chapter for experiments with graph autoencoder strategies for node clustering, providing a compact benchmark for evaluating whether citation timing can improve learning beyond the original static PubMed graph, even when the embedding encoder is not designed to leverage temporal information.

Dataset analysis and characteristics

The structure of the dataset is consistent with typical citation networks, summarized by Figure 3.8. The total-degree and in-degree distributions are heavy-tailed, with a small number of papers receiving many citations and many papers receiving few or none. In contrast, out-degree frequencies are more constrained, reflecting the finite number of references in each paper and the directed acyclic structure of citation networks. This imbalance is characteristic of the domain and introduces modeling challenges, since learning algorithms must handle strong degree heterogeneity and uneven activity levels.

Temporal variation in this dataset is comparatively constrained relative to social interaction datasets, where node and edge turnover is more pronounced. Because papers remain in the citation network once published and citation edges are time-stamped events, temporal variation reflects the arrival of new publications and citations rather than repeated interactions among the same entities. Figure 3.9 reports pairwise comparisons of active node sets across years, with overlap concentrated near adjacent periods;

the right matrix uses the Sørensen-Dice similarity measure, i.e., $|A \cap B|/(|A| + |B|)$, to quantify the similarity of active node sets across years. In contrast, Figure 3.4 illustrates stronger node and edge turnover in the CollegeMsg dataset, where pairwise interactions repeatedly occur among changing sets of students. This highlights how dataset domain determines which temporal structures are informative for downstream modeling, cautioning against treating temporal graph datasets as interchangeable benchmarks and motivating domain-specific model selection together with synthetic benchmarks for controlled experimentation and ablation studies, as later explored in Chapter 4.

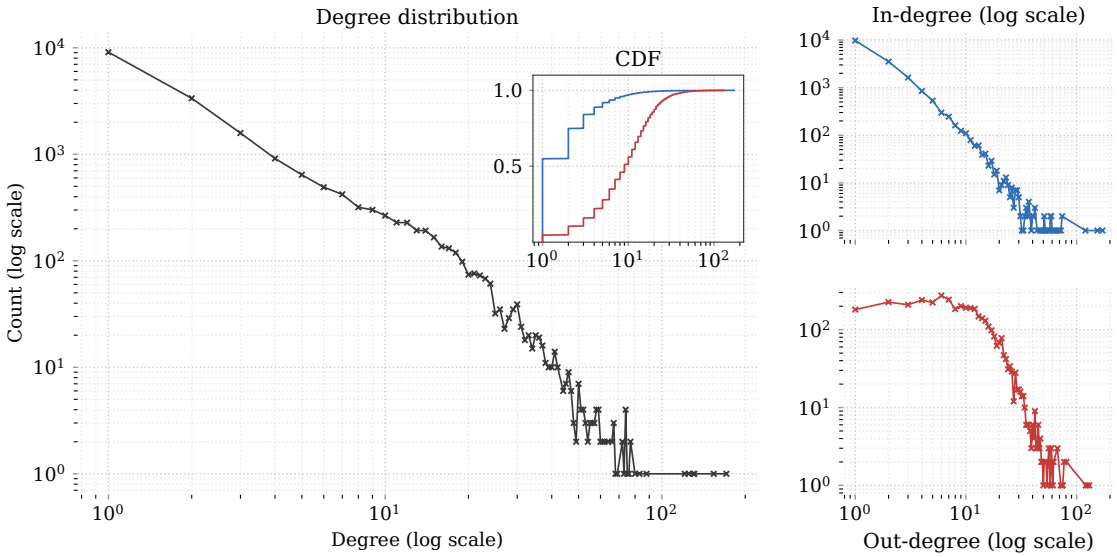


Figure 3.8: Degree distributions of the PubMed graph dataset. The left panel shows total degree, while the right panels separate in-degree and out-degree; the inset reports binned total-degree counts. The in-degree distribution is heavy-tailed, as expected in citation networks, while the out-degree is more constrained due to the acyclic and limited nature of paper bibliographies.

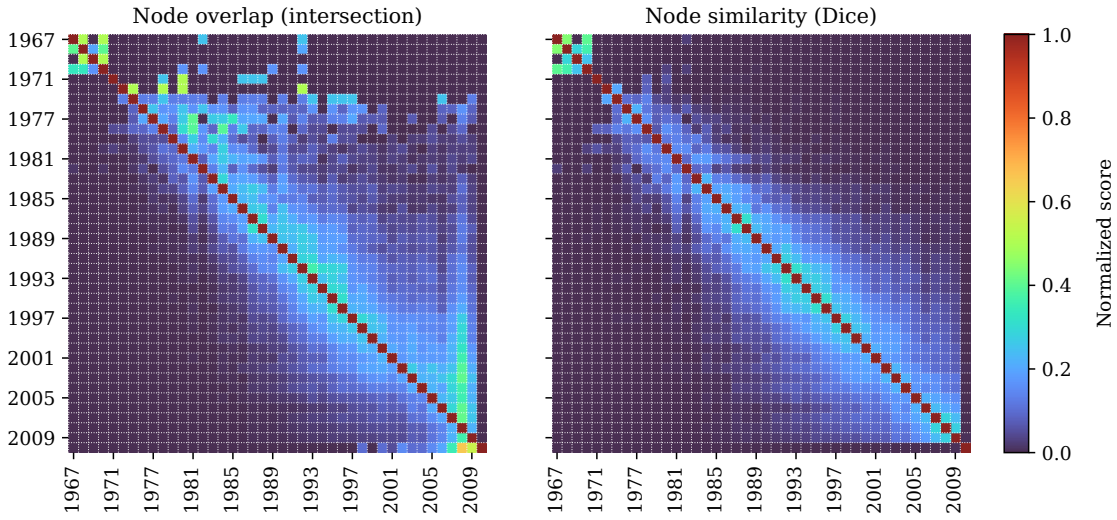


Figure 3.9: Node and edge set overlap in the PubMed-Temporal graph dataset with yearly splits. The low turnover of nodes and the empty intersection among edge sets is characteristic of citation networks, where edges occur only once and node additions to the network are permanent. As edge turnover is zero, Dice similarity coefficients are indicated on the right for each interval pair.

Code snippet for loading and basic usage

The following code snippet demonstrates how to load the PubMed-Temporal dataset:

```
» import networkx_temporal as tx
» TG = tx.generators.pubmed_graph(features=True)
» snapshots = TG.slice(bins=TG.number_of_snapshots()/2, attr='time')
```

This imports NetworkX-Temporal, loads the dataset as a temporal graph object named `TG`, and slices it into two-year snapshots using the edge-level `time` attribute. By default, the dataset is loaded with $T = 42$ snapshots. Once loaded, the graph follows NetworkX API conventions for accessing node and edge attributes and can be transformed or converted to other supported formats using the built-in library utilities.

3.2 Applications to the study of graph dynamics

Two applications of the software library are presented next: adapting a graph autoencoder and clustering model [185] to the temporal domain by topology augmentation, and analyzing entropy in production systems modeled as dynamic graphs [180].

3.2.1 Reconstruction and clustering with deep autoencoders[5]

An initial experiment was conducted to adapt a graph clustering model based on deep autoencoders [185] to temporal data through topology augmentation. The goal was to construct time-augmented graph representations and evaluate their effect on a real-world dataset with edge-level timestamps and node-level features. The selected model and augmentation procedure employing NetworkX-Temporal are described next, followed by experimental results and discussion. This example illustrates how the software can support machine learning workflows on dynamic graphs as a preprocessing layer.

Time-augmented graph generation

Motivated by multislice modularity [111], the augmentation process encodes temporal information topologically by slicing the graph into T discrete snapshots, adding temporal node copies ('proxy' nodes), and coupling corresponding copies across adjacent time steps through inter-slice edges. This construction allows static graph models, such as the one considered next, to operate on an augmented topology in which temporal proximity is represented through graph connectivity, enabling temporally informed node embeddings and community assignments without modifying the model architecture.

The result is an 'unrolled' graph in which each original node i is represented by at most T temporal copies, connected to its neighbors within each snapshot and to its own copies in adjacent time steps. In the experiments, the PubMed graph was discretized into $T \in \{2, 3, 4, 5, 6\}$ snapshots, with node features shared across temporal copies and weighted inter-slice edges controlling the strength of temporal coupling. This allows

[5]Part of the content in this section was presented in: *Deep Community Detection on Attributed Dynamic Graphs with a Spatio-Temporal Graph Neural Network.* Passos, N.A.R.A. (2024). Poster presentation. International School and Conference on Network Science (NetSciX'24). Jan. 22–25, 2024, Venice, Italy.

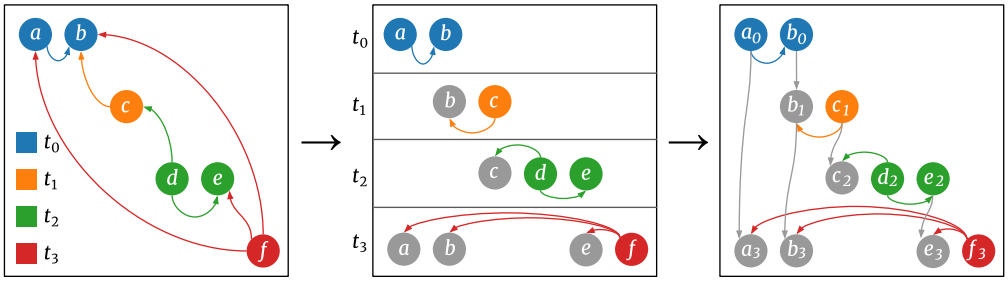


Figure 3.10: Graph augmentation process. In the final, unrolled graph, each node i has T copies (temporal nodes) i_t, with edges linking them to their neighbors in the same snapshot and to themselves in subsequent time steps. Note that the order and size of the graph therefore increases.

message passing to incorporate both contemporaneous neighbors and past states of the same node, exposing a static graph model to temporal structure through topology alone.

Deep Attentional *Augmented* Graph Clustering

The Deep Attentional Embedded Graph Clustering (DAEGC) model [185] learns node representations from both structure and content through an attentional encoder, optimized by a dual loss combining graph reconstruction and clustering objectives (Figure 3.11). Attention is computed over a polynomial adjacency matrix with $K = 2$, letting each node attend to its neighbors and their neighbors when aggregating information,

$$\hat{\mathbf{A}}^{(K)} = (\hat{\mathbf{A}} + \hat{\mathbf{A}}^2 + \ldots + \hat{\mathbf{A}}^K)/K, \tag{3.1}$$

where $\hat{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ is the normalized adjacency matrix with self-loops, and $\mathbf{D}$ is the diagonal degree matrix with $D_{ii} = \sum_{j \in \mathcal{N}_i \cup \{i\}} A_{ij}$, the self-loops ensuring a node's own features enter aggregation. The encoder thus considers 2-hop neighborhoods, while graph reconstruction is still performed on the original (binary) adjacency matrix.

Within this attentional paradigm, a latent representation is a weighted sum of neighbors' features modulated by learned attention coefficients. For each pair of nodes i and j, a constant a_{ij} is first obtained as the inner product between a weight vector $\mathbf{a}$ and their concatenated weighted features; a $\mathrm{softmax}$ then yields the final coefficients α_{ij}, with the polynomial adjacency matrix weighting neighbors by their proximity to i:

$$\alpha_{ij} = \frac{\exp(\sigma \hat{\mathbf{A}}^{(K)}_{ij} a_{ij})}{\sum_{n \in \mathcal{N}_i} \exp(\sigma \hat{\mathbf{A}}^{(K)}_{in} a_{in})} = \frac{\exp(\sigma \hat{\mathbf{A}}^{(K)}_{ij} (\mathbf{a}^\top [\mathbf{W}\mathbf{x}_i || \mathbf{W}\mathbf{x}_j]))}{\sum_{n \in \mathcal{N}_i} \exp(\sigma \hat{\mathbf{A}}^{(K)}_{in} (\mathbf{a}^\top [\mathbf{W}\mathbf{x}_i || \mathbf{W}\mathbf{x}_n]))}, \tag{3.2}$$

where σ is a Leaky ReLU nonlinearity and $\mathbf{W}$ is a weight matrix; from the perspective of attribute features alone, this is a single-layer feed-forward network. Both $\mathbf{a}$ and $\mathbf{W}$ are Xavier-initialized [55] and normalized, so coefficients are comparable across nodes.

Since the embeddings already encode structure and content, DAEGC reconstructs adjacencies with a simple inner-product (sigmoid) decoder. The model optimizes a composite loss function $\mathcal{L}_{\text{DAEGC}}$ that is the sum of a reconstruction term, equivalent to the binary cross-entropy between the original adjacency $\mathbf{A}$ and its reconstruction $\mathbf{A}'$, and a clustering term, computed as the Kullback–Leibler divergence between soft assignments

Q and a target distribution P and weighted by a hyperparameter $\gamma = 10$ (default):

$$\mathcal{L}_{\text{DAEGC}} = \mathcal{L}^{(r)}_{\text{DAEGC}} + \gamma \mathcal{L}^{(c)}_{\text{DAEGC}}, \tag{3.3a}$$

$$\mathcal{L}^{(r)}_{\text{DAEGC}} = \frac{1}{n} \sum_{i=1}^{n} \left(\mathbf{A}_{ij} \cdot \log \mathbf{A}'_{ij} + (1 - \mathbf{A}_{ij}) \cdot \log(1 - \mathbf{A}'_{ij}) \right), \tag{3.3b}$$

$$\mathcal{L}^{(c)}_{\text{DAEGC}} = \sum_{i \in V} \sum_{c \in C} P_{ic} \log \left(\frac{P_{ic}}{Q_{ic}} \right). \tag{3.3c}$$

In this formulation, soft assignments Q_{ic} measure the similarity between node and centroid embeddings through a Student's t-distribution, and the target P_{ic} optimizes them by emphasizing confident assignments and normalizing by cluster frequency, that is,

$$Q_{ic} = \frac{(1 + ||h_i - h_c||^2)^{-1}}{\sum_{k \in C} (1 + ||h_i - h_k||^2)^{-1}}, \quad P_{ic} = \frac{Q_{ic}^2 / \sum_{i \in V} Q_{ic}}{\sum_{k \in C} (Q_{ik}^2 / \sum_{i \in V} Q_{ik})}. \tag{3.4}$$

Centroids are initialized by k-means on the pretrained features, and P is recomputed every five epochs. Note that updating the target outside the gradient loop results in a model that is not end-to-end differentiable, a common trait of deep clustering methods by autoencoders, which the architecture later presented in Chapter 7 addresses.

Applying the unmodified DAEGC encoder to this unrolled graph yields what was termed Deep Attentional Augmented Graph Clustering (DAAGC). The encoder remains static by design: its attention mechanism and dual objective are unchanged, and nothing in the architecture is aware that the input encodes time. The augmentation instead acts entirely through the graph topology, and therefore through message passing. On the one hand, each node now attends to its own copies in adjacent snapshots alongside its contemporaneous neighbors, so that a node's embedding reflects its recent history and changing neighborhood, and the attention coefficients (Equation 3.2) may in principle weight temporal and spatial edges differently, all without architectural change; this added context is what produces the modest clustering gains reported next. On the other

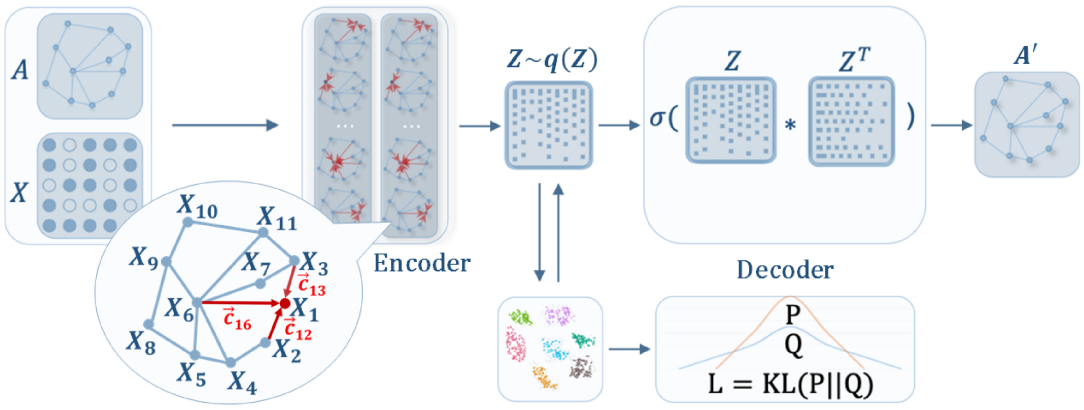
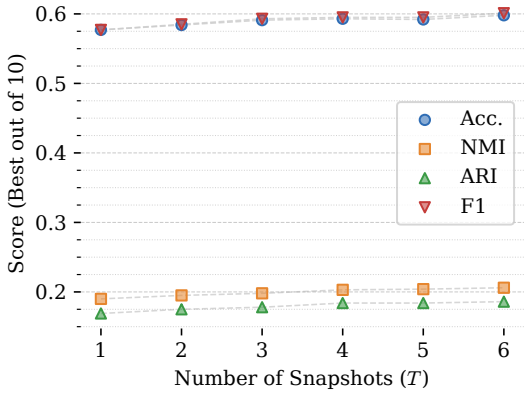


Figure 3.11: Architecture illustration of the DAEGC model, presented in [185]. Adjacency matrix $\mathbf{A}$ and node feature matrix $\mathbf{X}$ are encoded into latent node embeddings $\mathbf{Z}$ through a GAT-based encoder, where the inner product is used to reconstruct (decode) the adjacency matrix $\mathbf{A}'$ with a nonlinear activation (sigmoid) function. A second function q (k-means) obtains soft cluster assignments Q, optimized by KL divergence for a target distribution P, updated every 5 epochs.



Dataset	|T|	Acc.	NMI	ARI	F1
Static	1	0.577	0.190	0.169	0.577
Temporal	2	0.584	0.195	0.175	0.585
	3	0.591	0.198	0.178	0.593
	4	0.593	0.203	0.184	0.595
	5	0.592	0.204	0.184	0.595
	6	0.598	0.206	0.186	0.601

Figure 3.12: Clustering metrics on augmented graph. Results reported on best out of 10 runs.
Table 3.5: Clustering metrics on augmented graph. Results underlined on best out of 10 runs: accuracy, normalized mutual information (NMI), adjusted Rand index (ARI), macro F1-score.

hand, because temporal copies share identical features, inter-slice edges largely recirculate a node's own attributes across snapshots, increasing redundancy and drawing the representations of its copies toward one another — a form of oversmoothing that may worsen with depth, while long inter-slice paths can bottleneck information between distant time steps (oversquashing) [211]. Whether augmentation helps therefore hinges on the temporal signal introduced by the coupling outweighing the redundancy it injects, a balance governed by the number of snapshots T and the inter-slice coupling strength.

Experimental results and discussion

Evaluation considered only the PubMed-Temporal dataset and was conducted in a transductive setting in which final node embeddings predict node classes without any architectural change. As shown in Figure 3.12 and Table 3.5 (best of ten runs), topological augmentation slightly improved clustering as T increased, with a modest absolute gain of about 4% in accuracy (0.577 to 0.598 at $T = 6$), suggesting that reconstruction was able to exploit temporal information for more informative embeddings. The gains are therefore marginal, yet still incur noticeable computational costs from the increased graph order and size: across $t = 2$ to 6, the augmented graph grows from roughly $20{,}000$ to $24{,}000$ nodes and $45{,}000$ to $49{,}000$ edges, while the average runtime rises steadily from about $49\ s$ to $81\ s$, with 500 epochs for training on a single NVIDIA A100 GPU.

That is, in fact, the method's central limitation: slicing and unrolling grows the order and size of the graph with T, inflating training cost for dense or non-sparse operations and forcing coarse temporal resolutions (here limited to $T = 6$, aggregated from 42 available years). Time binning then becomes a critical hyperparameter, since coarse intervals obscure fast dynamics while fine intervals risk sparsity, and slice duration, aggregation, and snapshot constraints all affect the resulting topology. Although automatic selection through strategies not here discussed mitigate the need for manual tuning, such as higher-order De Bruijn projections [32], they do so at the price of added complexity.

The approach is also not universally appropriate: in PubMed-Temporal, node labels are fixed, and temporal variation enters only through citation timing, so temporal copies retain identical attributes, mainly exposing a static encoder to edge-level temporal in-

formation. Together, these shortcomings motivated the controlled benchmarking framework introduced in Chapter 5 and the dedicated temporal architecture found in Chapter 7. Although oversmoothing and oversquashing effects were not evident due to the small number of layers ($L = 2$) and snapshots ($T \leq 6$), they may arise in deeper architectures or with longer temporal paths, limiting generality to other dataset domains.

Lastly, note that the presented model performs graph clustering, but not community detection in the sense defined in Chapter 1: it optimizes adjacency reconstruction and cluster-assignment sharpening (Equation 3.3a) rather than a graph-theoretic objective such as modularity or conductance; which would instead place it among 'descriptive' [135] community detection methods. The baseline experiments in this chapter illustrate that the augmentation strategy can improve clustering performance, demonstrating the utility of temporal graph abstractions for machine learning workflows, but the gains are modest and come at the cost of increased graph size, training time, and hyperparameter tuning. This motivated the thesis's central move of optimizing a community objective directly through spectral and modularity-based methods for temporal graphs, presented in Chapter 5, and later returning to the neural setting with a model that optimizes such an objective end-to-end, introduced in Chapter 7. The next section explores how temporal graph abstractions may support applied analyses of information processing in dynamic systems, as a complementary use case to the experiments just presented.

3.2.2 Entropy and information processing in temporal graphs[6]

Entropy and information theory provide a framework for analyzing temporal graphs by quantifying uncertainty, predictability, and organization in time-evolving systems. In dynamic networks, entropy-based measures can assess how node and edge activity changes over time, revealing patterns of information flow, redundancy, novelty, and structural variability. High entropy indicates less predictable activity, such as frequent changes in connectivity or coordination demand, whereas low entropy suggests more regular or constrained temporal structure. These measures are therefore useful for studying real-world systems in which temporal dependencies and dynamics shape system behavior, including communication, biological, and organizational networks.

This section summarizes a co-authored application [180] of these concepts to pro-

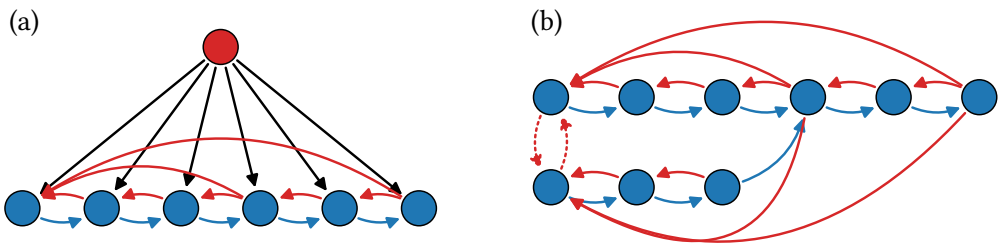


Figure 3.13: Illustration of complex coordination networks. (a) A production system with a central planning node (in red), plus local and global feedback loops. (b) Local and global feedback loops with synchronization. Graphs drawn with NetworkX-Temporal. Adapted from [180].

[6]The contents of this section have been partly published in: *Using Entropy Metrics to Analyze Information Processing Within Production Systems: The Role of Organizational Constraints.* Merode, F.v.; Boersma, H.; Tournois, F.; Winasti, W.; Passos, N.A.R.A.; Ham, A.v.d. (2025). Logistics 9(2), 46.

duction systems modeled as temporal graphs. Although the study is outside the main community detection focus of the thesis, it demonstrates how the temporal graph abstractions here developed can support applied analyses of dynamic coordination and information processing, facilitating the modeling and analysis of time-varying systems.

Application to real-world networks in simulated scenarios

In this setting, nodes represent agents and edges represent active coordination requests triggered by unfavorable events. A measure of Shannon entropy [15], then adapted to the graph domain, was used to quantify system-wide uncertainty in coordination activity, while degree centrality was used to identify agents with high coordination load. Two contrasting operational environments were compared: the Toyota Production System (TPS), modeled as a standardized system with closed-loop feedback, and a university obstetrics clinic, modeled as an open-loop push system with unpredictable patient arrivals and workload variability, increasing stochasticity. Figure 3.13 illustrates two representative coordination networks and highlights the differences in their topology.

In the TPS simulation, a Monte Carlo process (1000 replications per production period) evaluated entropy under two coordination mechanisms: local 'need help' requests and global 'andon' interventions. Zero-entropy states occurred in 38.40% of local help windows and in 76.62% andon coordination intervals, indicating that TPS coordination often remained predictable and inactive when tasks stayed within the allotted takt time. Although global coordination yielded higher average and maximum node degrees, its lower entropy suggests a more centralized and organized concentration of resources during major disruptions, consistent with the system's closed-loop feedback design.

In contrast, the obstetrics clinic simulations (500 days of coordination across three nurse segments) never reached a zero-entropy state, with entropy values ranging from 0.92 to 2.23 and average node degrees between 3.00 and 4.08. Because clinics lack feedback mechanisms to regulate arrivals and workload, coordination activity remained persistent and distributed across many agents. The comparison shows how organizational constraints shape information processing in temporal coordination networks: feedback-rich systems can suppress uncertainty through structured intervention, whereas open-loop systems sustain higher and more diffuse coordination complexity.

By employing graph-theoretical metrics on longitudinal data, the study [180] highlights the software's utility for information processing in complex systems with time-varying structures, and the need for robust temporal graph modeling frameworks to support such analyses. The following part of this thesis introduces the principles of community detection and graph simulation through stochastic block modeling, in order to present two distinct contributions: high-performance GPU implementations of spectral and modularity-based algorithms for temporal graphs, and a generator for time-varying attributed graphs for model benchmarking under controlled experimental conditions.

Part II
Community Detection

Nothing makes the earth seem so
spacious as to have friends at a distance;
they make the latitudes and longitudes.

Henry David Thoreau (1843)

Chapter 4

Detecting Communities in Networks

Complex systems often exhibit emergent properties that arise from interactions among their individual components. When these interactions form large networks, understanding system behavior from isolated components becomes impractical, motivating methods that identify mesoscopic organization. Community detection addresses this problem by seeking groups of nodes whose connectivity, attributes, and/or temporal behavior distinguish them from the rest of the network, yielding distinct definitions and approaches.

Community detection techniques may be broadly divided into *descriptive* and *inferential* methods [135]. Descriptive methods are exploratory: they treat the observed graph as fixed and usually optimize a quality function, such as modularity or conductance, that measures the 'goodness' of a partitioning. Inferential methods instead often assume a probabilistic generative model, based on statistical principles, and treat community labels as latent variables to be inferred from the observed edges. Both viewpoints aim to recover node groups, but they differ in what constitutes a community, whether uncertainty is estimated, and whether graph reconstruction is part of the modeling objective.

This distinction becomes more relevant when considering dynamic and attributed graphs, which are common in real-world applications. Static representations can hide temporal mechanisms: social interactions evolve, transportation routes are modified, and biological processes unfold over time. Node and edge attributes add further context for identifying communities, but also increase modeling complexity and may change the partition recovered by an algorithm [34]. Ignoring either aspect can therefore lead to oversimplified descriptions of the system. Both paradigms can account for these complications, for instance by extending quality functions to the temporal domain [10, 111] or by modeling community evolution as a Markovian process [51, 193], among others.

In computer science, communities learned with neural network-based approaches are sometimes referred to as 'deep' communities, particularly when multilayer representations are used to capture nonlinear structural or attribute patterns [97, 165]. Recent work has also begun to connect these models with classical community-detection objectives by incorporating graph-theoretic losses into graph neural architectures [179], allowing for a somewhat more direct interpretation of the clustering process and its results.

This chapter first discusses the theoretical foundations and practical implications of descriptive and inferential community detection. It then reviews methods based on spectral decomposition, heuristic optimization, and generative inference, before moving on to presenting the novel contributions of this research for temporal community detection.

4.1 Descriptive and inferential methods

A taxonomy for community detection methods may first seek to distinguish between descriptive and inferential approaches [135] — although both aim to partition the nodes of a graph into subsets, they differ in their underlying philosophies and methodologies.

A summary of the main characteristics of these two categories is provided by Table 4.1, highlighting their objectives, strengths, and limitations. Their distinction is not absolute, but it provides a useful framework for understanding the landscape of community detection techniques. While descriptive methods treat the observed graph as the object of analysis and seek a partition that optimizes a structural criterion or combination thereof, inferential methods may assume that the network was generated under a probabilistic model instead, and therefore seek to infer the latent community structure that best explains the observed edges — often by estimating model parameters, quantifying uncertainty, and reconstructing the graph (i.e., 'fitting' the model to the data).

Examples of the former include modularity [115], which compares the observed density of within-community edges against the expectation of a null model, and conductance [196], which measures the relative boundary size of a community with respect to its volume. Spectral techniques also fall under this category, as they rely on the eigenstructure of graph matrices to identify partitions that minimize cut-based objectives. Algorithms used to optimize such criteria include spectral methods [118, 182], greedy heuristics such as Louvain and Leiden [9, 176], and neural models that learn representations or assignments directly from the observed graph [145]. Their output is interpreted *post hoc* as a description of the data: no explicit generative mechanism is assumed and uncertainty quantification is limited, so graph reconstruction is not a natural by-product — making graph autoencoder models not directly comparable to generative models[1]. For exploratory analysis and large-scale networks, however, these methods remain attrac-

Category	Objective	Strengths	Limitations
Descriptive spectral decomposition; modularity maximization; greedy optimization (e.g., Louvain, Leiden); shallow graph encoders (skip-gram)	Directly optimize a quality function over partitions, such as cut size, modularity, conductance, or a relaxation thereof	Scalable and simple to implement; requires few modeling assumptions; suitable for exploratory analysis of large-scale static/temporal graphs	Limited uncertainty quantification; no explicit generation model; resolution limits, local optima, and detectability limits
Inferential stochastic block models (contextual, nested, mixed-membership, temporal); topic-based (e.g., RTM); latent-space models	Infer latent communities under a probabilistic generative model, usually by maximizing a likelihood or posterior, or minimizing MDL	Statistically grounded; selects the number of communities; supports model selection, uncertainty, hierarchies, attributes, and dynamics	Computationally more demanding; sensitive to modeling assumptions, priors, and inference approximations; less scalable to large graphs

Table 4.1: Descriptive and inferential approaches for community detection. Both categories can be extended to static, attributed, and dynamic graphs, but differ in their objectives, assumptions, and treatment of uncertainty. Note that some methods may combine elements of both paradigms.

[1] See Section 3.2.1. — Autoencoders learn low-dimensional node embeddings that can be used to reconstruct the adjacency matrix, but do not provide a probabilistic model for the graph generation process.

tive due to their scalability, interpretability, and comparatively simple implementation.

Inferential methods take a statistical view, treating the observed graph as a realization of a probabilistic generative model — most commonly a stochastic block model (SBM) [74, 90] — in which community labels are latent variables and model parameters are estimated by likelihood maximization, posterior inference, or related approximations. This provides a principled basis for model selection (based on information criteria such as minimizing description length), uncertainty estimation, graph reconstruction, and hierarchical organization through extensions such as the nested SBM [134]. The trade-off is computational cost: inference often requires Markov Chain Monte Carlo sampling, variational methods, or other approximations such as expectation-maximization, and the results depend on the chosen model assumptions and priors — a dependence that is not a defect, but an instance of a more general principle: no inference is free of inductive bias, and the assumptions made should be stated explicitly rather than left implicit [135]. Recent extensions incorporate node attributes into the generative process [34] or model community evolution over time through Markovian dynamics [193], thereby linking structural, contextual, and temporal information within a unified inferential framework.

From a practical standpoint, inferential methods may be less scalable to large networks, but they provide a rigorous approach when uncertainty quantification, model comparison, and interpretability are central requirements. Both paradigms have been extended to dynamic graphs, either by adapting static quality functions [10, 111] or by incorporating temporal dynamics into the underlying generative model [107, 193]. Moreover, recent neural approaches further blur these categories — embeddings learned by graph neural networks can be clustered with standard algorithms, yielding 'deep' graph clustering [185], while other architectures optimize graph-theoretic objectives such as modularity or conductance directly, yielding 'neural' community detection [179]. Relatedly, while skip-gram methods [60, 138, 173] may be taken as 'neural' methods, they are in fact equivalent to matrix factorization of a pointwise mutual information (PMI) matrix [93], and can be interpreted as spectral methods that implicitly factorize a Laplacian-like operator [143]. Notably, neural components are also not confined to descriptive approaches: for instance, recent work embeds a neural-prior within an inferential stochastic block model [36], in which node attributes determine community memberships, inferred through belief propagation and approximate message passing.

4.2 Static and dynamic community detection

This section presents three central approaches to community detection: spectral clustering and matrix factorization, heuristic optimization by maximizing quality functions, and statistical inference under generative models. Special attention is given to their temporal adaptations, forming the basis for the novel contributions of this research.

Static community detection provides the baseline setting from which dynamic and attributed formulations are typically derived. It considers a graph $G = (V, E)$, where V is the set of nodes and E the set of edges, and seeks to partition the nodes into k communities $\mathcal{C} = \{C_1, \ldots, C_k\}$ such that nodes in the same set are similar according to a structural, attribute-based, or statistical criterion — for instance, by maximizing the number of edges within communities compared to the number of edges between them.

Dynamic community detection extends static methods to evolving networks by al-

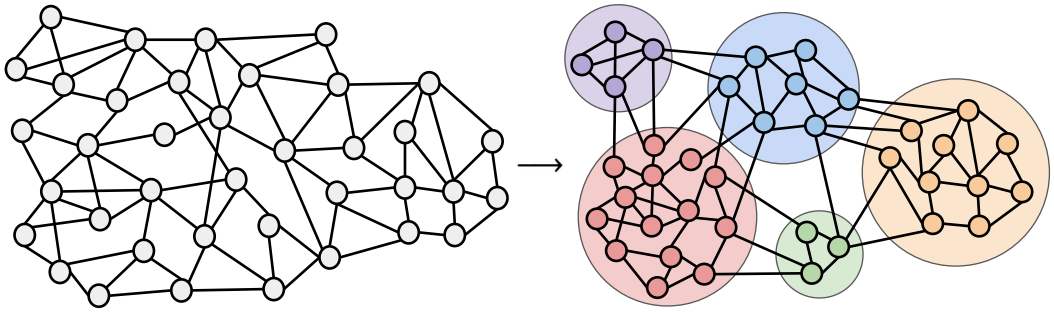


Figure 4.1: Example of a static graph with $|V| = 38$ nodes and $|E| = 81$ edges. A possible partition into $|C| = 5$ communities is shown, resulting in 19 edges between communities. Left and right graphs share the same topology; node positions are slightly adjusted for visual clarity.

lowing community assignments to vary over time. In discrete-time temporal graphs (Equation 2.2), this yields a sequence of snapshot-level partitions $\mathcal{C}^{(t)} = \{C_1^{(t)}, \ldots, C_{k_t}^{(t)}\}$, while in continuous-time temporal graphs (Equation 2.3), community membership may instead be represented as an event-level assignment, allowing interacting nodes in each event to change their community affiliation. In both scenarios, this amounts to the difference between *static* and *dynamic* community detection: the former seeks a single partition for the entire graph, while the latter allows partitions to evolve over time.

Although community detection on temporal graphs can be approached by applying static methods to each snapshot independently, followed by *post hoc* alignment of the resulting partitions, e.g., by Jaccard similarity [38], this general approach can be understood instead as a form of community *tracking* with a different set of challenges and assumptions [153]. The following sections focus instead on methods that explicitly incorporate temporal information into the community detection process, either by extending quality functions or by modeling community dynamics as a stochastic process.

4.2.1 Spectral decomposition and matrix factorization

Spectral decomposition methods view community detection as a relaxation of discrete partitioning objectives into continuous linear-algebraic problems. Instead of searching directly over all possible node partitions, they operate on graph matrices, such as the adjacency matrix, the Laplacian[2], normalized Laplacian, or modularity matrix, and use their eigenvectors or low-rank factors as node embeddings. Partitions are then obtained by thresholding these embeddings, or by a clustering algorithm such as k-means [118].

These methods are mathematically well-founded and computationally efficient on sparse graphs, but the continuous relaxation usually requires post-processing to obtain a discrete partition [184]. Their results can also depend strongly on graph connectivity, degree heterogeneity, and the chosen operator or normalization, since different matrices emphasize different notions of similarity [182]. From a practical standpoint, implementations differ in whether the number of communities k is fixed or decided *post hoc*, and in whether the embeddings are used directly for clustering or as an initialization for a subsequent step — complicating comparisons with heuristic and inferential methods.

[2]See Section 2.2. — Also known as the combinatorial Laplacian or Kirchhoff matrix if unnormalized.

The connection between spectral embeddings and community detection follows from relaxed cut objectives (Equations 2.6 and 2.7). For k communities, the eigenvectors associated with the k smallest nontrivial eigenvalues form an embedding matrix $\mathbf{U} \in \mathbb{R}^{|V| \times k}$, whose rows represent nodes in a low-dimensional Euclidean space. Relatedly, conductance [196] is a common metric for evaluating the quality of a community by measuring the fraction of its incident edge volume pointing outward, i.e., to its complement $\bar{C}_i$, so

$$\text{Conductance}(C_i) = \frac{\text{cut}(C_i, \bar{C}_i)}{\text{vol}(C_i)} = \frac{\text{cut}(C_i, \bar{C}_i)}{2m_i + \text{cut}(C_i, \bar{C}_i)}, \tag{4.1}$$

where m_i is the number of edges internal to C_i. Low values indicate that most incident edge volume remains inside the community, whereas high values indicate a comparatively large boundary. For graph bipartitioning, conductance is often symmetrized by normalizing the cut by $\min\{\text{vol}(C_i), \text{vol}(\bar{C}_i)\}$. Cheeger's inequality formalizes the relation between this normalized bipartition conductance and the spectrum of the normalized Laplacian by linking the second smallest eigenvalue to the conductance of the best cut [24], which explains why Laplacian eigenvectors can reveal assortative communities.

This cut-based view is also related to statistical-physics formulations of community detection. In Potts-type models, nodes are assigned discrete states corresponding to communities, and the objective is written as an energy function over neighboring or expected node pairs [144]. Aligned states along observed edges reduce the energy, while interactions induced by a null model penalize assignments that would also be expected at random. Such formulations connect spectral relaxations, modularity-like objectives, and spin-system (ferromagnetic) interpretations, and provide a basis for later temporal extensions through inter-slice couplings [111], explored in Section 4.2.2.

Matrix-factorization approaches provide a related but distinct view. Methods such as non-negative matrix factorization (NMF) are applied to a matrix (e.g., the adjacency matrix) based on low-rank factors under non-negativity constraints, e.g., by square-loss approximation, employed by various community detection methods [73, 187]. For a symmetric graph, $\mathbf{A} \approx \mathbf{W}\mathbf{W}^\top$, where the rows of $\mathbf{W}$ encode non-negative community-membership strengths — as nodes may have positive weights in multiple communities, the method naturally supports mixed (overlapping) memberships. For directed graphs, $\mathbf{A} \approx \mathbf{W}\mathbf{H}^\top$ instead, where $\mathbf{W}$ and $\mathbf{H}$ represent source- and target-side latent factors.

Attribute-aware variants and constrained factorizations further allow structural and contextual information to be combined, producing communities that are coherent in both topology and feature space, or by combining different graph operators in joint factorization procedures [194]. Despite this flexibility, spectral and factorization-based methods may still become expensive for large graphs, depending on the chosen operator, the number of communities, and the sparsity of the graph — related to the spectral gap of the operator, which influences the number of iterations required for convergence. When exact or high-accuracy relaxations are computationally impractical, heuristic optimization methods provide an alternative by trading optimality for scalability.

Spectral properties of dynamic communities

Dynamic community detection adds a temporal dimension to the spectral framework. Instead of seeking only a partition of V, the general objective becomes recovering com-

munity assignments that remain coherent across graph snapshots or event times. Although this can be approached by applying spectral methods independently at each snapshot and aligning the resulting partitions, an alternative is to instead encode structural affinity and temporal persistence in a single spectral problem, allowing joint temporal operators to recover smoothly evolving communities under suitable assumptions [51] and avoid the need for label alignment as a post-processing step.

The statistical physics interpretation also extends naturally to this setting. In static graphs, Potts-type formulations assign a discrete state to each node and minimize an energy function that rewards aligned states along observed edges and penalizes alignments expected under a null model [42, 144]. In temporal graphs, additional couplings can be introduced between time-adjacent copies of the same node, encouraging community assignments to remain stable unless the data support a transition. This is the same principle underlying multislice modularity [111], based on a supra-graph operator.

Spectral methods arise because these quadratic or energy-based objectives can often be relaxed into eigenvalue problems. For temporal graphs, this may involve supra-adjacency, supra-Laplacian, modularity, non-backtracking, or Bethe-Hessian operators (see Section 2.2.2). Their informative eigenvectors separate structural modes from noise and provide low-dimensional embeddings for temporal node copies. The choice of operator is important: ordinary Laplacian methods can localize on high-degree nodes or disconnected components in sparse graphs, whereas non-backtracking and Bethe-Hessian constructions are often better suited to sparse regimes near detectability limits [51, 168]

Detectability thresholds are usually formalized on planted community structures generated by a known model, under the assumption that the detection method is statistically consistent, i.e., it recovers the true partition as $n \to \infty$ [205]. In the static stochastic block model, recovery depends on a signal-to-noise ratio determined by the separation between within- and between-community connection rates relative to graph sparsity.

For a symmetric planted partition SBM with k equally sized groups, average within-community affinity a, and average between-community affinity b, the signal-to-noise (SNR) ratio for community recovery and detectability threshold may be expressed as

$$\mathrm{SNR}_{\mathrm{static}} = \frac{(a-b)^2}{k\,(a+(k-1)b)} > 1. \tag{4.2}$$

Below this threshold, the planted partition is statistically indistinguishable from random fluctuations in the sparse limit, even if the graph was generated with latent groups.

Temporal persistence changes this threshold by coupling observations of the same (latent) labels across time, so the detectability of evolving communities can be characterized by an effective SNR that scales the static ratio by a temporal-persistence gain,

$$\mathrm{SNR}_{\mathrm{dynamic}} = \frac{(a-b)^2}{k\,(a+(k-1)b)} \cdot \frac{1+\eta}{1-\eta} \;>\; 1, \tag{4.3}$$

where a and b are the average within- and between-community affinities (block probabilities), k is the number of communities, and $\eta \in [0, 1)$ controls the temporal correlation of community assignments. As $\eta \to 0$, consecutive labels become independent and the gain $(1+\eta/1-\eta) \to 1$, recovering the static threshold of Equation 4.2; meanwhile, as $\eta \to 1$, persistence makes the effective SNR diverge, so arbitrarily weak structure becomes detectable given enough temporally coupled snapshots. Note that node and edge

attributes may further improve detectability when informative of community membership [117], provided the model explicitly incorporates these signals in its formulation.

The dynamic case may therefore be read as an extension of the static planted partition setting. When community labels are persistent across snapshots, repeated observations reinforce the same latent partition and lower the structural separation required for recovery. When labels are weakly correlated, the benefit of temporal coupling vanishes and the problem approaches the static sparse SBM detectability regime. This distinction will be used in Chapter 5, where the generator parameter η controls community stability directly: $\eta = 1$ yields fixed memberships across snapshots, while $\eta = 0$ yields independent uniform reassignment under the symmetric transition parameterization, and η in Equations 4.2 and 4.3 may therefore be interpreted as analogous to a temporal SNR ratio.

The spectral properties of the temporal operator therefore determine which structural and temporal signals are recoverable, which clarifies when the problem is statistically feasible. The next sections use this connection to discuss temporal adaptations of modularity optimization and inferential models for dynamic graphs, which provide the theoretical basis for the methodological contributions introduced later in the thesis.

4.2.2 Heuristic optimization of quality functions

Numerous methods propose to detect communities by optimizing a quality function that measures the 'goodness' of a partition, usually relying on assortative assumptions.

Heuristic approaches to community detection operationalize structural definitions of communities through efficient, approximate optimization procedures. Like with cut-based objectives, such methods often assume 'good' partitions contain many edges within groups and comparatively few edges between them — such as Kernighan-Lin [81], one of the earliest algorithm examples, which iteratively swaps nodes between two groups to improve a cut-based objective. Similarly, Girvan-Newman [53] removes edges with high betweenness centrality to separate densely connected subgraphs; while label propagation provides a more local alternative, where nodes iteratively adopt labels from their neighborhoods [178, 208]. These methods are intuitive and often scalable, but depend on initialization, update order, stopping criteria, and implementation details, and therefore may converge to different or suboptimal partitions, especially if parallelized.

A central family of heuristic methods instead optimizes an explicit quality function over partitions. Given a candidate partition $\mathcal{C}$, a quality function $Q(\mathcal{C})$ assigns a scalar score measuring how well the partition satisfies a chosen structural criterion. The objective then becomes finding the partition $\mathcal{C}^*$ that maximizes (or minimizes) this score,

$$\mathcal{C}^* = \arg\max_{\mathcal{C}} Q(\mathcal{C}), \tag{4.4}$$

where the search space contains all possible partitions of the graph — a combinatorial problem that grows exponentially with the number of nodes. Because exact maximization is generally NP-hard, practical algorithms rely on greedy optimization, local refinement, simulated annealing, or other approximate procedures that merge, split, or reassign nodes when doing so improves the objective, i.e., when $Q(\mathcal{C}') > Q(\mathcal{C})$ for a candidate partition $\mathcal{C}'$ derived from $\mathcal{C}$, with no guarantee of reaching global optimality.

Static modularity optimization

The most widely employed quality function is modularity [115], which compares the density of within-community connections against the expected under a null hypothesis — a 'configuration' (random) graph with the same degree distribution as the observed network, usually Chung-Lu [23]. For a directed graph, modularity may be expressed as

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \gamma \frac{d_i^{\text{out}} d_j^{\text{in}}}{2m} \right) \delta(c_i, c_j), \tag{4.5}$$

where c_i is the community of node i, d_i^{out} and d_j^{in} are the out- and in-degrees of nodes i and j, respectively, m is the total number of edges, and $\delta(c_i, c_j) = 1$ when nodes i and j belong to the same community, and 0 otherwise. The resolution parameter $\gamma = 1$ (default) controls the community size scale, with $\gamma < 1$ favoring larger communities.

Modularity can also be expressed as a sum over communities, which clarifies its interpretation as a comparison between observed and expected internal connectivity:

$$Q = \sum_{c \in \mathcal{C}} \left[\frac{L_c}{m} - \gamma \left(\frac{d_c}{2m} \right)^2 \right], \tag{4.6}$$

where L_c is the number of within-community edges, and d_c the sum of its node degrees.

Greedy algorithms such as Louvain [9] and Leiden [176] provide scalable approximations to modularity maximization. Louvain alternates between local node moves that increase modularity and graph coarsening, producing a hierarchy of increasingly aggregated communities — after all nodes are assigned, the graph is collapsed into a new graph of communities (where each supernode represents a community), and the process repeats until no further improvement is possible. Leiden follows the same general strategy but adds a refinement phase that improves[3] the internal connectivity of communities and avoids some disconnected or poorly connected partitions produced by Louvain. Because these algorithms are fast, simple to apply, and effective on large weighted or directed graphs, they remain common choices for exploratory community detection.

Despite their practical success, modularity-based heuristics have several limitations. First, the optimization landscape is highly degenerate: many partitions may have similar modularity values, and local moves can converge to different optima depending on initialization and update order. Second, modularity has an intrinsic resolution limit [43]: for any fixed value of γ, small but well-defined communities may be merged because the null-model term depends on the total size of the graph. Third, modularity is a descriptive score, not a statistical test; a high-modularity partition does not by itself quantify uncertainty, provide a posterior distribution over partitions, or imply that the detected structure is statistically significant under a generative model [135]. These limitations motivate comparisons with inferential approaches such as stochastic block models, where model evidence, uncertainty, and graph reconstruction can be treated explicitly.

Connecting it to spectral theory, a modularity matrix $\mathbf{M}$ can be defined [116] as the difference between the observed adjacency matrix and a null (configuration) graph, i.e.,

$$\mathbf{M} = \mathbf{A} - \gamma \frac{\mathbf{d}_{\text{out}} \mathbf{d}_{\text{in}}^{\top}}{m}, \tag{4.7}$$

[3] Leiden's refinement step introduces a well-connectedness assumption that guarantees communities correspond to connected subgraphs and is shown to reduce time-to-convergence over Louvain [176].

where $\mathbf{d}_{\text{out}}$ and $\mathbf{d}_{\text{in}}$ are the out- and in-degree vectors, respectively, as in Equation 4.5.

If $\mathbf{C} \in \{0,1\}^{n\times k}$ is a hard community-indicator matrix, modularity can be written as

$$Q = \frac{1}{2m}\text{Tr}(\mathbf{C}^\top \mathbf{M}\mathbf{C}), \tag{4.8}$$

where corresponding edge normalizations follow for directed or weighted variants.

Relaxing the discrete constraints on $\mathbf{C}$ yields a spectral approximation in which nodes are projected onto leading eigenvectors of $\mathbf{M}$ and then partitioned by thresholding or clustering. Like with spectral clustering, the sign structure of the leading eigenvector determines the split for bipartitioning, while the absence of positive eigenvalues indicates that no modularity-increasing split is found under this spectral relaxation. Recursive application produces multiple communities, with the number of positive eigenvalues providing an upper bound estimate on the number of detectable communities — while the trivial solution is yielded by an all-ones eigenvector, i.e., with an eigenvalue of zero, where all nodes collapse to a single community resulting in a modularity score of $Q = 0$.

This formulation provides a bridge to temporal modularity: in dynamic graphs, the modularity matrix can be replaced by a supra-graph or temporal operator that includes both intra-slice structure and inter-slice coupling. The next section builds on this formulation to discuss two variants for discrete- and continuous-time temporal graphs.

Temporal modularity extends static modularity by incorporating time directly into the quality function. Rather than detecting communities independently at each snapshot, temporal variants of the function optimize structural fit and temporal coherence jointly. Next, two formulations are expanded upon: multislice modularity [111], which represents time through discrete graph slices, and longitudinal modularity [10], which treats interactions as time-stamped events ('link streams'). Both are descriptive objectives, with their matricial formulations connecting naturally to spectral optimization.

Multislice modularity optimization

Multislice modularity [111] extends the function to a sequence of snapshots by constructing a supra-adjacency matrix (Equation 2.10) encoding intra-slice (spatial) edges and inter-slice (temporal) couplings, allowing for a single optimization problem to arise.

Snapshots consider different null graphs, and inter-slice couplings encourage corresponding nodes to keep the same community assignment across time. For a temporal graph with slices indexed by $s, r \in T$, multislice modularity Q_{MS} may be expressed as

$$Q_{\text{MS}} = \frac{1}{2\mu}\sum_{s,r}\sum_{i,j}\left[\left(A_{ij}^{(s)} - \gamma_s \frac{d_{is}^{\text{out}}\, d_{js}^{\text{in}}}{2m_s}\right)\delta_{sr} + \delta_{ij}\omega_{jsr}\right]\delta\left(c_i^{(s)}, c_j^{(r)}\right), \tag{4.9}$$

where $A_{ij}^{(s)}$ is the adjacency matrix of snapshot s, d_{is}^{out} is the out-degree of node i in s, d_{js}^{in} is the in-degree of node j in s, m_s is the number of intra-slice edges, γ_s is the slice-specific resolution parameter, and ω_{jsr} is the coupling between node j from s to r.

The normalization constant 2μ is the total edge weight of the multislice graph, including intra-slice edges and inter-slice couplings. In the common consecutive-slice case, $\omega_{jsr} = \omega$ only when $r = s \pm 1$, so the parameter ω controls how strongly communities are encouraged to persist over adjacent snapshots. Alternatively, nodes may be coupled across its sequential appearances, similar to an 'unrolled' graph representation

(Figure 3.3), or a 'fully connected' temporal graph may be used, where $\omega_{jsr} = \omega$ for all $s \neq r$, which encourages global temporal coherence in community assignments.

This formulation can be represented as a modularity matrix over node-slice pairs. Let $\mathcal{V}_T = \{(i, s) : i \in V, s \in T\}$ denote the set of temporal nodes and copies. The multislice modularity matrix $\mathbf{M}_{\mathrm{MS}} \in \mathbb{R}^{|\mathcal{V}_T| \times |\mathcal{V}_T|}$ can then be similarly expressed in block form as

$$\mathbf{M}_{\mathrm{MS}} = \mathrm{diag}\big(\mathbf{M}^{(1)}, \ldots, \mathbf{M}^{(|T|)}\big) + \mathbf{\Omega}, \tag{4.10}$$

where each intra-slice modularity block follows Equation 4.7, and $\mathbf{\Omega}$ is the inter-slice coupling matrix whose nonzero entries connect temporal copies of the same node across slices according to ω_{jsr}. If $\mathbf{C}^{|\mathcal{V}_T| \times k}$ is the (temporal) community assignment matrix, then

$$Q_{\mathrm{MS}} = \frac{1}{2\mu} \mathrm{Tr}\big(\mathbf{C}^\top \mathbf{M}_{\mathrm{MS}} \mathbf{C}\big)\,. \tag{4.11}$$

This matrix form makes the relation to spectral modularity explicit: a binary $\mathbf{C}$ gives the hard partition objective, while relaxing it to real-valued assignments yields the usual spectral approximation. Greedy algorithms such as generalized Louvain or Leiden instead optimize the same objective directly on the multislice graph. Lastly, note that, with additional non-negativity and row-wise unit-normalization constraints, the relaxed assignments may also be interpreted as soft or mixed-membership community weights.

Longitudinal modularity optimization

A more recent extension proposes to compute time-aware modularity in a longitudinal paradigm [10]. Instead of selecting a window length to bin events into snapshots, longitudinal modularity treats edges as time-stamped interactions, or a link stream. The null model is adapted to account for the time that nodes spend in communities, and a smoothness penalty discourages frequent community switches. This makes the objective suitable for settings where temporal continuity is part of the community definition.

Under the mean-membership link expectation model, the expected number of links between two nodes is proportional to the geometric mean of their lifetimes within a community. Let $|T|$ denote the total observation time, and $|T_{i,C}|$ the amount of time a node i is found in community C. Longitudinal modularity can then be expressed as

$$Q_{\mathrm{L}} = \frac{1}{2m} \sum_{C \in \mathcal{C}} \left[\sum_{(i,j) \in C} A_{ij} - \gamma \frac{d_i^{\mathrm{out}} d_j^{\mathrm{in}}}{2m} \frac{\sqrt{|T_{i,C}|\,|T_{j,C}|}}{|T|} \right] - \frac{\omega}{2m} \sum_{i \in V} \chi_i, \tag{4.12}$$

where A_{ij} counts the interaction between nodes i and j over the observation interval, m is the total edge weight, d_i^{out} and d_j^{in} are the corresponding temporal degrees, ω controls the smoothness penalty, and χ_i counts the number of times a node transitions between communities across time, i.e., $\chi_i = \sum_{t_k \in T_i} 1$ if $t_{k+1} - t_k > \Delta$, where $\Delta = 1$ (default) is a tunable parameter that defines the minimum time interval for a 'switch' to be counted.

The first term of Equation 4.12 rewards interactions among nodes assigned to the same community, while the (temporal) null model corrects for expected links from long-lived memberships, and the final term penalizes discontinuous community trajectories. If memberships are fixed, the smoothness term, upper bounded by $2m - |V|$, becomes

constant and may be ignored during optimization; while the objective effectively reduces to that of static modularity on the aggregated graph, where $T_i = |T|$ for all $i \in V$.

This objective also admits a matrix form. Let $\hat{\mathbf{A}}$ be the aggregated weighted adjacency matrix of the link stream, and let $\mathbf{p}$ be a temporal presence vector whose entries encode normalized node lifetimes. The longitudinal modularity matrix may be expressed as

$$\mathbf{M}_{\mathrm{L}} = \hat{\mathbf{A}} - \gamma \frac{(\mathbf{d}_{\mathrm{out}} \odot \mathbf{p})(\mathbf{d}_{\mathrm{in}} \odot \mathbf{p})^{\top}}{2m}, \tag{4.13}$$

where $\odot$ denotes elementwise multiplication, so $\mathbf{M}_{\mathrm{L}}$ is the difference between observed temporal interactions and a rank-one null model weighted by node persistence, analogous to the static modularity matrix but corrected for temporal activity. For a (temporal) community assignment matrix $\mathbf{C}^{|\mathcal{V}_T| \times k}$, a compact matrix objective may be defined as

$$Q_{\mathrm{L}} = \frac{1}{2m} \mathrm{Tr}\big(\mathbf{C}^{\top} \mathbf{M}_{\mathrm{L}} \mathbf{C}\big) - \omega\, \mathcal{S}(\mathbf{C}), \tag{4.14}$$

and the smoothness term $\mathcal{S}(\mathbf{C})$, assuming fixed node ordering across time, rewritten as

$$\mathcal{S}(\mathbf{C}) = \frac{1}{2m} \sum_{i \in V} \sum_{t \in T_i} \left\| \mathbf{c}_i^{(t+1)} - \mathbf{c}_i^{(t)} \right\|_2^2 \approx \frac{1}{2m} \int_0^T \left\| \frac{d\mathbf{C}(t)}{dt} \right\|_F^2 dt. \tag{4.15}$$

i.e., the discrete sum of squared differences between consecutive time steps approximates the continuous-time Riemannian integral of the Frobenius norm of the derivative of $\mathbf{C}(t)$, up to a scaling factor determined by $|\mathcal{E}|$ or by an (optional) time discretization Δt.

This formulation yields a fully differentiable objective that can be optimized by gradient-based methods, and is one of the contributions of this thesis. Chapter 7 presents a graph neural network architecture designed for community detection in continuous-time temporal graphs optimizing such an objective in an end-to-end fashion.

4.2.3 Inference and generative models

Inferential approaches to community detection treat the observed graph as a realization of an underlying probabilistic model. Rather than optimizing a descriptive score over partitions, they specify a generative process for edges and infer the latent community labels that best explain the observed network. This provides a principled basis for likelihood-based inference, model selection, uncertainty quantification, and graph reconstruction by sampling from the fitted model. Among these approaches, the stochastic block model (SBM) is the canonical generative model for community structure [74, 90].

In the simplest undirected binary SBM, each node $i \in V$ is assigned a latent community label $z_i \in \{1, \ldots, k\}$, and edges are conditionally independent Bernoulli random variables given the labels. Let $\mathbf{B} \in [0,1]^{k \times k}$ denote the block-probability matrix, where B_{rs} is the probability of an edge between communities r and s. For a simple undirected graph without self-loops, the likelihood of the adjacency matrix $\mathbf{A} \in \{0,1\}^{n \times n}$ is

$$P(\mathbf{A} \mid \mathbf{z}, \mathbf{B}) = \prod_{i<j} B_{z_i z_j}^{A_{ij}} \left(1 - B_{z_i z_j}\right)^{1 - A_{ij}}. \tag{4.16}$$

A common special case assumes single within-community and between-community probabilities p and q, so that $B_{rr} = p$ and $B_{rs} = q$ for $r \neq s$. When $p > q$, the model

generates assortative communities; when $p < q$, it generates disassortative structures. This formulation is often found in literature as the planted partition model [26].

The basic SBM assumes that nodes in the same block are stochastically equivalent, and therefore have the same expected connectivity pattern. This assumption, however, is too restrictive for real networks, where degree heterogeneity is common even within the same community. The degree-corrected model (DC-SBM) [80] relaxes this constraint by introducing node-level parameters that modulate the expected number of edges while preserving block-level structure. In a Poisson formulation, the likelihood amounts to

$$P(\mathbf{A} \mid \mathbf{z}, \mathbf{\Lambda}, \boldsymbol{\theta}) = \prod_{i<j} \frac{\left(\theta_i \theta_j \lambda_{z_i z_j}\right)^{A_{ij}}}{A_{ij}!} \exp\left(-\theta_i \theta_j \lambda_{z_i z_j}\right), \tag{4.17}$$

where $\mathbf{\Lambda} \in \mathbb{R}_+^{k \times k}$ contains block-level Poisson rates, and $\boldsymbol{\theta}$ contains node-level degree parameters. The expected number of edges between nodes i and j is therefore $\theta_i \theta_j \lambda_{z_i z_j}$.

Under the DC-SBM, block-level affinity is therefore now separated from node-level degree propensities. Because the scaling of $\boldsymbol{\theta}$ and $\mathbf{\Lambda}$ is not identifiable without a constraint, a normalization is needed to ensure these two effects are identifiable. One common convention imposes $\sum_{i:z_i=r} \theta_i = 1$ for each community r; meaning that, inside each community, the node-level parameters must sum to unity. The Poisson formulation naturally permits multi-edges, but it approximates binary graphs in sparse regimes where $\theta_i \theta_j \lambda_{z_i z_j} \ll 1$ for all node pairs, so the probability of multiple edges is negligible and edge sampling behaves similarly to Bernoulli draws, as in the original formulation.

Several extensions adapt the SBM to richer forms of network structure. Nested SBMs introduce hierarchical priors over groups and support multiscale model selection through minimum-description-length principles [134]. Contextual or attributed SBMs incorporate node or edge attributes into the generative process, allowing structural and attribute-based evidence to jointly inform community assignments [34]. Mixed-membership variants further relax the assumption that each node belongs to a single block, representing communities as latent membership proportions rather than hard labels. These extensions increase modeling flexibility, but also make inference more dependent on the chosen priors, likelihood assumptions, and approximation scheme.

Although the main contributions of this thesis do not rely on SBM inference for detection, stochastic block modeling provides a rigorous framework for controlled benchmarking. By specifying the ground-truth labels, block affinities, degree heterogeneity, and attribute mechanisms, one can generate graphs with known community structure and tunable difficulty. This motivates the temporal SBM formulation introduced in the next chapter, which extends these principles to time-varying attributed graphs and provides synthetic benchmarks for evaluating community detection methods.

Inference and detectability in temporal graphs

Among inferential approaches, temporal extensions of stochastic block models introduce explicit priors for how community structure evolves over time. Rather than estimating independent partitions for each snapshot, these models couple latent labels across time, introducing temporal dynamics into their generative process. This provides a statistical counterpart to temporal smoothness in descriptive methods: communities are inferred as latent trajectories whose uncertainty can be quantified under the assumed model.

A common assumption is that node community assignments evolve according to a first-order Markov process. Transitions occur at discrete time steps, and the probability of a node transitioning communities depends only on its current assignment — a simple assumption that captures temporal dependence while keeping inference tractable [51].

Let $\boldsymbol{\tau} \in [0,1]^{k\times k}$ be a transition probability matrix, where τrs gives the probability that a node in community r at time $t-1$ transitions to community s at time t, i.e.,

$$P\big(z_u^{(t)} = s \mid z_u^{(t-1)} = r\big) = \tau_{rs}. \tag{4.18}$$

In this setting, a special case arises when the transition matrix is symmetric and homogeneous, so all communities share the same persistence probability η. In this case,

$$P\big(z_u^{(t)} = z_u^{(t-1)}\big) = \eta, \quad P\big(z_u^{(t)} = s \neq z_u^{(t-1)}\big) = \frac{1-\eta}{k-1}, \tag{4.19}$$

so the probability of a node retaining its label η, while the probability of switching to any other community is uniform across the remaining $k-1$ communities. Then,

$$\boldsymbol{\tau} = \eta\,\mathbf{I} + \frac{1-\eta}{k-1}\,(\mathbf{J}-\mathbf{I})\,, \tag{4.20}$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{J}$ is the all-ones matrix. Here, $\eta = 1$ defines fixed community assignments, $\eta = 1/k$ yields independent uniform labels at each time step, and $\eta = 0$ forces every node to switch to a different community at each time step.

In the same vein, a similar transition matrix parameterization may be considered,

$$\boldsymbol{\tau} = \eta\,\mathbf{I} + \frac{1-\eta}{k}\,\mathbf{J}. \tag{4.21}$$

In this form, $\eta = 0$ gives i.i.d. uniform community assignments across time, whereas $\eta = 1$ recovers the static-membership case. The diagonal transition probability is $\eta + (1-\eta)/k$, so η is not directly the probability of remaining in the same community. Under this convention, the signal-to-noise ratio in Equation 4.3 interpolates between persistent labels, where temporal observations reinforce the community signal, and the i.i.d. regime, where temporal information is not informative and the threshold equals the static case. Equivalently, it should be noted that the two parameterizations describe the same symmetric transition family under the relation $\eta = \eta' + (1-\eta')/k$, where η and η' are the diagonal probabilities given by Equations 4.20 and 4.21, respectively.

Inference for temporal SBMs therefore include estimating both structural parameters and temporal transition parameters. Depending on the model, these quantities may be estimated by maximum likelihood, posterior inference, Markov Chain Monte Carlo sampling, expectation-maximization, or variational approximations. The resulting posterior over label trajectories provides not only a partition at each time step, but also a probabilistic description of community persistence and change. As in static SBMs, inference continues to depend on the chosen priors, transition model, and approximation scheme.

Synthetic graphs generated under temporal SBM frameworks allow precise control over community structure. They therefore provide a rigorous basis for benchmarking dynamic community detection algorithms under known sparsity, noise, and time correlation. The next chapter introduces the Temporal Attributed Degree-Corrected Stochastic Block Model (TADC-SBM), the generative model developed in this thesis for producing attributed temporal graphs with ground-truth evolving communities, forming the basis for the experimental evaluation presented in the following chapters.

Chapter 5

Algorithms and Benchmarks

Unveiling a complex network's inner structure may be seen as the central challenge of community detection — although not all networks clearly display well-defined *functions*.

In biological networks, for instance, communities may correspond to cellular modules responsible for specific tasks within a cell or organism. In other settings, however, communities may instead describe regularities in the observed data, without necessarily corresponding to a functional entity. This distinction is important: inferential methods seek to explain observed connectivity through explicit generative assumptions, whereas descriptive methods identify mesoscale structure through, e.g., objective functions.

The ambiguity becomes more pronounced in dynamic and attributed contexts. Transient interactions may obscure persistent organization, while considering node and edge attributes may conflict with the structural or temporal signals. In such settings, validating communities as functional entities is difficult or beyond reach — and even the notion of a 'ground-truth' partition in the real-world is often ill-defined [131]. This raises a central interpretative challenge: the same network may support different partitions under distinct modeling assumptions. Community detection is therefore best treated both as modeling and evaluation problems: while the interpretation of detected groups depends on the assumptions under which they are obtained, benchmarks can control the experimental conditions under which methods are compared against known ground truths.

The theoretical discussion in Chapter 4 introduced community detection as a family of descriptive and inferential approaches. This chapter develops two methodological contributions from that background. The first follows the algorithmic route of the thesis: high-performance implementations of spectral and modularity-based algorithms for community detection in temporal graphs, integrated with the NetworkX-Temporal software ecosystem introduced in Chapter 3. The second follows the benchmarking route: a generative framework for evaluating community detection methods on time-varying attributed graphs under controlled structural, temporal, and attribute regimes.

The common motivation of this chapter is scalability under uncertainty: dynamic community detection increases modeling complexity in time-varying attributed graphs, as distinct signals may be reconciled to recover latent communities. At the same time, evaluation remains an issue: real-world networks rarely provide reliable ground-truth communities, and their observed topology may reflect multiple latent processes simultaneously. This chapter addresses both sides of the problem: how to make established temporal clustering methods computationally practical, and how to evaluate competing methods when structural, temporal, and attribute signals are known by construction.

5.1 The TADC-SBM generative model[1]

Stochastic block modeling provides a probabilistic framework for generating networks whose connectivity patterns are governed by latent group memberships. The Time-varying, Attributed, Degree-Corrected Stochastic Block Model (TADC-SBM)[2] extends this framework to generate dynamic graphs with evolving community structures, heterogeneous degree distributions, and community-aligned node and edge features [129]. The model is here employed as a controlled benchmark graph generator: by varying structural, temporal, and attribute parameters, it becomes possible to evaluate and compare community detection methods on known, controlled ground truths and signals.

5.1.1 Edge sampling procedure

Let $G^{(t)} = (V, E^{(t)})$ denote the graph snapshot at time $t \in \{1, \dots, T\}$, with $n = |V|$ nodes and k communities. Each node $u \in V$ has a community assignment $z_u^{(t)} \in \{1, \dots, k\}$, and the expected number of edges between nodes u and v depends on their current memberships through a block matrix $\mathbf{B}$. Each element B_{rs} is interpreted as an edge propensity between communities r and s, corresponding to the expected number of edges between the two groups. A degree correction vector θ introduces node-specific propensities, proportional to their expected degrees and normalized within each community. Together, these parameters allow the generator to combine group-level structure with heterogeneous degree distributions with controlled heterogeneity, including heavy-tailed configurations commonly observed in empirical networks (see Figure 3.5).

For each snapshot, edges are sampled conditionally independently given the memberships and model parameters. Following the Poisson degree-corrected SBM [80],

$$A_{uv}^{(t)} \sim \text{Poisson}\left(\lambda_{uv}^{(t)}\right), \tag{5.1}$$

where $A_{uv}^{(t)}$ is the observed edge (or multiedge) between nodes u and v at time t, and

$$\lambda_{uv}^{(t)} = \zeta\, \theta_u \theta_v B_{z_u^{(t)} z_v^{(t)}}, \tag{5.2}$$

where $\lambda_{uv}^{(t)}$ is the expected edge (or multiedge) between nodes u and v at time t, while $\zeta \in [0, 1]$ scales the sampled edge intensity and may be used to retain only a fraction of potential interactions. The degree vectors may be generated from a power-law distribution with exponent α_0, bounded by minimum and maximum expected degrees $d_{\min}$ and $d_{\max}$, before being normalized as required by the sampling procedure (see Figure 3.5).

The likelihood of an observed sequence of adjacency matrices may be expressed as

$$P(\{A^{(t)}\}_{t=1}^{T} \mid \{z^{(t)}\}_{t=1}^{T}, \mathbf{B}, \theta, \zeta) = \prod_{t=1}^{T} \prod_{u<v} \frac{\left(\lambda_{uv}^{(t)}\right)^{A_{uv}^{(t)}} e^{-\lambda_{uv}^{(t)}}}{A_{uv}^{(t)}!}. \tag{5.3}$$

[1] Section partly published in: *TADC-SBM: a Time-varying, Attributed, Degree-Corrected Stochastic Block Model.* Passos, N.A.R.A.; Carlini, E.; Trani, S. (2025). In Proceedings of the 30th IEEE Symposium on Computers and Communications (ISCC'25). IEEE Computer Society, Los Alamitos, CA, USA, 1–6.

[2] Available on PyPI: https://pypi.org/project/tadc-sbm/. Distributed under the Apache 2.0 License.

This expression omits self-loops for simplicity: for small values of $\lambda_{uv}^{(t)}$, the Poisson distribution approximates a Bernoulli distribution, effectively producing a simple graph with at most one edge between any pair of nodes. The implementation may generate multigraphs with self-loops before they are removed or projected to the graph representation used in the benchmark. When simple (binary) adjacencies are required, sampled edge counts may be projected to binary adjacencies by setting $A_{uv}^{(t)} = 1$ whenever at least one edge is sampled. If degree correction is not set, the model is simplified to a standard SBM by setting $\theta_u = 1$ for all nodes u. Note that the model is macrocanonical: the expected number of edges is controlled by the parameters, but actual counts may vary across time.

Under this projection, the corresponding edge probability is $1 - e^{-\lambda_{uv}^{(t)}}$. An additional parameter $\zeta \in [0, 1]$ introduces post hoc stochasticity: when $\zeta < 1$, sampled interactions may be unobserved, producing sparser snapshots and allowing low-degree nodes (who become isolated after edge removal) to disappear and reappear across time. This may be used to model incomplete observations — useful for link prediction settings, where only a fraction of the potential temporal edges is observed, which may be of interest for other graph learning models focused on distinct downstream tasks.

Temporal dynamics and community stability

Temporal dynamics are introduced by allowing node memberships to evolve across snapshots, following the dynamics introduced in Section 4.2. A transition matrix $\boldsymbol{\tau}^{(k\times k)}$ dictates the probability of a node transitioning from a community r to s, such that

$$P\left(z_u^{(t+1)} = s \mid z_u^{(t)} = r\right) = \tau_{rs}. \tag{5.4}$$

The parameter $\eta \in [0, 1]$ therefore effectively controls the probability of preserving community membership over time, i.e., the stability of temporal communities. Higher values of η generate stronger temporal signals, whereas lower values induce more frequent transitions and weaken the temporal signal available to community detection methods.

By default, transition probabilities are parameterized so that each row of $\boldsymbol{\tau}$ sums to one. Following the detectability formulation for dynamic networks [51], one option is to set the diagonal of $\boldsymbol{\tau}$ to η and distribute the remaining probability uniformly across the other $k-1$ communities, as in Equation 4.20. This interpolates between static communities ($\eta = 1$) and maximally unstable assignments ($\eta = 0$), where nodes always leave their

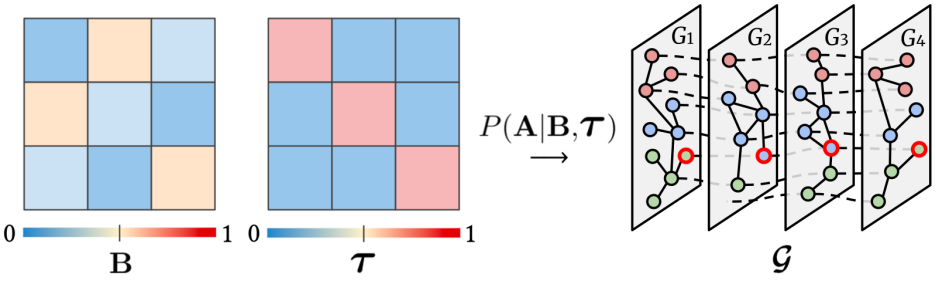


Figure 5.1: Illustration of the TADC-SBM model for a network with 10 nodes, 3 communities, and 4 snapshots. Note how highlighted node (in red) changes its community membership over time. For simplicity, initial node memberships and degree-correction parameters are omitted.

current community. A symmetric alternative, given by Equation 4.21, assigns transition mass uniformly across all communities, including the current one. Under this parameterization, the diagonal probability becomes $\eta + (1-\eta)/k$, so that $\eta = 0$ corresponds to uniform random reassignment and $\eta = 1$ again preserves the current community.

The model also includes a parameter $\xi \in \{0, 1\}$ controlling whether transition probabilities are evaluated with respect to current or initial memberships. If $\xi = 0$, transitions depend on the current community assignment, producing a standard Markovian evolution; if $\xi = 1$, transition probabilities are instead anchored to the initial membership assignment, allowing nodes to leave and later return to their original communities with consistent probability η over time. This distinction makes it possible to generate different forms of temporal organization, including drifting, recurrent, or more stable community trajectories. Note that the transition matrix $\boldsymbol{\tau}$ may also be manually defined to encode specific dynamics, such as preferential transitions between selected communities.

5.1.2 Feature attribution

TADC-SBM augments the generated temporal graph with node- and, optionally, edge-level attributes. Node features are generated from community-aligned Gaussian mixtures: each community c is assigned a centroid μ_c, and each node receives a d-dimensional feature vector sampled around the centroid of its assigned community.

For a node u with membership z_u, the feature vector $x_u \in \mathbb{R}^s$ is sampled as

$$x_u \sim \mathcal{N}(\mu_{z_u}, \sigma^2 \mathbf{I}), \tag{5.5}$$

where σ^2 controls the within-community feature variance and $\mathbf{I}$ is the identity matrix. Community centroids are sampled from a zero-centered normal distribution, so that

$$\mu_c \sim \mathcal{N}(\mathbf{0}, \sigma_c^2 \mathbf{I}), \tag{5.6}$$

where σ_c^2 controls the separation between community centroids in the feature space.

The ratio σ_c/σ therefore controls attribute relevance: larger values produce more separable feature clusters, whereas smaller values make node attributes less informative for recovering community memberships. In the benchmarking setting considered here, node features are generated once at the beginning of the process and remain fixed across snapshots. This deliberately separates temporal variation in the graph topology from the attribute signal, allowing models to be tested on whether they can reconcile dynamic structural information with static node-level features, and was chosen to provide

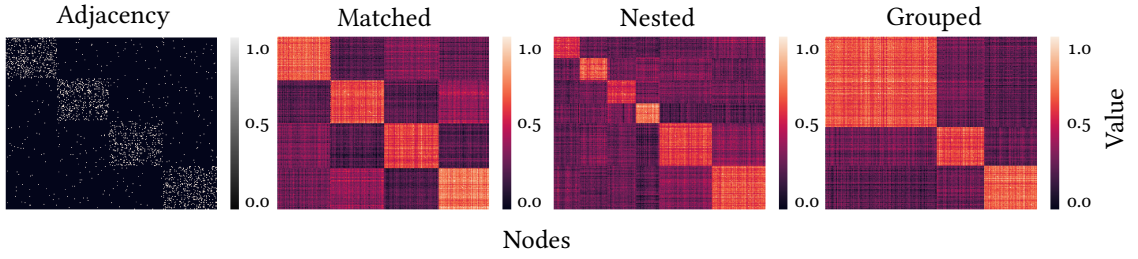


Figure 5.2: Community-aligned feature generation with TADC-SBM, adapted from [124]. Panels show the adjacency matrix and matched, nested, and grouped clusters in the feature space.

Type	Clusters	Impact
Matched	$\hat{k} = k$	Features and graph structures align; strongest detection signal.
Nested	$\hat{k} > k$	Features subdivide clusters; tests robustness to oversegmentation.
Grouped	$\hat{k} < k$	Features merge clusters; tests robustness to undersegmentation.

Table 5.1: Comparison among different types of attribute feature relationships to graph clusters in TADC-SBM, and their implications for model performance in community detection tasks.

a controlled setting for evaluating the interaction between topology, temporal dynamics, and feature informativeness for both temporal and static graph encoders.

Edge features may be generated with a similar logic, using different distributions for intra- and inter-community edges. For an edge (u, v), the edge feature vector $x_{e_{uv}}$ is

$$x_{e_{uv}} \sim \begin{cases} \mathcal{N}(\mathbf{0}, \mathbf{I}), & z_u = z_v, \\ \mathcal{N}(x_e \mathbf{1}, \mathbf{I}), & z_u \neq z_v, \end{cases} \tag{5.7}$$

where x_e controls the separation between intra- and inter-community edge-feature distributions, and $\mathbf{1}$ is an all-ones vector of the appropriate dimension. Larger values of x_e make edge attributes more informative with respect to community boundaries.

Hierarchical and nested feature spaces

The model also supports feature spaces whose cluster structure does not necessarily match the topological community structure. In the matched setting, each graph community corresponds to one feature cluster. In nested and grouped settings, the number of feature clusters $\hat{k}$ may differ from the number of topological communities k: several communities may share similar attributes, or a single community may contain multiple attribute subgroups, as shown in Figure 5.2. These configurations allow the generator to control the agreement between structural and attribute signals, including hierarchical structures, rather than assuming that topology and features encode the same partition.

This parameterization is useful for benchmarking methods under controlled signal alignment and disagreement. Feature-only methods, such as k-means, provide a baseline for attribute separability, while topology-only methods reveal how much community information is recoverable from the graph alone. Methods that jointly use topology and attributes can then be evaluated according to whether they improve recovery when the two signals are aligned, partially aligned, nested, grouped, or conflicting. Table 5.1 summarizes the feature-space configurations and possible impact on model performance.

5.1.3 Benchmarking protocols with synthetic data[3]

The purpose of the TADC-SBM model is, instead of estimating communities from empirical data, generating benchmark graphs in which the relevant signals are known by

[3]The content of this section was partly presented in: *Benchmarking Algorithmic and Neural Approaches for Temporal Community Detection.* Passos, N.A.R.A. (2025). Lightning Talk at the International School and Conference on Network Science (NetSci '25), Jun. 2–6, 2025, Maastricht, Netherlands.

construction. This allows community detection methods to be evaluated under controlled structural, temporal, and attribute regimes, avoiding the ambiguity of real-world metadata as ground truth [131]. In this setting, topology, temporal stability, and feature informativeness can be varied separately, making it possible to isolate which source of information a method is exploiting and how it performs in different scenarios.

This subsection does not attempt to derive a detectability threshold for TADC-SBM — while static and temporal SBM thresholds are known for idealized structural settings [51], their extension to time-varying, attributed, degree-corrected graphs is not straightforward. Feature signals may reinforce or conflict with structure signals depending on whether feature clusters are aligned (matched, nested, grouped) with the communities.

The static and temporal signal-to-noise conditions introduced in Chapter 4 (Equations 4.2 and 4.3) are therefore used as theoretical reference points. The objective here is empirical: to evaluate two simple baselines under controlled generated regimes, namely k-means on the node feature matrix and spectral clustering on the adjacency matrix. This provides separate measurements of the attribute-only and topology-only signals for a specific baseline configuration, to improve interpretation of evaluation results for graphs generated under different structural, temporal, and attribute regimes.

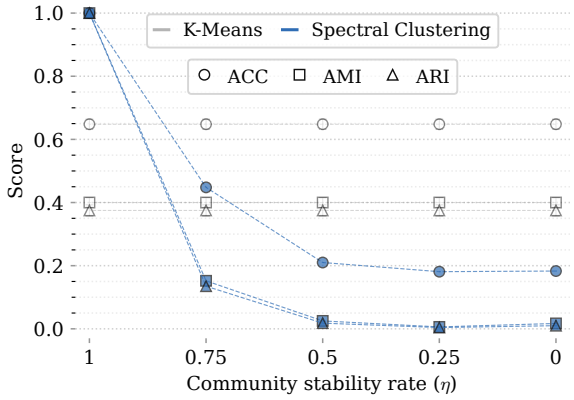
Baseline graph generation

The baseline graphs were generated with $n = 1024$ nodes, $k = 8$ communities, and $T = 8$ snapshots. Initial memberships were sampled uniformly, so that each node had probability $1/k$ of belonging to any community at $t = 1$. Edges were then sampled independently at each snapshot using the Poisson degree-corrected SBM procedure described in Section 5.1.1. Degree propensities followed a bounded power-law distribution with exponent $\alpha_0 = 2$, minimum degree $d_{\min} = 2$, and maximum degree $d_{\max} = 20$, yielding generated graphs with observed average degree $d_{\text{avg}} = 9.83 \pm 0.13$ across snapshots.

Temporal variation was controlled only through the community-stability parameter η, with $\eta \in \{0, 0.25, 0.5, 0.75, 1\}$. The transition matrix followed the symmetric parameterization described in Section 5.1.2: $\eta = 1$ preserves memberships across snapshots, whereas $\eta = 0$ corresponds to uniform random reassignment at each transition. The transition mode was fixed at $\xi = 0$, so transitions depended on current memberships, and the observation parameter was fixed at $\zeta = 1$, so all sampled interactions were re-

Dataset	$\lvert\mathcal{V}\rvert$	$\Sigma(\mathcal{V})$	$\lvert\mathcal{E}\rvert$	$\Sigma(\mathcal{E})$	$\lvert S\rvert$	d^V	$\lvert C\rvert$	$\lvert T\rvert$	Q	Cond.
SBM $\eta = 1$	1 024	8 192	36 764	40 269	1	32	8	8	0.433	0.476
SBM $\eta = .75$	1 024	8 192	38 340	40 095	1	32	8	8	0.102	0.778
SBM $\eta = .5$	1 024	8 192	38 774	40 213	1	32	8	8	0.062	0.815
SBM $\eta = .25$	1 024	8 192	38 929	40 259	1	32	8	8	0.052	0.824
SBM $\eta = 0$	1 024	8 192	38 707	40 055	1	32	8	8	0.054	0.822

Table 5.2: Baseline graphs generated with TADC-SBM. $|\mathcal{V}|$ and $|\mathcal{E}|$ denote nodes and edges in the temporal graph representation; $\Sigma(\mathcal{V})$ and $\Sigma(\mathcal{E})$ denote the sum of nodes and edges over time, counting repeated appearances across snapshots; $|S|$ is the number of connected components; d^V is the node-feature dimensionality; $|C|$ and $|T|$ are the number of communities and snapshots; and Q and Cond. report modularity and conductance of ground-truth communities, respectively.



Model	η	Accuracy
k-means	$*$	0.648 ± 0.016
Spectral	1	1.000 ± 0.000
	0.75	0.448 ± 0.000
	0.50	0.210 ± 0.000
	0.25	0.181 ± 0.000
	0	0.183 ± 0.000

Figure 5.3: Baseline performance on aggregated TADC-SBM graphs of Table 5.2. Attribute signal (static) remains independent of η, whereas structure signal becomes less informative as $\eta \to 0$.

Table 5.3: Mean accuracy of k-means on node attributes and spectral clustering on graph structure. Results averaged over 15 runs; $*$ indicates independence from η due to static node features.

tained. For evaluation, snapshots were reversed as $\mathcal{G}_\eta = \{G_\eta^{(T)}, \ldots, G_\eta^{(1)}\}$, so that the final snapshot contained the target community labels used for prediction.

Node attributes were generated once at the beginning of the process and kept fixed across snapshots, simulating a scenario where the attributes are known and stable, while the topology evolves over time. Node feature space has $s = 32$ dimensions, with within-cluster variance $\sigma^2 = 1$ and centroid variance $\sigma_c^2 = 6$. The baseline setting therefore varies temporal-topological stability while keeping the feature signal constant, allowing topology- and feature-only comparison under the same experimental conditions.

Empirical topology and feature baselines

This baseline experiment separates two signals: graph structure and node attributes, with the goal of measuring how much community information is available from each source in isolation. Spectral clustering is used as the topology-only baseline by applying it to the adjacency matrix of the generated (aggregated) graphs, while k-means is used as the feature-only baseline by applying it directly to the node feature matrix.

The stability parameter η controls the temporal component of the structural signal. When $\eta = 1$, community memberships remain fixed across snapshots, so topology repeatedly reflects the same planted partition. When $\eta = 0$, memberships are reassigned uniformly at random under the symmetric transition parameterization, removing temporal coupling between consecutive labels, similarly to the temporal-correlation parameter η from Equation 4.3, which may be interpreted as analogous in this setting.

The results in Table 5.3 and Figure 5.3 expose the intended separation between structural and attribute signals. As $\eta \to 0$, spectral clustering rapidly degrades because the final community labels become less aligned with the topology induced by unstable temporal trajectories. The experiment identifies that performance approaches near-random accuracy for $\eta \leq 0.5$, indicating that topology alone becomes weakly informative under low temporal stability — although the number of snapshots and other parameters (Equation 5.3) may influence the exact point at which this effect is observed for other graph simulation settings. In contrast, k-means on node attributes remains stable, since

the feature matrix is generated once, independently of η, and remains fixed over time.

As shown in this example, TADC-SBM can isolate topology-only and feature-only signals under controlled temporal regimes. Attributes provide an additional source of signal outside the assumptions of the purely structural SBM and dynamic SBM, to which a complete theoretical characterization of detectability jointly with structural and temporal signals may provide a more rigorous understanding of the observed empirical behavior. As discussed, their interaction with topology is nontrivial: feature clusters may match, partially align (nested/grouped), or conflict with the community structure, and this alignment determines whether attributes reinforce or obscure the structural signal.

This strategy is therefore useful for evaluating community detection methods under controlled conditions, and for identifying the regimes in which they are able to exploit structural, temporal, and attribute signals to recover latent communities. The next section and Chapter 7 use this generative framework to evaluate the performance of community detection methods and compare them against the baselines here described.

5.2 Scalable temporal community detection[4]

This section addresses the algorithmic problem of scaling temporal community detection to large dynamic graphs, returning to two classical families of methods: spectral clustering and modularity optimization. Both admit temporal extensions through snapshot coupling, supra-graph representations, and multislice formulations, but become substantially more expensive once the temporal dimension is made explicit: supra-graph representations increase the number of node copies with the number of snapshots, and multislice formulations add inter-slice couplings accounted for during optimization.

The two routes operate over GPU-compatible supra-graph representations, in which intra-slice edges encode the topology observed at each snapshot and inter-slice edges encode temporal dependencies between corresponding node copies (Section 2.2). Expressed in this form, the two objectives map onto operations supported by the NVIDIA RAPIDS ecosystem and cuGraph [119]. Implementations are exposed through NetworkX-Temporal [128] with zero-code-change backends: `nx-cugraph`'s Python bindings allow execution to be switched to the GPU by setting a single environment variable, preserving the high-level workflow used for temporal graph construction and analysis.

5.2.1 Implementation details

The spectral route uses a Bethe-Hessian operator (Equation 5.8) compatible with GPU sparse eigensolvers, as the matrix is symmetric and can be decomposed with available GPU routines. Meanwhile, the modularity route optimizes a multislice objective over the supra-adjacency matrix through greedy optimization with Leiden [176]. Both rely on CUDA primitives for spectral decomposition and sparse matrix computations, allowing the execution of temporal clustering objectives to be fully offloaded to the GPU.

[4]The content of this section is based on the preprint: *Accelerating Dynamic Graph Clustering on GPU Architectures with cuGraph.* Passos, N.A.R.A.; Carlini, E.; Trani, S. (2026). Submitted to: the 6th Workshop on Flexible Resource and Application Management on the Edge (FRAME '26), 32nd International European Conference on Parallel and Distributed Computing (Euro-Par '26), 24–28 August, 2026, Pisa, Italy.

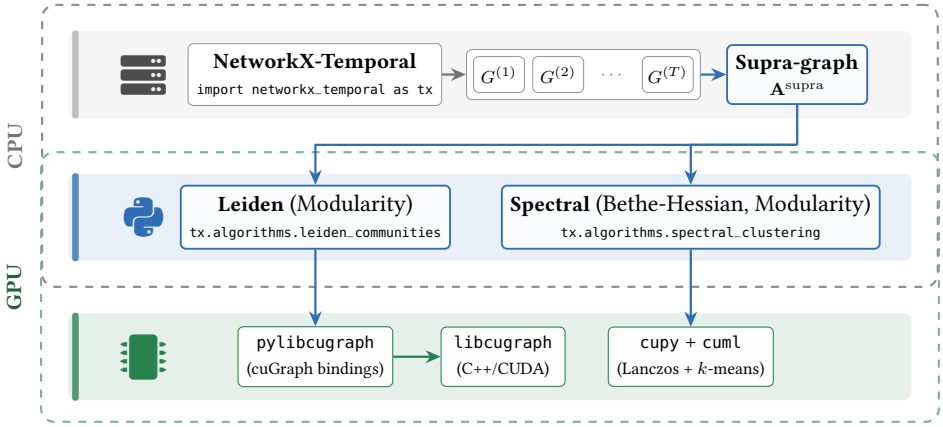


Figure 5.4: Pipeline of the GPU-accelerated backend [147]. A temporal graph is assembled into a supra-graph (Eq. 2.10); spectral routes are available for the Bethe-Hessian and multislice modularity matrices, the latter also solved by Leiden optimization, all fully on-GPU within RAPIDS.

To the best of current knowledge, this is the first RAPIDS/cuGraph backend for temporal community detection, making GPU acceleration available without leaving the temporal graph abstractions presented throughout this thesis, materialized in the software presented in Section 3.1. The recovered communities are accordingly those favored by the spectral or modularity-based objectives, discussed in Section 4.2. Note that these contributions focus on temporal graphs without attribute features (non-annotated), and do not address the integration of node or edge attributes into the optimization objectives.

Spectral clustering via the Bethe-Hessian

The spectral route is motivated by the limitations of ordinary Laplacian spectra in sparse graphs near detectability limits (Section 2.2.2), where non-backtracking operators provide a more robust alternative for recovering assortative community structure.

In spite of the temporal non-backtracking operator being well-defined (Equation 2.11), its direct GPU implementation is impractical in the current ecosystem. Although ARPACK's non-symmetric Arnoldi routines are available in SciPy [183], they require CPU-side iterations of per-step host-device transfers (`matvec` callbacks), which largely defeats the purpose of GPU acceleration due to the introduced repeated synchronization and data transfers between CPU and GPU devices. Moreover, GPU-based eigensolvers are currently restricted to symmetric (Hermitian or positive-definite) matrices, so the asymmetric non-backtracking (Hashimoto-type), which is direction-aware, cannot be directly decomposed with available CUDA primitives. However, the Bethe-Hessian operator is symmetric and retains asymptotic optimal properties, making it a suitable principled alternative that allows the spectral route to be fully executed on GPU.

The asymmetry of the non-backtracking operator arises from its directionality, which may be relaxed for the purpose of clustering[5]. It is therefore possible to reformulate it as a symmetric problem by considering a distinct operator that retains similar spectral

[5]Directionality is important for, e.g., causal inference, where non-backtracking walks encode ordered propagation across time. Such an application is outside the scope of this chapter, but a discussion on the potential of the operator as a *principled primitive* for *pooling* [59] is found in the conclusion (Part IV).

properties. The implementation uses a Bethe-Hessian (Equation 2.12) reformulation,

$$\mathbf{H}^{\text{supra}}(r) = (r^2 - 1)\mathbf{I} - r\mathbf{A}^{\text{supra}} + \mathbf{D}^{\text{supra}}, \tag{5.8}$$

where $\mathbf{A}^{\text{supra}}$ is the supra-adjacency matrix (Equation 2.10), $\mathbf{D}^{\text{supra}}$ is its degree matrix, and r is a regularization parameter. In static graphs, at $r \approx \sqrt{c}$, the number of negative eigenvalues estimates the number of communities [88]; a similar procedure may extend to the temporal case, allowing the GPU-based spectral route to be fully executed on the supra-graph representation, although the optimal choice of r for temporal Bethe-Hessian constructions is not yet fully characterized [29], and likely depends on temporal coupling strength, number of snapshots, and degree heterogeneity of the supra-graph.

This route is presented as an alternative spectral pathway rather than the main object of the runtime evaluation presented next; its systematic assessment across detectability regimes, including the effects of temporal coupling and supra-graph sparsity, remains an ongoing effort. Its role here is to show that a principled spectral formulation can be made compatible with GPU execution in temporal graph settings, yielding a complete GPU-based spectral clustering pipeline that is fully integrated with the temporal graph abstraction used throughout this thesis and the software presented in Section 3.1.

Multislice modularity via spectral approximation

The same supra-graph eigendecomposition admits a second spectral route, targeting the modularity objective directly rather than the detectability regime. Relaxing the assignment matrix $\mathbf{C}$ in Equation 4.11 to real values yields the standard spectral approximation of multislice modularity, whose solution is given by the leading eigenvectors of the multislice modularity matrix $\mathbf{M}_{\text{MS}}$ (Equation 4.10); embeddings of the temporal node copies may then be clustered with k-means, also available on GPU through cuML [120].

This route reuses the same GPU sparse-eigensolver as the Bethe-Hessian formulation, differing only in the operator decomposed and in the relevant end of the spectrum: the informative directions of $\mathbf{M}_{\text{MS}}$ are its most positive eigenvalues, whereas those of $\mathbf{H}^{\text{supra}}(r)$ are its negative ones. It thus provides a spectral counterpart to the greedy Leiden route, optimizing the same multislice modularity objective through eigendecomposition, and integrated with the NetworkX-Temporal software.

Implementation proceeds by enumerating temporal node copies, assembling the supra-adjacency matrix (Equation 2.10) as a compressed sparse row (CSR) matrix, and adding bidirectional inter-slice couplings for nodes shared by consecutive snapshots. Instead of materializing the dense multislice modularity matrix $\mathbf{M}_{\text{MS}}$, the matrix-vector product required by the eigensolver is computed implicitly by rank-one corrections

$$\mathbf{M}_{\text{MS}}\mathbf{x} = \mathbf{A}^{\text{supra}}\mathbf{x} - \sum_{t=1}^{T} \gamma \frac{\mathbf{d}^{(t)}\mathbf{d}^{(t)\top}\mathbf{x}^{(t)}}{2m_t}, \tag{5.9}$$

where $\mathbf{x}^{(t)}$ denotes the slice of $\mathbf{x}$ corresponding to snapshot t. This avoids storing the dense modularity matrix, which would require $\mathcal{O}(N_{\text{supra}}^2)$ memory, and instead only requires $\mathcal{O}(N_{\text{supra}} + \sum_t N^{(t)})$ memory for the sparse supra-adjacency matrix and the degree vectors of each snapshot. The leading eigenvectors are then row-normalized and clustered with k-means, after which temporal node labels (memberships) are obtained.

Multislice modularity via Leiden optimization

The modularity route optimizes the multislice objective of Equation 4.9 on the supra-graph. The supra-adjacency matrix encodes both intra-slice edges and inter-slice couplings of strength ω, so that a single optimization problem is defined over all node-snapshot pairs. Community labels are thereby made comparable across time through the coupled optimization itself, avoiding a separate *post hoc* matching step between independently clustered snapshots, often required in static clustering pipelines.

Optimization is performed by Leiden optimization [176], executed through cuGraph. The method remains heuristic and inherits the usual limitations of modularity optimization, including sensitivity to the resolution parameter γ, the inter-slice coupling ω, and the assumption of assortative structure [43]. Its advantage is computational: Leiden scales to large sparse graphs, and its GPU implementation exploits the parallel graph primitives of the RAPIDS ecosystem, with multi-GPU execution available through Dask-based workload distribution [30]. This makes large-scale temporal modularity optimization practical in settings where CPU execution would otherwise be prohibitive.

Unlike the spectral routes, here presented for completeness, multislice Leiden is the main empirical target of this section. A directed comparison is presented against a CPU reference implementation on the same supra-adjacency representation and under a matched optimization budget, isolating the effect of GPU acceleration on the runtime.

5.2.2 Experimental evaluation and discussion

The experimental evaluation focuses on temporal-versus-temporal runtime comparison: the GPU-based Leiden backend, optimizing a global null model (Eq. 4.5) on the supra-graph as a proxy for Q_{MS} (Eq. 4.9) [119], is measured against its multislice and snapshot CPU reference [176, 177]. Because the implementations differ in stopping criteria, All runs are capped at two optimization passes, matching the `leidenalg` default. The reported runtime therefore measures execution under a fixed, equal-work budget rather than time-to-convergence, allowing for a performance comparison between backends.

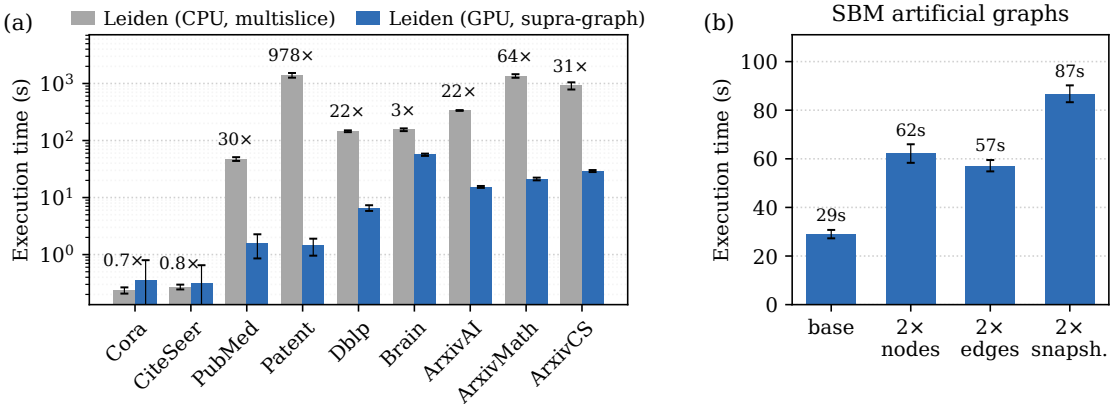


Figure 5.5: Runtime evaluation [147]. (a) Execution time of the GPU supra-graph method against the CPU multislice baseline across real-world datasets (log scale). (b) Effect of independently doubling each axis (nodes, edges, snapshots) on runtime over synthetic SBM instances. Algorithm runs were limited to two iterations only. Results and standard deviation reported over five runs. Experiments conducted on an Intel Xeon Gold 6330 (CPU) and an NVIDIA A100 80GB (GPU).

In general, observed speedups are substantial, but scale-dependent. On large temporal datasets, the GPU backend reduces execution times by one to nearly three orders of magnitude, with the largest gains appearing on datasets whose supra-graph construction and multislice optimization are expensive on CPU. The strongest case is the Patent [68] dataset, where runtime decreases from approximately $1397\,\mathrm{s}$ to $1.4\,\mathrm{s}$, corresponding to a speedup close to $978\times$. Other large-scale graph datasets also show consistent improvements, while dense but low-snapshot graphs yield more modest gains. These results indicate that the advantage of the GPU backend offers a surrogate optimization strategy for temporal community detection workloads in cases it is otherwise intractable.

The evaluation also identifies two boundary regimes. For small static graphs, CPU execution can remain faster because kernel launch and data-transfer overhead dominate the actual computation. At the other extreme, graphs with a very large number of snapshots expose a scaling limit of the supra-graph construction itself: inter-slice couplings grow with the temporal dimension, and graph construction may become the dominant cost even when optimization is GPU-accelerated. Synthetic SBM graphs are therefore used only to characterize scaling behavior along individual axes, such as nodes, edges, and snapshots, rather than to make head-to-head speedup claims against CPU baselines.

Dataset	**GPU supra-graph** `nx-temporal/cugraph`	**CPU multislice** `leidenalg/igraph`	**CPU snapshots** `leidenalg/igraph`
Cora† [197]	0.36 ± 0.44	0.24 ± 0.03	0.17 ± 0.03
CiteSeer† [197]	0.32 ± 0.33	0.27 ± 0.03	0.18 ± 0.01
Brain [141]	56.33 ± 2.60	154.74 ± 8.52	16.99 ± 2.36
Dblp [209]	6.58 ± 0.75	145.49 ± 5.22	7.84 ± 0.99
Patent [68]	1.43 ± 0.47	1396.98 ± 131.81	1.35 ± 0.14
PubMed [146]	1.56 ± 0.71	47.44 ± 3.60	1.20 ± 0.02
School [106]	211.05 ± 11.59	OOT	12.45 ± 0.92
ArxivAI [99]	15.37 ± 0.59	336.31 ± 4.87	17.59 ± 0.52
ArxivCS [99]	29.18 ± 1.18	916.28 ± 131.60	33.51 ± 1.86
ArxivLarge‡ [99]	589.69 ± 209.10	OOT	OOT
ArxivMath [99]	21.19 ± 1.17	1357.72 ± 96.97	43.20 ± 1.78
ArxivPhy [99]	520.93 ± 175.76	OOT	OOT
SBM [129]	29.01 ± 1.73	175.56 ± 11.92	62.28 ± 3.30
SBM-N [129]	62.16 ± 3.82	440.75 ± 52.19	130.89 ± 4.73
SBM-E [129]	57.16 ± 2.33	304.83 ± 18.26	83.21 ± 1.37
SBM-T [129]	86.75 ± 3.47	698.64 ± 73.16	123.67 ± 2.31

Table 5.4: Runtime comparison for temporal Leiden community detection. Values are mean seconds $\pm$ standard deviation over five runs; OOT denotes runs exceeding the 1-hour timeout. All runs use a fixed budget of two optimization passes; GPU times include transfer of the supra-adjacency matrix to device memory. CPU Leiden-over-snapshots results report the cost of independent optimizations on each snapshot without temporal coupling. †: Static graph baselines. ‡: The largest dataset, where a single run of the CPU multislice backend required $21502.9\,\mathrm{s}$ ($\approx 6\,\mathrm{h}$).

Dataset	$\lvert\mathcal{V}\rvert$	$\Sigma(\mathcal{V})$	$\lvert\mathcal{E}\rvert$	$\Sigma(\mathcal{E})$	$\lvert T\rvert$	$\lvert C\rvert$
Cora† [197]	2 708	2 708	5 278	5 278	1	7
CiteSeer† [197]	3 279	3 279	4 552	4 552	1	6
Brain [141]	5 000	60 000	878 207	947 744	12	10
Dblp [209]	28 085	101 797	153 822	222 165	27	10
Patent [68]	12 214	41 529	41 916	41 916	891	6
PubMed [146]	19 717	37 003	44 324	44 324	42	3
School [106]	327	309 006	7 004	188 508	7 375	9
ArxivAI [99]	69 854	142 316	696 819	696 819	27	5
ArxivCS [99]	169 343	374 433	1 157 799	1 157 799	29	40
ArxivLarge [99]	1 324 064	4 998 391	13 649 351	13 649 351	40	172
ArxivMath [99]	270 013	614 578	783 165	783 165	31	31
ArxivPhy [99]	837 212	3 917 817	11 703 425	11 703 425	40	53
SBM [129]	50 000	250 000	1 250 539	1 251 057	5	10
SBM-N [129]	100 000	500 000	2 500 515	2 501 000	5	10
SBM-E [129]	50 000	250 000	2 498 067	2 500 045	5	10
SBM-T [129]	50 000	500 000	2 495 376	2 497 636	10	10

Table 5.5: Dataset properties. $\lvert\mathcal{V}\rvert$ and $\lvert\mathcal{E}\rvert$ denote nodes and edges in the static graph representation; $\Sigma(\mathcal{V})$ and $\Sigma(\mathcal{E})$, the total sum of nodes and edges over time, i.e., counting as many times as they appear across snapshots; $\lvert T\rvert$ and $\lvert C\rvert$ are the number of snapshots and communities. The † marks static graphs used as single-snapshot baselines, commonly found in the literature. Real-world dataset descriptions and neural baselines are provided in the experiments of Section 6.2.2.

Lastly, note that the supra-graph construction still begins on CPU, so reported GPU times include transfer to device memory — a one-time, snapshot-dependent cost that is amortized over optimization passes. Nonetheless, GPU-accelerated optimization offers a scalable algorithmic route for community detection tasks, as the runtime results summarized in Table 5.4, plus the comparison and scaling behavior in Figure 5.5 show.

Additional spectral routes were also tested: on the SBM base graph, spectral modularity required 31.25 ± 1.46 s and Bethe-Hessian required 40.23 ± 1.86 s on GPU. The `igraph` [28] built-in Leiden algorithm was also evaluated as a faster static-snapshot backend, but is not included in the comparison as it does not provide the temporal multislice formulation used by the coupled CPU reference. Note that except for the Bethe-Hessian route, which is optimal in the sparse SBM regime here evaluated, these methods remain restricted to non-attributed graphs, and do not address the integration of node or edge attributes into the optimization objectives. Furthermore, spectral modularity inherits the same assortative assumptions and resolution sensitivity as the standard variant.

For time-varying attributed graphs, the GPU-accelerated multislice Leiden backend provides an algorithmic baseline against which neural methods may be compared, particularly the temporal modularity-based model introduced next. Chapters 6 and 7 provide the learning-based counterpart to this algorithmic perspective: first by introducing the foundations of learning on graphs, and then by developing the neural architecture for temporal community detection that forms the main empirical target of this thesis.

Part III
Learning on Graphs

Theoretical ambition without empirical research
may well be vacuous, but empirical research
without theoretical ambition will be blind.

Ian Shapiro (2002)

Chapter 6

Machine Learning on Graphs

Methods based on artificial neural networks have become central to learning on graph-structured data, contributing to a broader shift from hand-designed graph features toward end-to-end representation learning. In this setting, graph neural networks (GNNs) provide a general framework for learning node-, edge-, and graph-level representations directly from relational (i.e., non-Euclidean) structure and associated attributes — which now constitute one of the dominant paradigms in machine learning on graphs [102, 190].

The motivation for such models is that standard machine learning architectures are not directly suited to graph-structured data. Models designed for images, text, or time series usually rely on regular (Euclidean) domains, where locality, stationarity, and compositionality have relatively well-defined operational meanings. For instance, images are arranged on grids, just like sequences have an ordering; and convolutional or recurrent architectures can exploit these structures through shared filters or ordered transitions.

Graphs, by contrast, have no canonical node ordering, no fixed coordinate system, and no universal notion of spatial proximity: two nodes may be close because they share an edge, a role, a community, a path, or a higher-order relation. This has direct consequences for learning: a model for graph data must be invariant or equivariant to node relabeling, since permuting node identifiers does not change the underlying object. It must also handle irregular neighborhoods, heterogeneous degrees, and dependencies that may be local in graph distance but distant in low-dimensional embedding spaces.

Similarly, while hierarchical organization is common in graphs, it may arise through communities, motifs, functional modules, or nested structures rather than through the fixed spatial composition assumed by convolutional neural networks. These properties define graph learning as a geometric and relational problem, first and foremost — which GNNs address by learning representations through specific aggregation mechanisms.

This chapter reviews the main concepts for machine learning on graphs that substantiate the neural community detection contribution of this thesis. It introduces neural networks for graphs and the message-passing paradigm and discusses common GNN architectures and their limitations, including oversmoothing, oversquashing, scalability, and temporal modeling challenges. The chapter then turns to neural community detection, where learned node representations are used to recover communities through classification, clustering, or differentiable optimization of graph-theoretic objectives. This provides the conceptual bridge to Chapter 7, where temporal graph neural networks are evaluated and extended for community detection on time-varying attributed graphs.

6.1 Neural networks for graphs

Neural networks for graphs inherit the general motivation introduced above: learning must be performed on relational data without relying on a canonical node ordering.

Early machine learning approaches for graphs addressed this problem through feature engineering. Nodes, edges, subgraphs, or entire graphs were represented by explicit descriptors, such as node degree, clustering coefficient, centrality measures, graphlet-degree vectors, neighborhood-overlap scores, or graph- and subgraph-level graphlet[1] frequency distributions [161]. Such descriptors allowed conventional classifiers and clustering algorithms to be applied to graph data, but their quality depended strongly on the chosen features and on the scale of structure they were designed to capture.

Graph representation learning later reduced this dependence on manual descriptors. Skip-gram methods [60, 138, 173] automated part of the feature-extraction process by sampling local neighborhoods and mapping nodes to low-dimensional, real-vector space, so that nodes with similar structural roles or neighborhood co-occurrence patterns were close together. While these functions preserve notions such as local proximity, structural similarity, or higher-order neighborhood co-occurrence, representation learning and downstream tasks were decoupled: embeddings were learned independently of the task at hand, and used as input to a separate classifier or clustering algorithm.

Neural networks designed for graphs [56, 157] marked a further step in this progression by making representation learning task-adaptive. Rather than relying on predefined descriptors or precomputed embeddings, node representations were learned through differentiable aggregation over graph neighborhoods. Topology, node and (when available) edge attributes were then combined within the same trainable encoder, under supervised, semi-supervised, self-supervised, or unsupervised settings. This was a relevant advancement for machine learning on graphs, where the information used for, e.g., classification or regression tasks was learned from both structure and attribute signals.

In temporal graphs, the same principle was extended with mechanisms accounting for evolving structure, time-dependent features, and historical interactions, usually by combining graph encoders with memory, recurrence, attention, or other mechanisms that determine how past information affects current representations [21, 50, 94]. Temporal graph learning therefore adds a second axis of representation learning: the model must aggregate information not only across neighborhoods, but also across time.

Downstream tasks on static and temporal graphs are commonly organized by the level at which predictions are made. Node-, edge-, and graph-level tasks include classification (discrete-valued) and regression (continuous-valued) predictive tasks. In this context, clustering and community detection are the unsupervised counterparts of node-level classification, where the goal is to assign nodes to groups, making community detection a natural application of graph representation learning, where the loss function approximates the underlying community structure by optimizing an end-to-end differentiable (usually spectral) relaxation of a graph-theoretic objective [179].

The following sections introduce the message-passing paradigm underlying most GNN architectures, then review common layer mechanisms and extensions, including graph convolutional, attention-based, higher-order, and temporal message-passing models, with special attention given to their limitations and adaptations to temporal graphs.

[1] Small connected n-node subgraphs that describe a network in terms of local structural patterns.

Message-passing learning paradigm

In general terms, a GNN is operationally defined by local information exchange: node representations are updated by aggregating information from neighboring nodes. This requirement distinguishes graph learning from learning on ordered Euclidean domains, since the model output should not depend on the arbitrary ordering used to index the nodes. Such a requirement is usually fulfilled through the use of permutation-invariant operations, such as sum, mean, or maximum, combined with shared update functions that are equivariant to node relabeling, such as neural networks with shared weights.

The most common formalization is the message-passing paradigm [52]. At layer ℓ, each node i has a hidden representation $\mathbf{h}_i^{(\ell)}$. A message function first computes information exchanged between node i and its neighbors, optionally using edge features $\mathbf{e}_{ij}$, and a permutation-invariant operator aggregates these messages, expressed as

$$\mathbf{m}_i^{(\ell)} = \bigoplus_{j \in \mathcal{N}(i)} \phi\left(\mathbf{h}_i^{(\ell)}, \mathbf{h}_j^{(\ell)}, \mathbf{e}_{ij}\right), \tag{6.1}$$

where $\mathcal{N}(i)$ denotes the neighbors of node i, ϕ is a learnable message function, and $\bigoplus$ is a permutation-invariant aggregation operator, such as sum, mean, or maximum. The node representation is then updated using its previous state and the aggregated message,

$$\mathbf{h}_i^{(\ell+1)} = \psi\left(\mathbf{h}_i^{(\ell)}, \mathbf{m}_i^{(\ell)}\right), \tag{6.2}$$

where ψ is a learnable update function, typically implemented by a neural network followed by a non-linear activation function, such as ReLU [48]; while edge features may encode additional information, including edge weights, signs, labels, and timestamps.

Equations 6.1 and 6.2 describe a broad class of message-passing architectures. Different GNN layers correspond to distinct choices of message, aggregation, and update functions. For example, graph convolutional layers use degree-normalized weighted sums, while attention-based layers learn neighbor-specific weights during aggregation. In supervised or semi-supervised settings, these functions are trained by backpropagation from task labels; in unsupervised or self-supervised settings, they may instead be optimized through reconstruction, contrastive, clustering, or graph-theoretic objectives.

After L layers, the final node embeddings $\mathbf{h}_i^{(L)}$ may be used directly for node-level tasks, such as node classification, anomaly detection, or community assignment. For edge-level tasks, representations are usually obtained by combining the embeddings of the two incident nodes, for example by concatenation, inner products, or element-wise operations, followed by a learnable prediction function. For graph-level tasks, node embeddings are aggregated into a whole-graph representation through a readout function,

$$\mathbf{h}_G = \Psi\left(\bigoplus_{i \in V} \Phi\left(\mathbf{h}_i^{(L)}\right)\right), \tag{6.3}$$

where Φ and Ψ are learnable functions and $\bigoplus$ is again permutation-invariant. This readout may be applied after the final layer, or combined with mechanisms to obtain aggregated representations, such as hierarchical pooling strategies [59, 139], among others.

The depth of a message-passing network controls the size of the receptive field: after L layers, a node embedding may contain information from nodes up to L hops away.

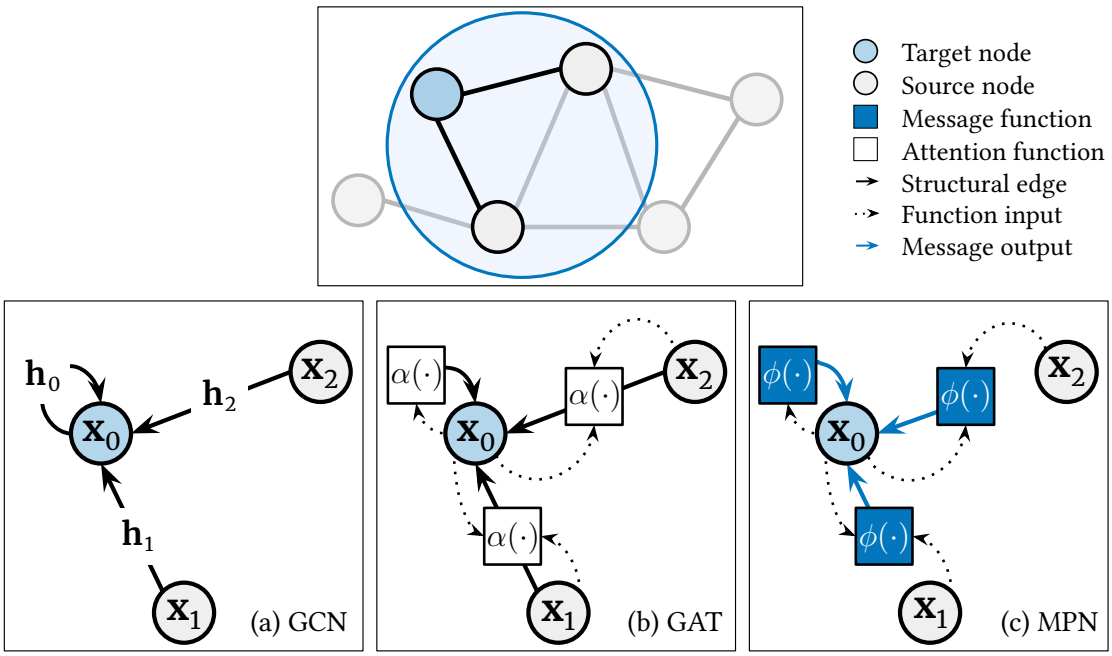


Figure 6.1: Message-passing mechanisms in GNNs. Top: target node and its receptive field in blue. Bottom (left to right): a GCN layer aggregates degree-normalized neighbor representations; a GAT layer learns neighbor-specific weights α that rescale them; and a general message-passing layer applies a learnable function ϕ that synthesizes a new message pre-aggregation [52, 84, 181].

This is useful for incorporating broader structural context, but it also introduces well-known limitations. Repeated aggregation may cause node embeddings to become increasingly similar, reducing their discriminative power, a phenomenon known as *over-smoothing*; conversely, information from many distant nodes may need to be compressed through narrow graph bottlenecks into fixed-size vectors, leading to *oversquashing* [211]. Deep (multilayer) GNNs may also suffer from memory costs and poor scalability, especially when extended to temporal graphs, which require additional mechanisms to handle evolving structure, temporal dependencies, and changing node or edge features.

Several architectural mechanisms have been proposed to mitigate these issues, including residual and jumping-knowledge connections, attention mechanisms, normalization, graph rewiring, sampling strategies, early-exit schemes, and higher-order message-passing operators. The following sections review some of the most common layer mechanisms and extensions, emphasizing their trade-offs in model expressivity and scalability, as well as suitability for downstream node-level clustering tasks.

6.1.1 Graph neural network architectures

GNN architectures may be broadly organized according to the layer mechanisms used for message construction, aggregation, and update. This taxonomy is not exclusive: practical models often combine convolutional, attentional, recurrent, pooling, or higher-order components depending on the task, graph structure, available features, and whether the setting is transductive or inductive. Nevertheless, distinguishing these layer families is useful to understand the assumptions and inductive biases of different GNN architec-

tures, especially their trade-offs in terms of efficiency and suitability for node clustering.

The following subsections present graph convolutional and attentional layers, two common message-passing implementation standards, before moving on to higher-order and temporal extensions addressing locality and dynamicity constraints in the topology.

Graph convolutional layers

Graph convolutional layers generalize the idea of convolution to graph-structured data by aggregating information from neighboring nodes according to the graph topology. In spectral formulations, convolution is defined through the eigenbasis of a graph Laplacian, so that filters operate on graph signals in the graph Fourier domain. Let $\mathbf{A}$ be the adjacency matrix, $\mathbf{D}$ the degree matrix, and $\mathbf{L}$ a graph Laplacian, as introduced in Section 2.2.1. A graph signal can then be filtered by a function of $\mathbf{L}$, but computing such filters exactly requires eigendecomposition of the Laplacian, which is expensive for large graphs and difficult to reuse across different graph structures.

Localized spectral filters reduce this cost by approximating the filter as a polynomial of the Laplacian. Chebyshev networks [33], for example, restrict the filter to a K-hop neighborhood by using a polynomial of degree K. The Graph Convolutional Network (GCN) from [84] further simplifies this construction by using a first-order approximation, corresponding to $K = 1$. In message-passing form, a GCN layer updates node i by aggregating degree-normalized features from its neighbors and itself:

$$\mathbf{m}_i^{(\ell)} = \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{1}{\sqrt{d_i d_j}} \mathbf{W}^{(\ell)} \mathbf{h}_j^{(\ell)}, \tag{6.4}$$

$$\mathbf{h}_i^{(\ell+1)} = \sigma\left(\mathbf{m}_i^{(\ell)}\right), \tag{6.5}$$

where d_i and d_j denote node degrees, $\mathbf{W}^{(\ell)}$ is a learnable weight matrix, and σ is a nonlinear activation function. The inclusion of self-loops allows each node to preserve part of its own representation during aggregation, standard among GCN-based models.

In matrix form, the same propagation rule is commonly written as a single operation,

$$\mathbf{H}^{(\ell+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(\ell)} \mathbf{W}^{(\ell)}\right), \tag{6.6}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ adds self-loops, $\tilde{\mathbf{D}}$ is the corresponding degree matrix, $\mathbf{H}^{(0)} = \mathbf{X}$ is the input feature matrix, and $\mathbf{H}^{(\ell)} \in \mathbb{R}^{|V| \times F^{(\ell)}}$ contains the node embeddings at layer ℓ. The symmetric normalization controls the scale of the aggregation and improves numerical stability when stacking layers. Since the operation is sparse and local, its computational cost scales with the number of edges, making GCNs suitable for large sparse graphs.

Stacking multiple GCN layers increases the receptive field: after L layers, each node representation may contain information from its L-hop neighborhood. This is useful for incorporating broader context, but it also exposes the model to the limitations discussed above, including oversmoothing and oversquashing. Many later variants modify the spectral filter, the normalization, or the aggregation scheme to improve stability, scalability, or expressiveness. Examples include Cayley polynomial filters [92], autoregressive moving-average filters [8], relational GCNs for multi-relational graphs [159], and edge-conditioned convolutions for incorporating edge features [163].

Graph convolutional layers have therefore become a foundational component of GNN architectures. They are efficient, differentiable, and naturally compatible with sparse graph data, but they inherit the assumptions of Laplacian-based smoothing: edges are treated as channels through which similar or compatible representations should be mixed. This makes them effective for many node-, edge-, and graph-level prediction tasks, while motivating alternative mechanisms where neighbor importance, edge semantics, higher-order structure, or temporal dependencies are modeled explicitly.

Graph attentional layers

Graph attentional layers modify the convolutional scheme (Equation 6.6) by learning how much weight should be assigned to each neighbor, that is, their relative importance.

This addresses a main limitation of graph convolutional layers: degree-normalized aggregation treats neighbor contributions according to a fixed rule, determined by the graph topology, rather than by their relevance to the target node or task at hand. Instead, attention mechanisms learn edge-specific coefficients from node features, allowing the model to emphasize different parts of a node's neighborhood during message passing.

The Graph Attention Network (GAT) [181] implements this idea through masked self-attention over graph neighborhoods. At layer ℓ, node features are first linearly transformed, and an attention score is computed for every edge (i, j) by applying a shared attention vector to the concatenated transformed features of the target and source nodes. The coefficients are then normalized over the neighborhood of node i, as defined by

$$e_{ij}^{(\ell)} = \sigma\left(\mathbf{a}^\top \left[\mathbf{W}^{(\ell)}\mathbf{h}_i^{(\ell)} \,\|\, \mathbf{W}^{(\ell)}\mathbf{h}_j^{(\ell)}\right]\right), \tag{6.7}$$

$$\alpha_{ij}^{(\ell)} = \frac{\exp\left(e_{ij}^{(\ell)}\right)}{\sum_{k\in\mathcal{N}(i)} \exp\left(e_{ik}^{(\ell)}\right)}, \tag{6.8}$$

$$\mathbf{h}_i^{(\ell+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(\ell)}\mathbf{W}^{(\ell)}\mathbf{h}_j^{(\ell)}\right), \tag{6.9}$$

where $\mathbf{W}^{(\ell)}$ is a learnable weight matrix, $\mathbf{a}$ is a learnable attention vector, $\|$ denotes concatenation, and σ is a nonlinear activation function, commonly LeakyReLU [103].

Employing a function to exponentiate and normalize $\mathrm{softmax}$ the obtained attention scores for neighbors $k \in \mathcal{N}(i)$ of each target node j ensures that the coefficients $\alpha_{ij}^{(\ell)}$ are positive and sum to unity [11]. This way, the aggregated message (Equation 6.9) becomes a learned weighted combination of neighboring representations, rather than a (fixed) degree-normalized average, as is the case in graph convolutional layers (Equation 6.4).

GAT layers are commonly implemented with multi-head attention. Several independent attention mechanisms are applied in parallel, each with its own parameters, and their outputs are either concatenated or averaged. Concatenation is typically used in intermediate layers to increase representational capacity, while averaging is often used in the final layer to stabilize predictions. Averaging over H heads can be written as

$$\mathbf{h}_i^{(\ell+1)} = \sigma\left(\frac{1}{H}\sum_{h=1}^{H}\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(h)}\mathbf{W}^{(h)}\mathbf{h}_j^{(\ell)}\right), \tag{6.10}$$

where $\alpha_{ij}^{(h)}$ and $\mathbf{W}^{(h)}$ denote the attention coefficients and weight matrix of head h. This mechanism improves stability and allows different heads to attend to different structural or attribute-based aspects of the neighborhood, increasing model expressiveness.

A limitation of the original GAT formulation is that its attention ranking is static in the sense identified by [12]. For a fixed set of transformed neighbor features, the ordering induced by the mechanism is independent of the query node, limiting the ability of the model to express genuinely query-dependent attention patterns. GATv2 addresses this by changing the order of linear transformation and non-linearity, so that attention scores are computed dynamically from the joint representation of the target and source nodes:

$$e_{ij}^{(\ell)} = \mathbf{a}^\top \sigma\left(\mathbf{W}^{(\ell)}\left[\mathbf{h}_i^{(\ell)} \,\|\, \mathbf{h}_j^{(\ell)}\right]\right), \qquad \alpha_{ij}^{(\ell)} = \frac{\exp\left(e_{ij}^{(\ell)}\right)}{\sum_{k\in\mathcal{N}(i)} \exp\left(e_{ik}^{(\ell)}\right)}. \tag{6.11}$$

This allows the relative importance of neighbors to depend more flexibly on the target node, increasing the expressiveness of the mechanism without changing the basic neighborhood-aggregation paradigm, yielding a more powerful aggregation scheme.

Attention-based GNNs are useful when the relevance of neighboring nodes is heterogeneous, e.g., when only a subset of local interactions is informative for prediction or clustering. They also provide an adaptive alternative to fixed convolutional weights and are directly compatible with inductive learning, since attention coefficients are computed from node features rather than from node identities. However, attention does not remove the fundamental locality of message passing: information still propagates through graph neighborhoods layer by layer, and the model remains subject to oversmoothing, oversquashing, and scalability constraints. Moreover, attention coefficients should not be automatically interpreted as explanations of model behavior, since they are learned optimization variables that may not identify semantically meaningful relations.

Such limitations motivate complementary spatial architectures and extensions, including pseudo-coordinate mixture models such as MoNet [109], sampling-based inductive methods such as GraphSAGE [69], and higher-order aggregation strategies such as k-WL GNNs [110] — based on the Weisfeiler–Leman graph isomorphism[2] test, commonly known as 1-WL or *color refinement* (Figure 6.2) — which can distinguish certain non-isomorphic graphs that ordinary message passing cannot. The former are discussed next, since they address a different limitation of ordinary message passing: its bounded ability to distinguish graph structures beyond the expressive power of the 1-WL test.

Higher-order message passing

Higher-order GNN architectures aim to capture structural information that is not fully represented by ordinary node-wise message passing. Instead of exchanging messages only between adjacent nodes, these models propagate information over larger objects, such as k-tuples of nodes, motifs, subgraphs, or other higher-order neighborhoods. The goal is to increase the expressive power of the learned representations, especially in settings where community structure, molecular function, or graph identity depends on patterns that are not reducible to pairwise local aggregation in an efficient manner.

[2] A similar issue (automorphism) is observed at a local level: nodes may receive identical representations under ordinary message passing, even when they play different roles in the global structure [87].

A central motivation comes from the known connection between message-passing GNNs and the Weisfeiler–Lehman (WL) hierarchy. Under standard assumptions, ordinary message-passing GNNs are at most as expressive as the 1-dimensional WL test for graph isomorphism [192]. Consequently, pairs of non-isomorphic graphs that cannot be distinguished by 1-WL may also receive indistinguishable representations from such GNNs, even when their global organization differs. The molecular graphs in Figure 6.2 illustrate this limitation: local aggregation can produce identical node representations for structurally different graphs, because the information available at each node remains indistinguishable under repeated 1-hop refinement. Higher-order GNNs address this limitation by extending the objects over which messages are exchanged, allowing the model to represent dependencies involving multiple nodes simultaneously.

One prominent example is given by k-WL GNNs [110], which generalizes pairwise message passing to k-tuples of nodes. Each hidden representation is associated with a tuple S, and messages are exchanged between neighboring tuples according to a tuple-neighborhood relation. In analogy with Equations 6.1 and 6.2, it can be written as

$$\mathbf{m}_S^{(\ell)} = \bigoplus_{T \in \mathcal{N}(S)} \phi\left(\mathbf{h}_S^{(\ell)}, \mathbf{h}_T^{(\ell)}\right), \tag{6.12}$$

$$\mathbf{h}_S^{(\ell+1)} = \psi\left(\mathbf{h}_S^{(\ell)}, \mathbf{m}_S^{(\ell)}\right), \tag{6.13}$$

where S and T are k-tuples of nodes, $\mathcal{N}(S)$ denotes the neighboring tuples of S, and ϕ and ψ are learnable message and update functions. By operating on tuples rather than individual nodes, the model can encode interactions among multiple nodes and distinguish graph structures that ordinary 1-WL-equivalent message passing cannot.

The gain in expressiveness comes with a clear computational cost. The number of possible k-tuples grows as $\mathcal{O}(n^k)$, making higher-order message passing substantially more expensive than ordinary node-level aggregation. For this reason, higher-order GNNs are particularly useful when complex substructures are central to the task, as in molecular graphs, motif-rich networks, or settings where cycles and multi-node dependencies carry important information. For tasks where local neighborhood information is sufficient, such as many node classification or clustering problems, standard message-passing architectures may remain preferable because they are cheaper, easier to scale, and often generalize well in embedding space, in spite of their limited expressiveness.

Other architectures increase expressiveness without necessarily implementing the full k-WL construction. Motif-based and subgraph-based GNNs aggregate information

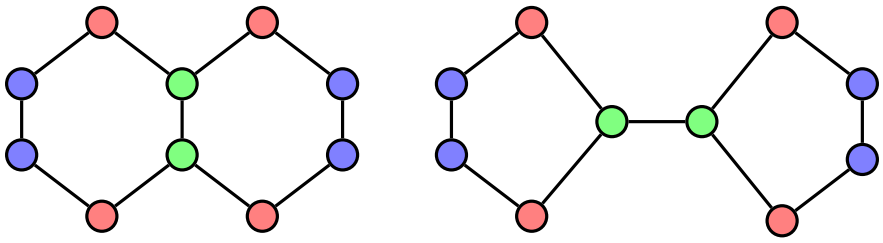


Figure 6.2: Graph representations of decalin (left) and bicyclopentyl (right), two non-isomorphic molecular graphs with the same order and size (10 nodes, 11 edges), but different cycle structure, i.e., closed walks through the graph. Node colors reflect the stable coloring obtained by 1-WL.

over selected structural patterns or sampled subgraphs [2, 195], providing a compromise between ordinary message passing and exhaustive tuple-level propagation. Graph Isomorphism Networks (GINs) [192] follow a related but distinct direction: they strengthen ordinary node-wise aggregation by using injective sum aggregation followed by a multilayer perceptron, making them maximally expressive among 1-WL-equivalent message-passing architectures, but not generally more expressive than 1-WL.

Higher-order message passing therefore improves the ability of GNNs to represent graph structure, but does not remove the broader trade-off between expressiveness and scalability. Designing architectures that capture richer structural dependencies while remaining practical on large graphs remains an active research topic. The temporal domain adds another dimension to this trade-off, since higher-order dependencies may also evolve over time, which temporal message-passing extensions seek to address.

Temporal message passing

When applied to dynamic graphs, GNNs must account not only for relational structure, but also for its evolution over time. This introduces additional modeling and computational challenges: the model must decide how past interactions influence current representations, how long historical information should be retained, and how temporal dependencies should be combined with structural message passing. As a result, issues such as scalability, memory efficiency, and interpretability become more pronounced.

Temporal GNNs address these challenges by augmenting message passing with mechanisms for processing evolving edges, node features, and interaction histories. Early approaches often combined graph convolutional layers with recurrent neural networks, such as Long Short-Term Memory (LSTM) units [50] or Gated Recurrent Units (GRUs) [21], to model dependencies across snapshots. More recent models use temporal attention, memory modules, time encodings, or decay functions to weight historical information according to its relevance to the current prediction time. Broadly, these architectures may be divided into snapshot-based and event-based models — a taxonomy following the dynamic graph representations discussed in Chapter 2 [102].

Snapshot-based models operate on a sequence of graph snapshots, usually obtained by discretizing time into fixed intervals. They are natural extensions of static GNNs, since each snapshot can be processed by a graph encoder and the resulting representations can then be propagated through time. For example, EvolveGCN [126] uses recurrent units to update the parameters of a GCN across snapshots, allowing the graph convolution itself to evolve over time. DySAT [155], instead, combines structural and temporal self-attention: structural attention operates within each snapshot, while temporal attention aggregates node representations across snapshots using positional encodings to represent temporal order. These approaches are often easier to implement and can reuse static GNN layers, where time intervals are treated as discrete steps.

Event-based models operate directly on continuous-time interactions. Rather than aggregating edges into snapshots, they update representations when events occur, making them better suited to irregularly sampled temporal data and interaction streams. For instance, TGN [151] maintains node-level memory states that are updated after events, while TGAT [191] uses temporal attention and time encodings to aggregate information from temporal neighborhoods. Other approaches encode ordered interaction sequences through higher-order projections; for example, De Bruijn GNNs (DBGNNs) [142] repre-

sent temporal dependencies through sequence-based graph constructions that preserve the direction of time — allowing the model to capture patterns in interaction sequences that are suitable, for instance, for link prediction and causal inference tasks.

A related class of models uses temporal point processes to represent the timing of interactions. For instance, HTNE [210] models the intensity of an interaction between nodes i and j at time t [72] as a function of time-dependent embeddings, expressed as

$$\lambda_{i,j}^{(t)} = \exp\left(\mathbf{h}_i^{(t)\top}\mathbf{W}\mathbf{h}_j^{(t)} + b\right), \tag{6.14}$$

where $\lambda_{i,j}^{(t)}$ is the conditional intensity of the interaction, $\mathbf{h}_i^{(t)}$ and $\mathbf{h}_j^{(t)}$ are node embeddings at time t, $\mathbf{W}$ is a learnable weight matrix, and b is a bias (baseline intensity) term.

Temporal attention can also be used to weight past interactions when updating node representations. A generic event-based update can be expressed similar to 6.1 as

$$\mathbf{h}_i^{(t)} = \sigma\left(\sum_{(j,t_j)\in\mathcal{N}(i,t)} \alpha_{ij}(t,t_j)\mathbf{W}\mathbf{h}_j^{(t_j)}\right), \tag{6.15}$$

where $\mathcal{N}(i,t)$ denotes the temporal neighborhood of node i up to time t, and $\alpha_{ij}(t,t_j)$ weights the contribution of an interaction with node j at time t_j. The coefficients may depend on temporal distance, learned attention scores, or decay functions, allowing recent or otherwise relevant events to contribute more strongly to the current representation. Models for temporal community detection build on this idea by combining time-aware embeddings with clustering or community-aware objectives [35].

The distinction between snapshot-based and event-based models reflects a trade-off between temporal resolution and computational complexity. Snapshot-based models are often simpler and compatible with existing GNN layers, but may lose information when events are aggregated into coarse time windows. Event-based models preserve finer temporal order and irregular timing, but require memory management, temporal neighborhood sampling, and careful control of computational cost. Other temporal architectures, such as VGRNN [67] and ROLAND [198], further combine recurrent, variational, or hierarchical state updates to model temporal variation in node representations.

Despite these developments, temporal GNNs remain difficult to evaluate for community detection. Many architectures are designed for link prediction, forecasting, or representation learning, rather than for node clustering on dynamic graphs. Their learned embeddings may capture useful temporal patterns, but the resulting communities often require post hoc clustering or indirect evaluation. Moreover, temporal mechanisms can reduce interpretability, since it becomes difficult to isolate whether a detected group is driven by topology, attributes, recent events, long-term memory, or temporal attention weights. This gap motivates the focus of this thesis on benchmarking and extending temporal GNNs for community detection on time-varying attributed graphs.

6.2 Community detection with neural networks

Despite the aforementioned challenges, neural networks for graphs remain a promising direction for community detection, as they can learn representations that integrate structural, attribute, and temporal information in a single end-to-end differentiable model.

Integrating graph-theoretic objectives, such as modularity or conductance, into the training of a neural network allows the model to learn embeddings that are optimized for community structure, rather than for generic node classification or link prediction tasks. For time-varying graphs, this integration is particularly valuable, as it allows for improved expressiveness by jointly modeling spatial, temporal, and feature signals, in spite of relatively fewer models specifically designed for the task [102].

Within this scope, *neural community detection* refers to the recovery of node groups in real-dimensional spaces obtained from graph data through a parametric mapping — with the caveat that the mapping is learned by optimizing a graph-theoretic objective or a differentiable relaxation thereof. This setting is distinct from the inferential framework of probabilistic block models, which define communities as latent variables in a generative process, and may be best understood as a descriptive method instead[3] [135].

Node embeddings may be assigned distinct community labels through two main strategies. The first is applying a clustering algorithm such as k-means in the embedding space — this strategy is employed by spectral clustering techniques, usually on the eigenvectors of a graph Laplacian [118], and by autoencoder-based models that reconstruct graph structure from learned embeddings [185][4]. The second is to let the model output a fixed-dimensional assignment matrix, whose rows represent community scores or probabilities [179]. Hard assignments may then be obtained by selecting the largest probability (argmax), while soft assignments preserve membership weights and support overlapping structures. Both strategies reduce community detection to a clustering problem, but differ in whether the notion of community is learned implicitly through the geometry of the embedding space or explicitly through a post-processing step.

A further distinction concerns the objective being optimized. Some neural models reconstruct the graph, as in autoencoder-based approaches, and then cluster the learned embeddings. Others optimize self-supervised, contrastive, or information-theoretic objectives that encourage structurally related nodes to have similar representations. A third family directly optimizes differentiable relaxations of graph-theoretic quality functions, such as modularity, thereby improving comparability with established community detection methods while retaining the representational capacity of neural encoders. This last direction is particularly relevant to this thesis because it allows community detection to be trained end-to-end without requiring observed metadata labels as supervision.

The term *community* must therefore be used with care in the neural setting. In an inferential model, it is defined by the role it plays in the data-generating process; in a neural model, it is usually defined by the geometry of the learned representation considering a specific objective. This distinction is of especially importance to avoid conflating community detection with metadata prediction, as embeddings may encode recent interactions, temporal proximity, or feature similarity rather than solely mesoscale structure.

This section discusses and provides a performance evaluation of algorithmic and neural methods for community detection, on real-world and synthetic benchmarks. It then proceeds continues with discussion of ground truths, metadata, and evaluation protocols for real-world networks — of special relevance to temporal graphs and attributed networks, in special for GNN architectures, whose learned representations may encode both spatio-temporal and feature information for downstream clustering tasks.

[3] See Section 4.1 for a discussion of the descriptive-inferential distinction in community detection.

[4] See Chapters 4.2 and 3.2 for discussions on spectral theory and graph autoencoders, respectively.

6.2.1 Neural methods for community detection

Community detection with neural networks is a relatively recent area of research, often found in literature as deep[5] community detection or deep graph clustering [100, 165, 171]. Although many GNN architectures have been adapted to unsupervised node clustering tasks [52, 69, 181] — modifying the aggregation scheme (Figure 6.1), the readout layer, the training objective, or the way learned embeddings are post-processed — most models were, however, not designed to optimize graph-theoretic objectives. These strategies thus greatly differ in how explicitly the notion of community is encoded by the model, yielding distinct families of approaches, which may be distinguished as follows.

Among the earliest neural approaches are autoencoder-based architectures that reconstruct graph structure from learned node embeddings [7]: an encoder maps nodes to (latent) representations, while a decoder attempts to reconstruct the adjacency matrix. The learned embeddings are then clustered, or jointly regularized by a clustering objective. For instance, DAEGC [185], introduced in Section 3.2, uses a graph-attention encoder related to Equation 6.10 and reconstructs the adjacency matrix by inner products between node embeddings[6]. Its loss combines an adjacency-reconstruction term, i.e., binary cross-entropy, with a clustering term based on the Kullback–Leibler divergence between soft assignments and an auxiliary target distribution. The reconstruction term encourages embeddings to preserve graph structure, while the clustering term sharpens the assignment distribution in the embedding space. However, the model is not differentiable with respect to the clustering step, which is performed as a post-processing operation (k-means) to obtain hard assignments every few epochs, and dense adjacency reconstruction can lead to quadratic memory and time costs on large graphs. Sparse reconstruction losses, graph-regularized clustering objectives, and variational autoencoder formulations, such as VGAE [83], have been proposed to improve on scalability.

A second family of models incorporates community detection more directly into the training process through self-supervised or clustering-aware objectives. For instance, SeCOMM [4] combines a GCN-based encoder, as in Equation 6.6, with a self-supervised contrastive loss and a self-expressive community-detection loss. The contrastive component encourages connected or structurally similar nodes to have similar embeddings and pushes dissimilar nodes apart, while the self-expressive component promotes cluster structure in the learned representation space. The model further uses high-confidence similarities as soft 'must-link' and 'cannot-link' constraints, controlled by threshold hyperparameters. The output layer then produces membership probabilities with a $\mathrm{softmax}$, allowing the clustering objective to be optimized together with the encoder by backpropagation. This moves the clustering step closer to the neural model itself; but the result still depends on chosen similarity thresholds and assumptions encoded by the self-supervised objective, which need not align with predefined community notions.

A third line of work optimizes continuous relaxations of graph-theoretic objectives in an end-to-end differentiable manner. This direction is particularly relevant for community detection, as it connects graph representation learning with quality functions already used by established methods, such as modularity, conductance, or normalized cut (Equations 4.5, 4.1, and 2.6, respectively). Deep Modularity Network (DMoN) [179] is a

[5]The term is loosely employed for *deep learning*, even on shallow architectures, and conflates with earlier literature where it strictly defined groups located *deep within* a noisy topological structure [19].

[6]See Section 3.2 for experimental results on deep attentional *augmented* graph clustering with [185].

representative example of this class of models, which employs a (multilayer) GCN-based encoder to produce a soft community assignment matrix and optimizes a differentiable relaxation of modularity [116], with an added regularization term to prevent the trivial solution of assigning all nodes to a single community[7]. Its loss can be written as

$$\mathcal{L}_{\text{DMoN}} = -\frac{1}{2m}\operatorname{Tr}\left(\mathbf{C}^\top \mathbf{M}\mathbf{C}\right) + \vartheta\left(\frac{\sqrt{k}}{n}\left\|\sum_i \mathbf{C}_i^\top\right\|_F - 1\right), \tag{6.16}$$

where $\mathbf{C} \in \mathbb{R}^{n\times k}$ is the soft community-assignment matrix, $\mathbf{M}$ is the modularity matrix defined in Equation 4.7, m is the number of edges, n is the number of nodes, k is the number of communities, ϑ controls the strength of the regularization term, and $\|\cdot\|_F$ denotes the Frobenius norm, i.e., the square root of the sum of squared entries. While the first term corresponds to a relaxed modularity objective in the continuous domain, related to the spectral formulation in Equation 4.8, the second term encourages balanced community assignments, resembling ratio-cut minimization in Equation 2.7.

More recent work has proposed alternative regularization strategies to preserve assignment diversity by combining distance-, variance-, and entropy-based penalties [154], integrating contrastive learning techniques [101], or decomposing the modularity matrix $\mathbf{M}$ into submatrices to capture multiscale group structures [54]. However, these extensions introduce additional hyperparameters and complexity, reliance on mini-batch sampling that is not guaranteed to preserve global community structure, and potential trade-offs between group fairness and overall modularity, which may complicate practical implementation and evaluation. Still, the modularity-optimizer objective of DMoN provides a framework for neural community detection that is aligned with established graph-theoretic objectives and compatible with the expressive power of GNN encoders.

Temporal neural methods for community detection

A growing number of GNN architectures have been adapted to produce time-aware node embeddings, and a few couple a neural community-detection objective to a time-aware model. To the best of our knowledge, however, none optimize a temporal graph-theoretic objective end to end: observed methods operate instead on individual snapshots and introduce the temporal domain architecturally, through recurrence and/or auxiliary penalties in the loss function, rather than optimizing for some notion of *dynamic* community.

The most common pattern is to embed DMoN as a component of a larger temporal pipeline. The model has been applied per snapshot to estimate modularity, whose shifts over time serve as a univariate signal for change point detection in highly attributed dynamic networks [137]; benchmarked as one of several clustering-based graph-pooling losses inside spatio-temporal GNNs for time-series clustering [70]; and used as a graph-clustering module, yielding anatomically coherent embeddings over an event-derived directed acyclic graph [3]. However, in each of these cases, the temporal domain is external to the community-detection objective, and the model remains confined to operating on discrete time intervals to produce time-aware embeddings on segmented snapshots.

Closest to the *dynamic* community detection setting is DynMoCo [105], an end-to-end framework designed for learning on molecular graphs, where a recurrent encoder produces time-dependent community assignments along trajectory dynamics. The

[7] See Section 4.2.2 for a spectral interpretation. — The objective's global maximum if $\vartheta = 0$ (Eq. 6.16).

model introduces a GCN-RNN/DMoN hybrid architecture, where the GCN encoder processes each snapshot and the RNN captures temporal dependencies between snapshots. The loss optimizes a sum of per-snapshot (static) modularity scores, including a domain-knowledge term and a smoothness penalty $\sum_t \|\mathbf{S}_t - \mathbf{S}_{t-1}\|_F$, where $\mathbf{S}_t$ is the soft assignment matrix at time t, resembling the temporal modularity couplings and smoothness penalty in Equations 4.9 and 4.12. However, the objective itself is still time-agnostic: it relies on the domain-informed architecture and temporal penalty computed per-snapshot to produce coherent membership assignments over time. These methods may therefore be understood as pairings of static modularity (Equation 4.5) with a snapshot-based architecture, rather than as optimizers of a temporal graph-theoretic objective.

In another direction, a representative line of work models interactions as continuous-time point processes, often using Hawkes-type dynamics [72] to encode the effect of past events on future interactions. For example, TGC [100] extends HTNE [210] by incorporating temporal attention mechanisms that weight past interactions when updating node embeddings. A simplified decay-based attention coefficient may be expressed as

$$\alpha_{ij}(t, t_j) = \frac{\exp\left(-\beta(t - t_j)\right)}{\sum_{k \in \mathcal{N}(i,t)} \exp\left(-\beta(t - t_k)\right)}, \tag{6.17}$$

where β is a (learnable) parameter controlling how strongly older interactions are decayed. Such mechanisms allow recent interactions to contribute more strongly to the current embedding, making the model responsive to changes in graph structure and group membership over time. In the case of TGC, the network is represented as a continuous-time temporal graph (Equation 2.3), abstracting away from discrete snapshots and allowing the model to capture fine-grained temporal dynamics. However, its learning step is decoupled from graph-theoretic objectives: the model learns time-aware embeddings and clusters them indirectly, based on a post hoc step (e.g., k-means), rather than directly optimizing a graph-theoretic objective in an end-to-end manner.

Building on this gap, Chapter 7 introduces a neural architecture that extends modularity optimization to continuous-time graphs by incorporating a Hawkes-Laplacian operator that weights historical information according to its relevance to the current interaction time. By combining topology, node/edge attributes, and temporal dependencies, the model aims to recover communities whose structure changes over time, while remaining trainable through an unsupervised graph-theoretic objective and maintaining the same complexity as the static DMoN model. Architectural details and empirical evaluation of the model versus its baseline are presented in Chapter 7, while the next sections present an experimental evaluation of the above methods and discuss why ground truths and benchmarking protocols are especially delicate for neural community detection — in special for real-world graphs, both in the static and temporal graph regimes.

6.2.2 Experimental evaluation of models[8]

A first study found that augmenting neural models with temporal structure did *not* consistently improve the recovery of ground-truth communities over algorithmic baselines

[8]Based on: *Deep Community Detection in Attributed Temporal Graphs: Experimental Evaluation of Current Approaches*. Passos, N.A.R.A.; Carlini, E.; Trani, S. (2024). In Proceedings of the 3rd Graph Neural Networking Workshop (GNNet '24), 20th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '24), Association for Computing Machinery, New York, NY, USA, 1–6.

[145]; a finding later corroborated on synthetic graphs sampled, under known ground truth, from the time-varying, attributed, degree-corrected stochastic block introduced in Section 5.1 [129]. This, however, amount less to a verdict against neural methods than to a relocation of the problem: the challenge is not that neural models are inaccurate, but that they are oftentimes difficult to train and evaluate in a likewise principled manner.

For non-attributed temporal community detection, principled algorithmic methods remain the most employed method, where the decisive obstacle is their scalability rather than their accuracy; the case for neural models is strongest precisely in the attributed regime that descriptive spectral and modularity [116] methods cannot exploit, and even there their gains are clearest where structural, attribute, and temporal signals align, rather than as a universal advantage [135]. Notwithstanding, the evaluation of neural models — as with other methods beyond the paradigm — is complicated by the lack of principled ground truths, often replaced by observed metadata that may not reflect the underlying mesoscale structures [131]. This section presents an experimental analysis on real-world and synthetic graphs, including algorithmic baselines and neural models, highlighting the challenges of benchmarking and the need for careful interpretation of the results, according to distinct evaluation metrics and dataset domains [132].

Experimental evaluation on real-world datasets

Models were trained in a transductive learning setting (Figure 3.7), where the full graph is available [100], and the best epoch on accuracy was selected for their evaluation.

Six real-world temporal graphs from different domains were considered, whose main characteristics are reported in Table 5.5. *ArxivAI* [99] is a citation network of papers in Artificial Intelligence; *Brain* [141] is a network of human brain regions connected by functional connectivity; *Dblp* [209] is a co-authorship network in Computer Science; *Patent* [68] is a citation network of patents from the US Patent Office; *PubMed*[9] [146] is a citation network of biomedical papers in diabetes research; and *School* [106] is a contact network of interactions between students in a high school. All datasets are publicly available and have been used to evaluate algorithmic and static/temporal GNN-based models on node-level clustering tasks [100, 145] as a proxy for community detection. Static citation datasets in Table 5.5 (*Cora*, *Citeseer*) were not considered due to their small size, although extensively studied in the literature; while larger citation networks, such as *ArxivPhy* (representative of the same domain as *ArxivAI*), were too large to be processed efficiently by all methods, and were thus excluded from this evaluation.

Following [100], node features for all datasets except *PubMed* were generated by Node2Vec pretraining [60], using unbiased random walks ($p = q = 1$), 10 walks per node, walk length 80, and 128 feature dimensions. For *PubMed*, node-level features are given by the most relevant terms in paper abstracts, extracted with a TF-IDF vectorizer and reduced to 500 dimensions [113, 197]. Before Node2Vec pretraining, graphs were preprocessed by removing isolated nodes and self-loops; the resulting embeddings were then normalized to zero mean and unit variance along each feature dimension.

Results show that no model class dominates across datasets. When evaluated with chance-adjusted metrics — adjusted Mutual Information (AMI) and adjusted Rand index

[9] See Section 3.1.2 for a discussion on this dataset and its extension to the temporal domain. — Feature generation with Node2Vec for other datasets restricted the inductive evaluation to PubMed only in [145].

(ARI) — GNN-based methods are competitive on some graphs, but simpler topology-based baselines remain stronger on others. DMoN yields the highest AMI on *ArxivAI*, TGC [100] performs best on *Dblp*, and DAEGC [185] the best mean scores on *PubMed*, although with comparatively high variance. By contrast, spectral clustering is strongest on *Brain* and *Patent*, despite being applied to the aggregated graph and therefore ignoring temporal information, while Node2Vec remains strongest on *School*. These results suggest that the benefit of neural architectures depends strongly on whether graph structure, node features, and temporal dynamics provide complementary signals.

A careful reading along each metric reveals more than this initial ranking, and illustrates the broader difficulty of evaluating community detection against metadata labels [131]. On *ArxivAI*, for instance, DMoN yields low accuracy but the highest mean AMI score ($.372 \pm .008$). This suggests that it recovers a partition sharing information with the metadata labels, but not one that aligns well under one-to-one class matching. In practice, this may occur when the recovered groups split or merge metadata classes, or when their size distribution differs from the observed labels. By contrast, Node2Vec and TGC obtain the highest ARI and near-best AMI, at the lower end of DMoN's uncertainty range. Their partitions are therefore more consistent at the pairwise co-membership level: nodes assigned to the same metadata class are more often clustered together, and nodes assigned to different classes are more often separated. Other methods show different weaknesses: Spectral clustering reaches moderate accuracy, but almost zero AMI and ARI, indicating its apparent class agreement largely disappears under chance-corrected label and pairwise agreement measures. DynNode2Vec [104] performs poorly on all three metrics — its strategy of sequentially initializing each snapshot from previously learned embeddings does not yield partitions aligned with one-to-one class matching (accuracy after Kuhn-Munkres correction) [86], shared metadata information (AMI), or pairwise co-membership (ARI). Although tNodeEmbed [164] obtains the highest accuracy ($.673 \pm .001$) by aligning snapshot embeddings through orthogonal transformations, its lower AMI and ARI relative to static Node2Vec indicate that its temporal strategy improves label matching without improving the underlying clustering structure.

	k-means	Spectral	Leiden	Node2Vec	DynNode2Vec	tNodeEmbed	DAEGC	DMoN	TGC
ArxivAI	–	.016	.302	.363	.001	.299	OOM	**.372**	.363
Brain	–	**.498**	.470	.466	.049	.434	.431	.466	.497
Dblp	–	.008	.302	.351	.006	.345	.344	.281	**.355**
Patent	–	**.365**	.203	.270	.139	.248	.315	.195	.329
PubMed	.312	.162	.256	.289	.192	.297	**.320**	.202	.254
School	–	.940	.911	**.998**	.025	.025	.994	.964	.994

Figure 6.3: Heatmap of transductive baseline performance. Cells report mean Adjusted Mutual Information (AMI); darker shades indicate stronger agreement with metadata labels; best result per dataset in bold. OOM marks out-of-memory runs on an NVIDIA A100 GPU (80 GB). Mean and standard deviation for accuracy, AMI, and adjusted Rand index are reported in Table 6.1. *k*-means reported only if node features are available, otherwise obtained by Node2Vec pretraining.

Dataset	Model	Accuracy	AMI	ARI
ArxivAI [99]	Spectral	.389 ± .002	.016 ± .008	.012 ± .006
	Leiden	.525 ± .033	.302 ± .027	.239 ± .031
	Node2Vec	.646 ± .001	.363 ± .001	**.404 ± .001**
	DynNode2Vec	.268 ± .001	.001 ± .001	.000 ± .001
	tNodeEmbed	**.673 ± .001**	.299 ± .001	.312 ± .001
	DAEGC	OOM	OOM	OOM
	DMoN	.319 ± .016	**.372 ± .008**	.196 ± .011
	TGC	.646 ± .001	.363 ± .001	**.404 ± .001**
Brain [141]	Spectral	**.485 ± .001**	**.498 ± .001**	**.320 ± .001**
	Leiden	.404 ± .011	.470 ± .016	.300 ± .017
	Node2Vec	.452 ± .002	.466 ± .001	.270 ± .001
	DynNode2Vec	.163 ± .002	.049 ± .001	.019 ± .001
	tNodeEmbed	.417 ± .002	.434 ± .001	.243 ± .002
	DAEGC	.414 ± .010	.431 ± .009	.253 ± .010
	DMoN	.397 ± .007	.466 ± .005	.269 ± .003
	TGC	.434 ± .001	.497 ± .003	.288 ± .003
Dblp [172]	Spectral	.289 ± .001	.008 ± .001	.000 ± .001
	Leiden	.400 ± .013	.302 ± .003	.187 ± .005
	Node2Vec	.466 ± .001	.351 ± .001	.207 ± .001
	DynNode2Vec	.155 ± .002	.006 ± .001	.002 ± .001
	tNodeEmbed	.451 ± .001	.345 ± .001	.203 ± .001
	DAEGC	.465 ± .010	.344 ± .004	**.219 ± .004**
	DMoN	.379 ± .056	.281 ± .046	.161 ± .037
	TGC	**.471 ± .001**	**.355 ± .001**	.209 ± .001
Patent [68]	Spectral	**.575 ± .011**	**.365 ± .023**	**.319 ± .046**
	Leiden	.431 ± .026	.203 ± .016	.140 ± .027
	Node2Vec	.400 ± .020	.270 ± .022	.171 ± .026
	DynNode2Vec	.354 ± .038	.139 ± .045	.089 ± .042
	tNodeEmbed	.424 ± .048	.248 ± .024	.177 ± .035
	DAEGC	.462 ± .029	.315 ± .052	.271 ± .052
	DMoN	.349 ± .051	.195 ± .064	.136 ± .058
	TGC	.503 ± .019	.329 ± .027	.272 ± .035
PubMed† [146]	k-means	.595 ± .001	.312 ± .001	.281 ± .001
	Spectral	.593 ± .003	.162 ± .008	.143 ± .004
	Leiden	.633 ± .018	.256 ± .020	.245 ± .033
	Node2Vec	.687 ± .001	.289 ± .001	.312 ± .001
	DynNode2Vec	.543 ± .001	.192 ± .001	.135 ± .001
	tNodeEmbed	.673 ± .001	.297 ± .001	.307 ± .001
	DAEGC	**.713 ± .040**	**.320 ± .042**	**.339 ± .065**
	DMoN	.579 ± .059	.202 ± .019	.176 ± .026
	TGC	.677 ± .001	.254 ± .002	.280 ± .002
School [106]	Spectral	.967 ± .001	.940 ± .001	.933 ± .000
	Leiden	.851 ± .023	.911 ± .008	.843 ± .016
	Node2Vec	**.999 ± .002**	**.998 ± .004**	**.998 ± .004**
	DynNode2Vec	.200 ± .004	.025 ± .002	.013 ± .001
	tNodeEmbed	.200 ± .002	.025 ± .001	.013 ± .001
	DAEGC	.997 ± .006	.994 ± .009	.993 ± .010
	DMoN	.983 ± .002	.964 ± .003	.961 ± .004
	TGC	.997 ± .001	.994 ± .001	.994 ± .001

Table 6.1: Real-world graph baselines. Values are mean ± standard deviation over 5 runs. Bold and underline mark the highest and second-highest means; OOM denotes out-of-memory runs on NVIDIA A100 GPU (80 GB). Unless marked by †, features are obtained by Node2Vec pretraining.

Across the remaining datasets, the multi-metric comparison shows that performance rests on the interplay between model assumptions and graph structure, attributes, and temporal dynamics. On *Brain*, spectral clustering obtains the best scores on all three metrics, while TGC is nearly tied on AMI and Leiden remains competitive on ARI, suggesting that the aggregated topology already contains a strong community signal. On *Dblp*, TGC gives the best ACC and AMI, whereas DAEGC gives the best ARI; however, Node2Vec and tNodeEmbed remain close, indicating that random-walk proximity already captures much of the metadata-aligned structure. On *Patent*, spectral clustering again dominates across metrics, with TGC second-best on all three, which points to a topology-driven partition for which temporal and neural components provide limited additional benefit. On *PubMed*, DAEGC achieves the highest mean scores, but with substantially larger variability, while k-means, Node2Vec, and tNodeEmbed remain competitive; this suggests that the original node features are informative, but also that reconstruction-based neural clustering is less stable. Finally, on *School*, static Node2Vec performs best across all metrics, followed closely by DAEGC and TGC, whereas DynNode2Vec and tNodeEmbed collapse to low scores. This is consistent with continuous-time data inducing sparse snapshots, in which temporal random walks provide less reliable context than the aggregated static graph. Meanwhile, modularity-based methods (Leiden and DMoN) remain competitive, highlighting that such objectives can remain effective when the empirical domain matches their underlying assumption of assortative community structure, as expected for repeated interactions among high school students.

This evaluation illustrates the challenges of benchmarking neural community detection methods on real-world datasets. It is, however, not meant to be exhaustive; the limited availability of temporal graphs with known ground truths, the diversity of graph structures and attribute distributions, and the sensitivity of neural models to hyperparameter choices all contribute to the difficulty of drawing general conclusions about model performance. Synthetic generators are therefore useful because they provide controlled experimental conditions for benchmarking and ablation studies.

Experimental evaluation on synthetic datasets

Following the same evaluation procedure, this section presents results for the small-scale graphs created with TADC-SBM (Section 5.1.2), which allow to assess model performance under varying levels of community stability and community-aligned features with known ground truths. Table 5.2 summarizes their characteristics, and Figure 5.3 shows baseline performance on the aggregated graph's structure and feature signals.

The most informative regime is $\text{SBM}_{\eta=.75}$, where communities are no longer static but still retain enough temporal persistence for recovery to be meaningful. In this setting, the methods separate sharply: TGC obtains the best scores on all metrics ($.681 \pm .005$ accuracy, $.434 \pm .006$ AMI, and $.415 \pm .007$ ARI), k-means remains the strongest non-temporal baseline, and DAEGC is competitive but less stable. By contrast, Spectral, Leiden, DMoN, and the random-walk-based encoders fall substantially below their static-community performance. This transition from $\eta = 1$ to $\eta = .75$ is therefore the key result of the synthetic evaluation: a small amount of community change ($\approx 25\%$ nodes transitioning on each snapshot) is sufficient to break methods whose objective is static or whose embeddings summarize neighborhoods without explicitly emphasizing recent interactions. At the same time, the performance attained by DAEGC and TGC on $\eta < 1$

suggest that when the attribute signal remains informative, temporal attention mechanisms can exploit it to increase model expressiveness when topology alone is insufficient.

A careful reading of the three metrics reinforces this interpretation. For $\eta = 1$, Spectral, DAEGC, and DMoN recover the planted partitions, while Leiden remains competitive. The `leidenalg` [176, 177] implementation does not allow fixing a number of communities for heuristic optimization, which was partially circumvented by setting the initial node memberships uniformly-at-random to a discrete interval $[0, k-1]$; however, aggregations yielding higher local optima may still result in a different number of communities, which is reflected in the lower scores obtained by the method. Note that Leiden$_S$ optimization is reported on the aggregated graph (static modularity); whereas supra-graph (multislice modularity) allowed ground truth recovery for $\eta = 1$ (Table 6.3).

For $\eta < 1$, the chance-adjusted metrics separate the methods more clearly than accuracy alone: TGC is the only method that consistently improves upon the k-means attribute-only reference, merit of the pipeline employing an ad hoc clustering step on the learned embeddings. DAEGC remains the strongest alternative neural baseline in such regimes, attaining a comparatively lower performance decrease than DMoN ($\eta = 0.75 \to 0.5$), possibly due to the use of a K–polynomial Laplacian of degree $K = 2$ in the encoder (Equation 3.1) in the author's default parameterization: in the regimes where a fraction $\eta < 0.5$ of the nodes transition between communities, the 2-hop neighborhood thus allows the model to capture second-order degree information corresponding to a fraction $\eta \geq 0.5$ of the nodes that have not transitioned, yielding more stable embeddings and clustering. Results for DMoN in Table 6.2 and Figure 6.2 are also reported considering a two-layer GCN strategy (with $L = [256, 128]$ embedding dimensions); experiments using a single-layer encoder reinforce this interpretation, yielding lower scores on all metrics ($.918 \pm .005$ accuracy, $.813 \pm .011$ AMI, $.815 \pm .011$ ARI). Meanwhile, the very low AMI and ARI of SkipGram-based methods (Node2Vec, Attri2Vec, DynNode2Vec, and tNodeEmbed) indicate that sampled proximity, even when augmented with attributes or temporal alignment, does not recover the planted temporal communities in this benchmark. The regimes analyzed are therefore particularly challenging for neural methods, where temporal adaptations of random-walk and message-passing encoders are insufficient to recover the planted communities, even if the attributes are kept static.

	k-means	Spectral	Leiden$_S$	Node2Vec	Attri2Vec	DynNode2Vec	tNodeEmbed	DAEGC	DMoN	TGC
SBM$_{\eta=1}$	.400	**1.000**	.945	.066	.066	.060	.066	**1.000**	**1.000**	.438
SBM$_{\eta=.75}$	.400	.152	.132	.023	.026	.012	.026	.356	.039	**.434**
SBM$_{\eta=.5}$	.400	.025	.019	.007	.005	.005	.005	.218	.007	**.432**
SBM$_{\eta=.25}$	.400	.006	.010	.002	.002	.006	.002	.151	.002	**.431**
SBM$_{\eta=0}$	.400	.017	.008	.004	.004	.005	.004	.173	.002	**.433**

Figure 6.4: Heatmap of synthetic graph baselines. Cells report mean Adjusted Mutual Information (AMI); darker shades indicate stronger agreement with planted communities. Best result per graph in bold, where node transition probability is $1 - \eta$. Mean and standard deviation for accuracy, AMI, and adjusted Rand index reported in Table 6.2. All datasets share the feature set.

Dataset	Model	Accuracy	AMI	ARI
$SBM_{\eta=1}$	k-means	.648 ± .016	.400 ± .015	.375 ± .018
	Spectral	**1.000 ± .000**	**1.000 ± .000**	**1.000 ± .000**
	$Leiden_S$	.849 ± .055	.945 ± .022	.848 ± .048
	Node2Vec	.216 ± .000	.066 ± .000	.041 ± .000
	Attri2Vec	.216 ± .000	.066 ± .000	.041 ± .000
	DynNode2Vec	.213 ± .001	.060 ± .002	.037 ± .001
	tNodeEmbed	.216 ± .000	.066 ± .000	.041 ± .000
	DAEGC	**1.000 ± .000**	**1.000 ± .000**	**1.000 ± .000**
	DMoN	**1.000 ± .000**	**1.000 ± .000**	**1.000 ± .000**
	TGC	.687 ± .004	.438 ± .005	.421 ± .005
$SBM_{\eta=.75}$	k-means	.648 ± .016	.400 ± .015	.375 ± .018
	Spectral	.448 ± .000	.152 ± .000	.135 ± .000
	$Leiden_S$	.379 ± .043	.132 ± .016	.115 ± .017
	Node2Vec	.195 ± .001	.023 ± .001	.014 ± .000
	Attri2Vec	.199 ± .002	.026 ± .001	.017 ± .000
	DynNode2Vec	.177 ± .002	.012 ± .002	.006 ± .001
	tNodeEmbed	.199 ± .002	.026 ± .001	.017 ± .000
	DAEGC	.628 ± .050	.356 ± .040	.337 ± .055
	DMoN	.221 ± .014	.039 ± .009	.029 ± .006
	TGC	**.681 ± .005**	**.434 ± .006**	**.415 ± .007**
$SBM_{\eta=.5}$	k-means	.648 ± .016	.400 ± .015	.375 ± .018
	Spectral	.210 ± .000	.025 ± .000	.018 ± .000
	$Leiden_S$	.204 ± .017	.019 ± .008	.012 ± .005
	Node2Vec	.174 ± .006	.007 ± .004	.004 ± .002
	Attri2Vec	.175 ± .006	.005 ± .004	.003 ± .002
	DynNode2Vec	.176 ± .003	.005 ± .001	.003 ± .000
	tNodeEmbed	.175 ± .006	.005 ± .004	.003 ± .002
	DAEGC	.466 ± .088	.218 ± .050	.180 ± .058
	DMoN	.170 ± .005	.007 ± .004	.004 ± .001
	TGC	**.681 ± .003**	**.432 ± .005**	**.415 ± .005**
$SBM_{\eta=.25}$	k-means	.648 ± .016	.400 ± .015	.375 ± .018
	Spectral	.181 ± .000	.006 ± .000	.004 ± .000
	$Leiden_S$	.180 ± .006	.010 ± .002	.006 ± .002
	Node2Vec	.170 ± .004	.002 ± .001	.001 ± .001
	Attri2Vec	.168 ± .005	.002 ± .002	.001 ± .001
	DynNode2Vec	.167 ± .001	.006 ± .001	.003 ± .000
	tNodeEmbed	.168 ± .005	.002 ± .002	.001 ± .001
	DAEGC	.375 ± .054	.151 ± .017	.110 ± .025
	DMoN	.165 ± .006	.002 ± .003	.001 ± .001
	TGC	**.680 ± .003**	**.431 ± .004**	**.414 ± .004**
$SBM_{\eta=0}$	k-means	.648 ± .016	.400 ± .015	.375 ± .018
	Spectral	.183 ± .000	.017 ± .000	.010 ± .000
	$Leiden_S$	.181 ± .006	.008 ± .004	.005 ± .002
	Node2Vec	.173 ± .003	.004 ± .001	.002 ± .001
	Attri2Vec	.171 ± .006	.004 ± .004	.002 ± .002
	DynNode2Vec	.165 ± .001	.005 ± .000	.002 ± .000
	tNodeEmbed	.171 ± .006	.004 ± .004	.002 ± .002
	DAEGC	.390 ± .086	.173 ± .054	.128 ± .060
	DMoN	.165 ± .009	.002 ± .004	.002 ± .003
	TGC	**.682 ± .006**	**.433 ± .009**	**.416 ± .009**

Table 6.2: Synthetic TADC-SBM [129] graph baselines. Values are mean ± standard deviation over 5 runs. Bold and underline mark the highest and second-highest means per dataset and metric. Fraction of nodes changing communities in each snapshot approximately given by $1 - \eta$.

Dataset	Variant	Accuracy	AMI	ARI
$SBM_{\eta=1}$	$Leiden_{MS}$	**1.000 ± .000**	**1.000 ± .000**	**1.000 ± .000**
	$Leiden_{S}$	.849 ± .055	.945 ± .022	.848 ± .048
$SBM_{\eta=.75}$	$Leiden_{MS}$	**.432 ± .010**	**.149 ± .005**	**.131 ± .004**
	$Leiden_{S}$	.379 ± .043	.132 ± .016	.115 ± .017
$SBM_{\eta=.5}$	$Leiden_{MS}$	.186 ± .010	.019 ± .006	.011 ± .003
	$Leiden_{S}$	**.204 ± .017**	.019 ± .008	**.012 ± .005**
$SBM_{\eta=.25}$	$Leiden_{MS}$	**.181 ± .011**	**.013 ± .004**	**.007 ± .002**
	$Leiden_{S}$	.180 ± .006	.010 ± .002	.006 ± .002
$SBM_{\eta=0}$	$Leiden_{MS}$	**.181 ± .008**	**.014 ± .005**	**.008 ± .003**
	$Leiden_{S}$	.181 ± .006	.008 ± .004	.005 ± .002

Table 6.3: Leiden baselines on SBM graphs. Values are mean ± standard deviation over 5 runs; bold marks the highest means. $Leiden_{S}$ and $Leiden_{MS}$ denote static and multislice modularity optimization on the aggregated and supra-graph constructions, respectively (Eq. 4.5 and 4.9).

Meanwhile, modularity optimization with the default Leiden [176] algorithm implementation [177], which yields a consistent [205] clustering objective for $\eta = 1$, fails to recover the planted partition for $\eta < 1$ (Table 6.3), even when using the multislice construction (Eq. 4.9) that explicitly encodes temporal dependencies, as shown in Table 6.3. In the SBM graphs generated with $\eta < 1$, however, it yields better results than DMoN, which optimizes a similar objective by gradient descent (Equation 6.16), suggesting that the latter's optimization is more sensitive to the temporal mismatch between historical and final partitions, which the feature signal cannot fully compensate for in the evaluated regimes. This observation motivates work on neural encoders that explicitly model temporal dependencies in the clustering objective, as explored in Chapter 7 of this thesis — although the results here shown also suggest that the benefits of such an approach may be limited to regimes where (i) the planted communities are sufficiently stable when the graph is aggregated, i.e., their (spatial) signal remains informative, which depends on the resulting within/between edge ratio and average degree of the communities in the sparse SBM regime (Equation 4.2); (ii) the spatio-temporal signal in itself is sufficiently informative, which depends on the degree of temporal correlation between snapshots and the strength of the coupling parameter (Equation 4.3); or (iii) the feature-based signal is able to compensate for the corruption of the previous ones, which varies based on the degree of alignment between the feature and community signals (Figure 5.2).

The remaining observations clarify the limits of the experiment. First, spectral clustering on the aggregated graph performs well only when the planted communities are stable, as expected. Second, although TGC attained the best results in the domains where $\eta < 1$, its performance on $SBM_{\eta=1}$ was worse than methods relying on graph topology alone (Spectral, Leiden), optimizing structure and features (DMoN), or using a reconstruction-based objective (DAEGC), as the model does not yield a consistent graph clustering objective [205], as discussed in Section 4.2.1. Third, Attri2Vec [201] does not improve over Node2Vec, despite incorporating node features, suggesting that a random-

walk objective jointly exploiting it is insufficient to retrieve community structure when the sampled neighborhoods did not enhance the spatio-temporal signal. And last, the experiment perturbed the structure-based signal, but kept features static and ground truths on the last snapshot, thereby favoring a particular modeling approach; future comprehensive benchmarks may seek to introduce dynamic node- and edge-level features as well, to assess the robustness of neural methods to different types of signal strains.

This interpretation is supported by the empirical structure of the generated graphs, which contain $|\mathcal{V}| = 1024$ nodes over 8 snapshots and have similar average degree (≈ 9.8) in all graphs (Table 5.2). The main variation is therefore not graph density, but the alignment between historical topology and the target partition: for $\eta = 1$, all snapshots remain assortative with respect to the planted labels, with an average within/between edge ratio of 1.26, whereas, as $\eta \to 0$, snapshots become weakly aligned or even disassortative with respect to the final partition, with average ratios of $.30$ for $\eta = .5$ and $.28$ for $\eta = 0$. The final snapshot itself remains as strongly assortative as the final snapshot of $\text{SBM}_{\eta=1}$ in all simulations; the benchmarks thus create a mismatch between *historical* topology (from *evolutionary* dynamics) and the community assignment being evaluated — connecting it to the detectability discussion (Section 2.2.2): recovery depends not only on the instantaneous assortative signal, but also on the temporal stability of the labels. In the dynamic case, decreasing η weakens the correlation between earlier labels and the final partition, so that earlier snapshots dilute the effective signal of objectives that aggregate, smooth, or optimize independently across time. The poor performance of Spectral, Leiden, and DMoN for $\eta < 1$ should therefore not be read as a modeling failure a priori, but as a consequence of optimizing objectives whose target are naive to the temporality of community structure. This also explains why random-walk proximity fails in the same regimes, regimes (Node2Vec, Attri2Vec, DynNode2Vec, tNodeEmbed); whereas other time-aware methods may benefit from, e.g., modeling historical interactions to emphasize more recent signals, thus yielding distinct modeling assumptions.

This section's results therefore do not serve as a general ranking of the evaluated baseline methods; rather, they show that different models implicitly target distinct temporal objects: e.g., a final-snapshot partition, a persistent partition over the whole graph, or a temporally smoothed community trajectory. Future synthetic evaluations should ideally seek to vary transition probabilities, feature alignment (Figure 5.2) and dynamics, number of snapshots, and other properties systematically; the TADC-SBM generator also admits natural extensions to directed, weighted, multigraph, dynamic-feature, and mixed-membership settings. The next section discusses the broader implications of these results for neural community detection and the challenges of defining ground truths.

6.2.3 Ground truths on neural community detection[10]

Benchmark evaluations of (neural and algorithmic) community detection methods often rely on observed node metadata as a proxy for ground truth [131]. This assumption is not generally justified: metadata may encode meaningful node properties, but it need not coincide with the mesoscale organization induced by topology, attributes, and their

[10]Part of this section was partly presented in: *Efficient 'Neural' Community Detection on Temporal Graphs: Challenges, Advancements and Opportunities.* Passos, N.A.R.A. (2025). Poster presentation at the Future Artificial Intelligence Research General Conference (FAIR-GC '25), Dec. 10–12, 2025, Rome, Italy.

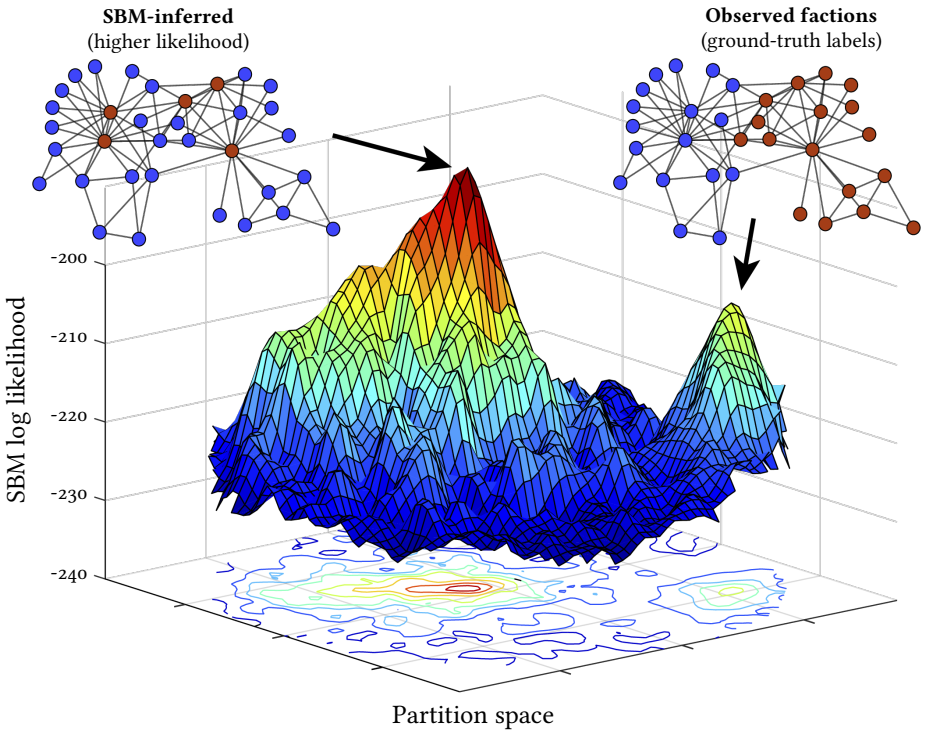


Figure 6.5: Zachary's Karate Club network [199]. The log-likelihood surface for SBM bipartitions shows two peaks: the lower peak aligns with ground-truth factions, while the higher peak represents a leader-follower partition, illustrating how different community definitions may be favored by different criteria, even in simple, unattributed networks. Adapted from [131].

temporal co-evolution. The issue is especially important for neural models, whose expressive encoders can fit strong attribute regularities, but not in a *principled* manner — including cases where those attributes are only weakly coupled to the structural community signal. In such settings, high agreement with metadata may indicate attribute prediction rather than community recovery, while low agreement may reflect a mismatch between metadata and the graph-generating structure rather than model failure[11].

This distinction is visible even in simple networks. In Zachary's Karate Club [199], the observed factions are often used as ground-truth labels, although they do not necessarily coincide with the partition preferred by an explicit generative model [131]. Figure 6.5 illustrates this mismatch: the metadata-based split corresponds to one local optimum of the SBM likelihood surface, whereas a different leader-follower partition attains a higher likelihood under the fitted model. The point is not that the likelihood-maximizing partition is universally correct, but that metadata, topology, and model likelihood can define different valid community descriptions of the same graph, as Figure 6.5 illustrates.

This ambiguity is a characteristic of the non-uniqueness of community structure. For neural models, this is compounded by optimization and representation learning: the loss landscape is usually highly non-convex, and the learned embeddings may encode multiple sources of signal, including topology, attributes, and temporal dependencies, in a way that it becomes unclear which is driving the resulting partition. When evaluation is reduced to metadata recovery, it becomes difficult to distinguish community detection

[11] See Section 4.2.3 for a discussion of community detectability thresholds in temporal SBM graphs.

from attribute prediction or node classification: a model that accurately predicts metadata may be exploiting patterns other than in the community structure, whereas a model that disagrees with metadata may still recover coherent structural partitions.

Metadata and community structure should therefore be treated as related, but distinct properties of a system. The former describes observed attributes or annotations, such as demographic variables, institutional affiliations, interaction labels, or roles; while the latter refers to regularities in how nodes interact, connect, or co-evolve within the same network. Signals may indeed align, as shown in Figure 5.2 — and when they do, their joint consideration allows more expressive models to identify more informative patterns, which is where GNNs' strength lies — but there is no general reason to assume that the most informative structural partition is also the one that best predicts observed metadata.

Real-world metadata also reflects measurement choices. Annotations may be noisy, incomplete, delayed, temporally mismatched, or defined for purposes unrelated to the network process being studied. In the American college football network [53], for example, conference labels are often used as metadata, although labels and links may refer to different seasons or institutional arrangements. Such cases do not make metadata useless; rather, they make it insufficient as a definitive benchmark for community detection.

For this reason, the evaluation of neural community detection benefits from controlled benchmarks in which topology, attributes, and temporal dynamics are generated jointly with known underlying communities. Models like TADC-SBM [129] provide such controlled environments, where it is possible to separate sources of signal and test whether a model recovers communities from edges, attributes, temporal persistence, and different combinations thereof — as shown in Figure 5.5 to evaluate the effect of doubling the number of nodes, edges, and snapshots on model performance. The next chapter presents a neural architecture that extends modularity optimization to continuous-time graphs, then evaluated on both synthetic and real-world benchmarks, including TADC-SBM and temporal networks with additional node-level metadata.

Chapter 7

Neural Community Detection

Established descriptive methods for community detection [135], such as spectral clustering and modularity optimization, uncover mesoscale structures by analyzing the graph's topology. Node and edge metadata generally enters these methods through additional assumptions and bolt-on extensions, on top of their objective functions — motivating the exploration of expressive models that can *learn* from these sources simultaneously.

Neural community detection methods differ not only in architecture, but also in the object they optimize: some reconstruct graphs, some learn embeddings for post hoc clustering, and some directly optimize differentiable relaxations of graph-theoretic community objectives. The empirical results shown in the previous chapter further demonstrate that neural and temporal mechanisms are useful only when their inductive biases align with the structural, attribute-based, and temporal signals present in the data.

This chapter builds on that observation by introducing the Longitudinal Modularity Network (LMoN), a neural model that extends differentiable modularity optimization to continuous-time attributed graphs. LMoN extends the DMoN framework [179] by replacing the static modularity objective with a temporally reweighted longitudinal modularity [10] operator. Unlike the models reviewed in Chapter 6, LMoN does not specifically introduce temporal dependence by the means of an encoder: it modifies the graph-theoretic operator entering the community objective, so that recency-weighted interaction history affects both message passing and modularity optimization.

In the strict conceptualization of neural community detection adopted in this thesis, LMoN optimizes a community objective directly rather than clustering embeddings produced by an auxiliary representation-learning loss. The model maps a time-varying attributed graph to (soft) community assignments by combining a temporal graph operator, a graph neural encoder, and a differentiable longitudinal modularity objective. When temporal variation is removed, the operator reduces to the static case, and LMoN recovers the DMoN setting; improvements over the static baseline can therefore be attributed to the added temporal mechanism rather than to a change in encoder class.

The purpose of this chapter is to evaluate whether such a temporal graph-theoretic objective improves neural community detection on graphs with mixed topology, attributes, and temporal dynamics. The architecture and objective are first introduced, including the temporal operator and implementation details. The model is then evaluated on real-world and synthetic graphs against algorithmic and neural baselines. The chapter concludes by discussing when LMoN improves over static neural modularity, when it fails, and what these outcomes imply for temporal graph learning tasks.

7.1 The LMoN graph neural network[1]

The Longitudinal Modularity Network (LMoN) is a GNN that optimizes a differentiable function mapping a time-varying attributed graph $\mathcal{G} := (\mathcal{V}, \mathcal{E}, \mathbf{X})$ to a (soft) community assignment matrix $\mathbf{C} \in [0,1]^{n \times k}$, where $n = |\mathcal{V}|$ is the number of nodes, k is the number of communities, and $\mathbf{X} \in \mathbb{R}^{n \times d}$ is a d-dimensional node feature matrix with metadata.

The model comprises three components, illustrated in Figure 7.1: a temporal operator that reweights the graph by interaction recency, producing a (learnable) time-decayed adjacency matrix $\hat{\mathbf{A}}$; an optionally multilayer graph encoder (e.g., a GCN [84]) that maps $\hat{\mathbf{A}}$ and $\mathbf{X}$ to low-dimensional node embeddings; and a clustering layer that produces soft assignments by a $\mathrm{softmax}$, optimized on the temporal modularity objective.

Soft memberships relax the discrete modularity-maximization problem into a continuous, real-valued space suitable for gradient-based optimization [82], and naturally accommodate overlapping (mixed-membership) communities as a convex combination of cluster assignments. Hard assignments are recovered by an $\arg\max$ over each node's membership vector, which is used for evaluation against ground-truth partitions.

Note that the clustering layer is agnostic to the specific choice of an encoder: although the instantiation here evaluated uses a graph convolutional network, chosen for its scalability and direct comparability over the static baseline (DMoN), the temporal operator and objective are both compatible with other architectures, including attention-based, inductive encoders, and domain-informed priors such as for molecular dynamics [105].

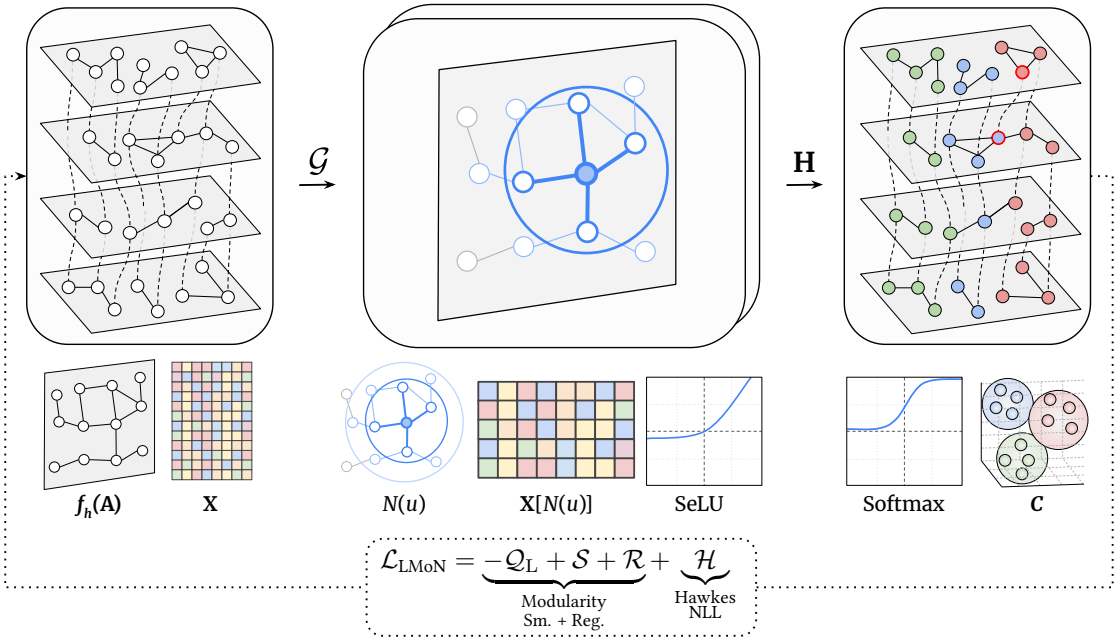


Figure 7.1: Illustration of the LMoN architecture. A temporal graph $\mathcal{G}$ is reweighted into a time-decayed matrix $\hat{\mathbf{A}}$ and encoded with node attributes $\mathbf{X}$ to produce soft assignments $\mathbf{C}$. Training minimizes $\mathcal{L}_{\text{LMoN}}$, combining modularity, smoothness, regularization, and a (decoupled) Hawkes negative log-likelihood of the event sequences through a distinct optimizer. Revised from [127].

[1] Initially presented in *Dynamic Graph Clustering with Graph Neural Networks.* Passos, N.A.R.A. (2026). Poster presentation at the Learning on Graphs (LoG) Italian Meetup, 9–11 June, 2026, Pisa, Italy.

7.1.1 Optimization objective

LMoN is trained by minimizing an unsupervised, multi-component loss that performs community detection directly through a differentiable modularity objective.

The loss function is composed of two main terms: a community detection objective $\mathcal{L}_{\text{mod}}$ and an optional temporal likelihood $\mathcal{L}_{\text{time}}$, which supervises the temporal operator through a Hawkes-based negative log-likelihood of events. It may be expressed as

$$\mathcal{L}_{\text{LMoN}} = \mathcal{L}_{\text{mod}} + \mathcal{L}_{\text{time}}, \tag{7.1a}$$

$$\mathcal{L}_{\text{mod}} = -\mathcal{Q}_{\text{L}} + \mathcal{S} + \mathcal{R}, \tag{7.1b}$$

$$\mathcal{L}_{\text{time}} = \mathcal{H}, \tag{7.1c}$$

where $\mathcal{Q}_{\text{L}}$ is a spectral longitudinal modularity objective; $\mathcal{R}$ is a collapse regularization term; $\mathcal{S}$ is a temporal smoothness term [10]; and $\mathcal{H}$ is a Hawkes-process negative log-likelihood that supervises the temporal operator — its parameters are updated separately from the community objective, and it does not define the modularity criterion itself.

Community detection objective

The modularity term applies the spectral relaxation underlying DMoN to the longitudinal modularity matrix, defined by Equation 4.13. It may be expressed as

$$\mathcal{Q}_{\text{L}} = \frac{1}{2m} \operatorname{Tr}\left(\mathbf{C}^{\top} \mathbf{M}_{\text{L}}\, \mathbf{C}\right), \qquad \mathbf{M}_{\text{L}} = \hat{\mathbf{A}} - \mathbf{P}, \tag{7.2}$$

where $2m = \sum_{ij} \hat{A}_{ij}$ is the total weighted degree, $\hat{\mathbf{A}}$ is the temporally decayed adjacency matrix (Section 7.1.2), and $\mathbf{M}_{\text{L}}$ is the longitudinal modularity matrix. The null model $\mathbf{P}$ is defined by the mean-membership longitudinal link expectation (Equation 4.12) [10],

$$\mathbf{P} = \frac{1}{2m} \left(\mathbf{d}_{\text{out}} \odot \mathbf{p}\right) \left(\mathbf{d}_{\text{in}} \odot \mathbf{p}\right)^{\top}, \tag{7.3}$$

where $\mathbf{d}_{\text{out}}, \mathbf{d}_{\text{in}}$ are the weighted degree vectors (equal for undirected graphs, $\mathbf{d}_{\text{out}} = \mathbf{d}_{\text{in}} = \mathbf{d}$), $\mathbf{p}$ is a node-presence vector encoding normalized lifetimes, and $\odot$ denotes elementwise multiplication. This extends the classical modularity null model to account for temporal node activity, without requiring the objective to be formulated as a sequence of independent snapshot modularities, and recovers the standard configuration model when presence is uniform, such as in static graphs or when all nodes remain present.

The collapse regularizer $\mathcal{R}$, inherited from DMoN [179], discourages the trivial solution in which all nodes share a single community, and may likewise be expressed as

$$\mathcal{R} = \vartheta \left(\frac{\sqrt{k}}{n} \left\| \sum_{i} \mathbf{C}_{i\cdot} \right\|_2 - 1 \right), \tag{7.4}$$

where ϑ controls its strength and $\|\cdot\|_2$ is the Euclidean norm of the vector of per-cluster membership masses; the term is minimized (zero) when community sizes are balanced. Unlike harder orthogonality constraints, this relaxed penalty discourages partition collapse without forcing assignments to be exactly balanced or dominating the primary

objective — therefore plays a role analogous to the size-normalization in ratio cut minimization (Equation 2.7), but in a differentiable soft-assignment form introduced in DMoN as a less restrictive alternative to orthogonality-based pooling penalties [179].

The temporal smoothness regularizer $\mathcal{S}$ encourages assignments that are stable over time, penalizing nodes that switch communities between consecutive interaction times,

$$\mathcal{S} = \frac{\omega}{2m} \sum_{u \in \mathcal{V}} \sum_{t} \left\| \mathbf{c}_u^{(t+1)} - \mathbf{c}_u^{(t)} \right\|_2^2, \tag{7.5}$$

the differentiable relaxation of the longitudinal smoothness term of Equation 4.15, where ω controls its strength and $\mathbf{c}_u^{(t)}$ is the soft assignment of node u at time t. Note that, in several real-world datasets, community assignments are not expected to change over time (*static* community detection in *dynamic* graphs); in such cases, the model predicts a single assignment matrix for the temporally reweighted graph, and so the smoothness term may be omitted ($\omega = 0$) as it does not contribute to the objective. The term is retained in the formulation for settings in which the model produces time-indexed assignments $\mathbf{C}^{(t)}$ and temporal community trajectories must be explicitly estimated [105].

Combining the community terms in the loss yields an explicit objective, defined as

$$\mathcal{L}_{\text{mod}} = -\frac{1}{2m} \operatorname{Tr}\left(\mathbf{C}^\top \mathbf{M}_{\text{L}} \mathbf{C}\right) + \frac{\omega}{2m} \sum_{u \in \mathcal{V}} \sum_{t} \left\| \mathbf{c}_u^{(t+1)} - \mathbf{c}_u^{(t)} \right\|_2^2 + \vartheta \left(\frac{\sqrt{k}}{n} \left\| \sum_i \mathbf{C}_{i\cdot} \right\|_2 - 1 \right), \tag{7.6}$$

which balances longitudinal modularity, temporal coherence, and community sizes.

Under static conditions, the objective reduces to the DMoN spectral modularity relaxation; therefore, the usual modularity-based consistency proof [179, 205] applies. In temporal cases, the same proof applies insofar as the graph is assortative and the spatio-temporal signal is aligned with the target partition, such as in the sparse SBM regime presented in Section 5.1, for which detectability thresholds are well-studied (Section 5.1.2).

The model optionally augments this objective with a Hawkes negative log-likelihood term for the temporal operator (Figure 7.1), which is optimized with a separate optimizer and small weight to avoid dominating the primary objective and allow for time-naive encoders like GCNs to exploit temporal dynamics from the reweighted graph in a scalable manner. Note that, although the Hawkes likelihood is not strictly a community objective, it is compatible with the encoder-agnostic design of LMoN and may be used to supervise the temporal operator when the interaction sequences are known, without requiring additional supervision on the community assignments themselves, as seen next.

Temporal supervision objective

For an edge (i, j) observed over a time window $[0, T]$, let $T_{ij} = \{t_1, \ldots, t_{N_{ij}}\}$ denote its interaction times. LMoN models these events with a conditional intensity defined as

$$\lambda_{ij}(t) = \mu_{ij} + \alpha \sum_{t_k \in T_{ij} : t_k < t} \kappa(t - t_k), \tag{7.7}$$

where μ_{ij} is a baseline interaction rate, α controls self-excitation, and $\kappa(\cdot)$ is a non-negative decay kernel. Employing, for instance, an exponential kernel $\kappa(t) = \exp(-\beta t)$, the intensity is a Hawkes process [72] that models the likelihood of future interactions

based on past events, where the decay parameter β controls how quickly the influence of past events diminishes over time. Other choices of kernel are possible, such as power-law or Rayleigh, and the model supports trainable α, β and μ parameters, as seen next.

The temporal supervision term $\mathcal{L}_{\text{time}} = \mathcal{H}$ (Equation 7.1) is an auxiliary likelihood used to fit the parameters of the temporal operator that produces $\hat{\mathbf{A}}$. It encourages the decay applied to each interaction to be consistent with the observed event sequence, while leaving the community criterion defined by the other terms unaffected. Under the standard point-process likelihood, the negative log-likelihood over observed events is

$$\mathcal{L}_{\text{time}} = \mathcal{H} = -\sum_{(i,j)\in\mathcal{E}} \sum_{t_k \in T_{ij}} \log \lambda_{ij}(t_k) + \sum_{(i,j)\in\mathcal{E}} \int_0^T \lambda_{ij}(t)\, dt. \tag{7.8}$$

As a modeling approach, this is useful when event timestamps contain information about the persistence or recency of interactions. In this setting, $\mathcal{H}$ provides a learnable decay signal: the event term rewards high intensity at observed interaction times, while the compensator penalizes intensity assigned to intervals in which no interaction occurs. The resulting operator emphasizes edges whose recent histories are consistent with persistent interaction, rather than treating all aggregated edges as equally informative.

Maximizing event likelihood is nevertheless not equivalent to recovering communities. An event sequence may be temporally predictable without being modular, and a modular partition may remain stable even when the point-process likelihood is weak, for example when interactions are sparse, bursty, or weakly self-exciting [18]. Therefore, $\mathcal{H}$ is used only as a supervisory signal for the temporal operator: optimized separately from $\mathcal{L}_{\text{mod}}$ and assigned a small contribution to the total loss, the temporal likelihood informs the reweighted graph without being treated as evidence of community quality.

7.1.2 Encoding strategy

The objective above is agnostic to the encoder architecture. This section describes the instantiation used in the experiments: a Hawkes-Laplacian temporal operator followed by a two-layer graph convolutional encoder. The encoder itself is deliberately simple, since graph convolution has already been introduced in Chapter 6 and provides direct comparability with the static DMoN baseline. The temporal contribution of LMoN is instead located in the operator passed to the encoder and to the modularity objective.

Hawkes-Laplacian operator

The Hawkes-Laplacian operator reweights the adjacency matrix and applies a symmetric normalization, following Equations 7.7 and 2.5. Current implementation multiplies the time-decayed aggregated edge weight by A_{ij}, so pairwise interactions that have a higher baseline weight are emphasized to support weighted graphs (binary otherwise). For an observed edge among nodes (i, j) at time t, its time-decayed weight is given by

$$\hat{A}_{ij}(t) = A_{ij}(t) \left(\mu_{ij} + \alpha \sum_{t_k \in T_{ij}:\, t_k < t} w_k\, \kappa(t - t_k) \right), \tag{7.9}$$

where A_{ij} is the edge weight, T_{ij} is the set of past interaction times for the edge, w_k is the weight of the k-th interaction, and remaining parameters follow previous definitions.

Following a similar parameterization to [100], the baseline intensity may be set to

$$\mu_{ij} = \sqrt{\mu_i \mu_j}, \qquad \mu_i = \mathrm{softplus}(\mathbf{x}_i^\top \mathbf{w}_\mu + b) + \epsilon, \tag{7.10}$$

where $\mathbf{x}_i$ is the feature vector of node i, $\mathbf{w}_\mu$ is a learnable weight vector, and b is a learnable scalar bias (shared across all nodes). The $\mathrm{softplus}$ operator ensures non-negativity, while a small constant $\epsilon \ll 1$ keeps $\mu_i > 0$, avoiding unstable gradients when $\mu_{ij} \to 0$.

Message passing is then performed on the normalized reweighted adjacencies $\tilde{\mathbf{A}}$, i.e.,

$$\tilde{\mathbf{A}} = \hat{\mathbf{D}}^{-1/2} \hat{\mathbf{A}} \hat{\mathbf{D}}^{-1/2}, \qquad \hat{\mathbf{D}}_{ii} = \sum_j \hat{A}_{ij}, \tag{7.11}$$

and subsequently used for message passing, so temporal information enters the model *before* the encoder. Interactions are reweighted by their accumulated pairwise intensity, so the resulting adjacency is used to propagate information through the graph with the age-based discounting introduced next (Eq. 7.13). The same $\hat{\mathbf{A}}$ also enters the longitudinal modularity term through $\mathbf{M}_{\mathrm{L}} = \hat{\mathbf{A}} - \mathbf{P}$ (Eq. 7.2), so the Hawkes-Laplacian operator affects both representation learning and (temporal) community objective optimization.

Kernel functions

The temporal kernel $\kappa(\Delta)$ determines how the influence of an interaction changes with its age $\Delta = t - t_k$. The experiments in this chapter employ the exponential kernel

$$\kappa_{\exp}(\Delta) = \exp(-\beta \Delta), \qquad \beta = -\log \rho, \tag{7.12}$$

where β is the decay rate and ρ is the corresponding discrete-time retention factor. This parameterization allows unconstrained optimization of $\rho \in (0, 1)$ through a sigmoid transform while ensuring non-negative decay. It also helps with interpretation: $\rho \approx 1$ retains long interaction histories, whereas $\rho \approx 0$ rapidly suppresses past interactions.

Two alternative kernel functions are also implemented: power-law kernels, which decay more slowly than an exponential kernel and may be appropriate when old interactions retain influence over long horizons; and Rayleigh kernels, which model influence as initially increasing before decaying, making it suitable for domains where activity peaks after a delay ($\Delta = 1/\sqrt{\beta}$) rather than immediately after an event, such as engagement cascades in online social networks. These alternatives are best viewed as domain-informed extensions of the temporal operator, which do not coincide with the sparse SBM regime established in Section 6.2.2. Moreover, they also encode different assumptions about interaction dynamics and do not admit the same efficient exponential recursion in the current implementation; in special, the exponential event-time readout is refined into the node-anchored recency operator of Equation 7.13, detailed next.

Node-anchored recency kernel

To allow the model to distinguish stale from recent interactions, the operator (Eq. 7.9) is modified to consider the interaction time specific to each pair of nodes. Pairwise intensity is thus read out at an *anchor* time τ_{ij}, discounting each event by its age $\tau_{ij} - t_k$;

stale interactions are therefore downweighted, so that the adjacency reflects recency relative to the prediction horizon. This results in a single aggregated adjacency weight,

$$\tilde{A}_{ij} = A_{ij}\Big(\mu_{ij} + \alpha \sum_{t_k \in T_{ij}} w_k \, \kappa(\tau_{ij} - t_k)\Big), \qquad \tau_{ij} = \tfrac{1}{2}\big(t_i^{\text{last}} + t_j^{\text{last}}\big), \tag{7.13}$$

where the sum now runs over all past interactions of the pair and $t_i^{\text{last}} = \max\{t : (i, \cdot, t) \in \mathcal{E}\}$ is node i's last activity. Anchoring to the mean of the two endpoints' last-activity times preserves a lone tie between mutually dormant nodes: when a pair's only interaction is a single shared event, τ_{ij} coincides with it, giving age 0 and full weight, whereas a stale edge incident to a still-active node is downweighted. Equation 7.13 thus breaks the per-pair shift invariance of Eq. 7.9, and under full presence ($t_i^{\text{last}} = T \; \forall i$) it reduces to a global horizon anchor $\tau_{ij} = T$, so the node- and horizon-anchored operators coincide whenever all nodes remain active. The normalized operator $\hat{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}$ then feeds both message passing and the modularity objective for community assignments.

7.1.3 Implementation details

LMoN learns a differentiable map $f : \mathcal{G} \mapsto \mathbf{C} \in [0, 1]^{n \times k}$ on a temporal attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where $\mathbf{X} \in \mathbb{R}^{n \times d}$ is the node-attribute matrix and $\mathbf{C}$ is a row-stochastic soft assignment matrix. The implementation relies on the TensorFlow framework with adaptive moment estimation [82] for stochastic gradient descent, with (optional) early stopping enabling fully unsupervised training and model selection — a common scenario for the task when no ground truths are available [179]. The next sections detail its computational complexity, hyperparameterization, and relation to its baseline, DMoN.

Computational complexity

Optimizing a relaxed form of modularity through gradient descent yields a highly non-convex loss landscape, which calls for careful management of model complexity. The main computational issue is that the modularity matrix is dense if formed explicitly, which the model avoids materializing by employing a sum of sparse-matrix multiplications and rank-one degree normalization, bringing its time complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(d_v^2 n)$ per edge, where n are nodes and d_v is their feature dimensionality [179].

The Hawkes-Laplacian operator first constructs a (sparse) temporally reweighted adjacency $\hat{\mathbf{A}}$ from observed event histories (Equation 7.9), which the graph convolutional encoder then receives in its normalized form (Eq. 7.11). The modularity objective (Eq. 7.2) instead employs the unnormalized longitudinal matrix (Eq. 4.13), therefore expressed by

$$\mathbf{M}_{\text{L}} = \hat{\mathbf{A}} - \gamma \frac{(\mathbf{d}_{\text{out}} \odot \mathbf{p})(\mathbf{d}_{\text{in}} \odot \mathbf{p})^\top}{2m}, \tag{7.14}$$

meaning that temporal activity modifies the rank-one null model (Eq. 4.7) through presence-weighted degrees, while the observed term remains the reweighted adjacency.

The modularity trace (Equation 7.2) is evaluated using the decomposition

$$\operatorname{Tr}\big(\mathbf{C}^\top \mathbf{M}_{\text{L}} \mathbf{C}\big) = \operatorname{Tr}\Big(\mathbf{C}^\top \hat{\mathbf{A}} \mathbf{C} - \mathbf{C}^\top (\mathbf{d}_{\text{out}} \odot \mathbf{p})^\top \, (\mathbf{d}_{\text{in}} \odot \mathbf{p}) \, \mathbf{C}\Big), \tag{7.15}$$

where $\mathbf{d}_{\text{in}} = \mathbf{d}_{\text{out}}$ for undirected graphs. The first term is computed by sparse multiplication over the nonzero entries of $\hat{\mathbf{A}}$, and the second is a rank-one degree-presence correction. The dense $n \times n$ modularity matrix is therefore never constructed; together with a GCN encoder, the per-epoch cost grows linearly with the number of edges, $\mathcal{O}(m\, d_v^2\, n)$.

The regularization term $\mathcal{R}$ is inexpensive and computed in $\mathcal{O}(n)$ [179], while the smoothness term $\mathcal{S}$ is computed in $\mathcal{O}(2|\mathcal{E}| - n)$, as each interaction contributes with two possible node reassignments to a distinct community — excluding initial assignments, which do not constitute a transition and are therefore not penalized (Equation 4.15) [10].

For the exponential kernel used in the experiments, event histories are grouped by edge and sorted by time. If the event stream is unordered, this preprocessing costs $\mathcal{O}(|\mathcal{E}| \log |\mathcal{E}|)$ over $|\mathcal{E}|$ temporal events; after sorting, the causal Hawkes excitation and compensator are evaluated in a single pass, with cost $\mathcal{O}(|\mathcal{E}|)$. Alternative kernels such as power-law and Rayleigh do not use this exponential recursion in the current implementation and are therefore not the basis for the complexity reported here. Meanwhile, the node-anchored readout (Eq. 7.13) requires one additional pass to obtain each node's last-activity time t_i^{last} and a per-pair aggregation at the anchor over the (observed) edges, both linear in the number of events, preserving the overall complexity of the operator.

The overall result is a scalable model that is suitable for large-scale sparse temporal graphs, a regime which many real-world networks are expected to inhabit. Note that the static embeddings outputted by the encoder (GCN) are optimized for cases where the community structure is expected to be static, like in the real-world datasets described in Table 5.5 and evaluated next. For the synthetic graphs instead, the last-snapshot ground truths were considered, so to allow fair comparison with the static baseline (DMoN) and match the encoder's output. In such cases, the smoothness term may therefore be omitted from the computational cost ($\omega = 0$) as it does not contribute to the objective.

Hyperparameterization

The Hawkes decay parameters are updated separately from the community objective, with gradient clipping and regularizers for numerical stability. In the absence of ground truths, the number of communities k, learning rate, dropout, and regularization strengths ω, φ (Equations 7.4 and 7.5) may be selected by grid search, with training halted by early stopping using an unsupervised criterion — e.g., the modularity score or structural metrics, such as the global ratio of edges within communities (i.e., coverage[2]), or normalized by number of classes (i.e., conductance; Eq. 4.1). Note that k might be set to a maximum number of communities larger than that of the dataset — through the use of $\mathrm{softmax}$, the model is not constrained to mapping nodes to all k communities [179].

If baseline intensities are learned (Equation 7.10), the weight vector $\mathbf{w}_\mu$ is initialized on a normal[3] distribution to preserve activation variance during the forward pass and gradient variance during the backward pass [55], and the bias term is initialized as $b = \mathrm{softplus}^{-1}(\mu_0)$, with $\mu_0 = 1$ (default), so $\mu_i \approx \mu_0$ on average for all nodes at start. Lastly, as in DMoN, the clustering layer's weight matrix is initialized to be orthogonal,

[2] If unnormalized by community size, $\mathrm{coverage}(G) = 1 - \mathrm{conductance}(G)$, i.e., its complement; coverage is biased toward fewer, larger clusters, while conductance may favor many, smaller clusters.

[3] Since μ_i enters the log-likelihood both directly and via the jointly learned α, β, an arbitrarily miscaled baseline, e.g. from zero-initializing b or α, could dominate the initial loss and slow convergence.

$\mathbf{W}^\top\mathbf{W} = \mathbf{I}$, preserving activation norms [156], while the bias term is initialized to zero as there is no target value to encode at start, i.e., no prior preference for any community.

Recovering DMoN as a special case

Note that DMoN may be seen as a special case of LMoN when temporal information is discarded and the graph is treated as static, so that $\hat{\mathbf{A}}$ reduces to $\mathbf{A}$ up to normalization.

The presence-weighted null model $\mathbf{P}$ of longitudinal modularity (Equation 7.3) reduces to the standard configuration model (Eq. 4.5) [23] if and only if node activity is uniform ($p_i = 1$ for all nodes i, yielding an all-ones vector $\mathbf{p}$); in such case, $\mathcal{L}_{\text{DMoN}} = \mathcal{L}^{\text{mod}}_{\text{LMoN}}$.

For the full loss, uniform presence makes every node anchor coincide with the horizon, $\tau_{ij} = T$; if the baseline intensity is kept constant with $\mu_{ij} = \mu$ for all pairs (i, j), and the kernel applies no discounting ($\kappa \equiv 1$, i.e. $\rho \to 1$), then $\hat{A}_{ij} = A_{ij}\,\mu\,(1{+}\alpha \sum_{t_k \in T_{ij}} w_k)$. Under uniform activity (e.g., in citation graphs, where each observed pair interacts exactly once), this per-pair factor is identical across pairs and reduces to a global rescaling of $\hat{\mathbf{A}}$, which the symmetric normalization (Eq. 7.11) absorbs. The Laplacian operator of DMoN is thus recovered, and then and only then $\mathcal{L}_{\text{DMoN}} = \mathcal{L}_{\text{LMoN}}$ (Eq. 6.16 and 7.1).

If temporal information is available but community memberships are fixed over time instead (*static* community detection on *dynamic* graphs), the model may still be trained with the temporal operator and longitudinal modularity objective, as node lifetimes within communities are accounted for in $\mathbf{P}$. In both described cases, the smoothness term $\mathcal{S}$ (Eq. 7.5) may be omitted ($\omega = 0$), as nodes do not switch communities over time.

7.2 Experimental evaluation and discussion

The central question of this thesis is revisited here in its most demanding regime, where community structure must be inferred jointly from topology, attribute metadata, and temporal dynamics. The hypothesis is that LMoN should improve over single-source and static baselines when the structural signal-to-noise ratio varies over time, while metadata remains comparatively stable, as in the sparse TADC-SBM regime (Section 5.1).

Experimental setup

In the evaluated configuration, the encoder is a GCN applied to a weighted matrix $\tilde{\mathbf{A}}$ and node features $\mathbf{X}$, followed by a $\mathrm{softmax}$ clustering layer producing $\mathbf{C} \in [0, 1]^{n \times k}$, with a fixed number of communities k set to the known ground truth for each dataset.

The embedding dimension was set to $L = [512]$ for experiments with both real-world and synthetic datasets — following previous observation that a two-layer encoder already sufficed to recover ground truths in the aggregated $\eta = 1$ graph (Table 6.2), so to allow the observation of any relative performance gains. The number k of communities is set to the known ground truth for each dataset, as with the presented baselines in Section 6.2.2. Remaining parameters were set to the same values for all experiments: excitation fixed at $\alpha = 1$, decay parameterized to $\beta = -\log(\rho)$ with initial value$\rho = 0.9$, employing an exponential kernel, and baseline intensity fixed to a scalar $\mu = 1$. This setup allows studying the effect of the temporal operator and longitudinal objective without

introducing additional trainable parameters. Learning rates for the community and temporal objectives were set to 0.001, the Hawkes likelihood was assigned a small weight (0.01) to avoid dominating the primary objective, and the remaining hyperparameters were set as the same for DMoN, i.e., collapse regularization $\vartheta = 1.0$ and dropout rate of 0.3 to avoid overfit to spurious optima [166]. Models were trained for a maximum of 1000 epochs and evaluated in a transductive learning setting (Fig. 3.7) on the last epoch.

Note that real-world datasets in this experiment (Table 5.5) all have temporal dynamics but static community memberships. The synthetic SBM graphs (Table 5.2) are evaluated against their last-snapshot ground truths, where the structural signal is at its strongest based on transition probability η (Figure 5.3). In strict analytical terms, the evaluation is then a direct test of whether the combination of a Hawkes-Laplacian operator (Equation 7.9) and longitudinal objective (Eq. 7.2) with mean-membership correction (Eq. 7.3) improve model performance by reweighting the graph to emphasize recent and persistent interactions before message passing and modularity optimization. Using the same convolutional encoder as DMoN keeps the receptive field fixed across models — although an encoder-versus-encoder comparison is out of the scope of this experimental section, it should be noted that different encoder strategies may be employed with either model, as LMoN's is encoder-agnostic and compatible with alternatives for inductive learning and other distinct, more expressive message-passing schemes (Figure 6.1).

7.2.1 Real-world temporal graphs

In summary, on the real-world graphs evaluated, whether the Hawkes-Laplacian operator yields a resolvable improvement over the static DMoN baseline depends on the informativeness of the spatio-temporal signal and its alignment with the target partition.

As shown in Table 7.1, on four of the seven datasets considered for evaluation, the label-based metrics of both models fall within run-to-run variance. The exceptions are *Brain*, where LMoN improves on all three metrics (accuracy, AMI, ARI) by a margin beyond their standard deviation, and *PubMed* and *School*, where DMoN leads by smaller but resolvable margins; elsewhere, the two models display similar performance. This is explained by a mechanistic interpretation: on citation datasets (all except *Brain*, *Dblp* and *School*), each pairwise interaction is observed only once, so the self-exciting term of the Hawkes intensity is close to null and the operator reduces to a mild baseline reweighting of the static graph. The three remaining datasets are precisely those on which node pairs interact repeatedly, so the self-exciting term is active and the reweighted operator can depart from the aggregate — a clear gain on *Brain*, no resolvable difference on *Dblp*, and a small but consistent loss on *School*. Lastly, note that *Patent* has much higher temporal resolution than *PubMed*, so the decay kernel spreads a single interaction across a very long horizon, yielding an advantage for DMoN's static operator, but statistically indistinguishable performance in the larger citation datasets *ArxivAI* and *ArxivMath*.

The structural quality of the communities is here measured by *coverage*, the fraction of edges falling within communities, pooled over the whole graph; and by *conductance*, a more stringent, per-community measure, averaged unweighted across clusters (Footnote 2). On five of the seven datasets, both models achieve near-identical coverage, and DMoN attains lower (better) conductance overall. However, on *Brain*, LMoN achieves a resolvable improvement in coverage and label-based metrics, while conductance is only

slightly better than the baseline. This highlights how maximum-coverage partitions may differ from the target partition itself, and suggests that recency weighting may concentrate each community on a persistent core of recurring interactions. Lastly, on *School*,

Dataset	Model	Acc. ↑	AMI ↑	ARI ↑	Cov. ↑	Cond. ↓
ArxivAI [99]	LMoN	**.539 ± .023**	.291 ± .018	**.260 ± .022**	**.802 ± .002**	.235 ± .010
	DMoN	.521 ± .014	**.298 ± .009**	.251 ± .010	.789 ± .003	**.216 ± .004**
ArxivMath [99]	LMoN	**.323 ± .010**	**.368 ± .006**	**.196 ± .005**	.884 ± .003	.114 ± .003
	DMoN	.319 ± .013	**.368 ± .009**	.193 ± .011	**.885 ± .002**	**.111 ± .003**
Brain [141]	LMoN	**.403 ± .004**	**.470 ± .002**	**.270 ± .002**	.293 ± .001	**.720 ± .001**
	DMoN	.333 ± .037	.397 ± .033	.233 ± .034	**.525 ± .040**	.724 ± .040
Dblp [209]	LMoN	.392 ± .021	.290 ± .013	.172 ± .009	.819 ± .004	.178 ± .004
	DMoN	**.394 ± .019**	**.293 ± .020**	**.179 ± .015**	**.821 ± .003**	**.177 ± .004**
Patent [68]	LMoN	.360 ± .040	.181 ± .031	.137 ± .039	.997 ± .002	.004 ± .002
	DMoN	**.390 ± .065**	**.223 ± .080**	**.175 ± .080**	**.998 ± .001**	**.003 ± .001**
PubMed [146]	LMoN	**.622 ± .001**	.208 ± .003	.192 ± .003	**.926 ± .000**	**.080 ± .000**
	DMoN	**.622 ± .001**	**.216 ± .001**	**.196 ± .001**	.922 ± .001	.084 ± .001
School [106]	LMoN	.981 ± .003	.959 ± .007	.956 ± .008	.933 ± .001	.072 ± .001
	DMoN	**.993 ± .002**	**.986 ± .004**	**.984 ± .004**	**.935 ± .000**	**.070 ± .001**

Table 7.1: Evaluation on real-world datasets. Values are mean ± standard deviation over 5 runs. Highest means are marked in bold; arrows indicate whether higher/lower values are preferred. Dataset properties are described in Table 5.5. Further model comparisons are drawn in Table 6.1.

Dataset	Model	Accuracy ↑	AMI ↑	ARI ↑	Cov. ↑	Cond. ↓
$SBM_{\eta=1.00}$	LMoN	**.894 ± .004**	**.766 ± .007**	**.776 ± .009**	**.471 ± .003**	**.530 ± .003**
	DMoN	.888 ± .015	.755 ± .022	.765 ± .027	.467 ± .010	.535 ± .010
$SBM_{\eta=.75}$	LMoN	**.249 ± .014**	**.056 ± .008**	**.038 ± .006**	.192 ± .002	.810 ± .003
	DMoN	.216 ± .008	.037 ± .004	.024 ± .003	**.193 ± .002**	**.808 ± .002**
$SBM_{\eta=.5}$	LMoN	**.194 ± .005**	**.020 ± .004**	**.012 ± .002**	.183 ± .002	.818 ± .002
	DMoN	.186 ± .005	.013 ± .002	.007 ± .001	**.185 ± .001**	**.816 ± .001**
$SBM_{\eta=.25}$	LMoN	**.177 ± .005**	**.009 ± .002**	**.005 ± .001**	.180 ± .001	.821 ± .001
	DMoN	**.177 ± .002**	.004 ± .002	.003 ± .001	**.182 ± .001**	**.819 ± .001**
$SBM_{\eta=0}$	LMoN	**.182 ± .010**	**.013 ± .007**	**.007 ± .004**	.180 ± .001	.820 ± .001
	DMoN	.169 ± .004	.003 ± .001	.002 ± .001	**.183 ± .001**	**.818 ± .001**

Table 7.2: Evaluation on synthetic datasets. Values are mean ± standard deviation over 5 runs. Highest means marked in bold; arrows show higher/lower value preference. Datasets, baselines, and further model comparisons are described in Table 5.2, Figure 5.3, and Table 6.2, respectively.

the aggregated graph already encodes community structure — recency weighting can then only discard informative history, and LMoN settles consistently but only slightly below the baseline, with remarkably similar coverage and conductance levels.

Therefore, the experimental results are consistent with the reduction property from LMoN to DMoN, and with the thesis's broader position: a temporal community detection objective confers an advantage only when the temporal signal is informative to recover the target partition, and possibly more when it aligns with metadata. Among the real-world datasets, the benefits of LMoN are thus best characterized in networks with repeated interactions, where recurrence carries structure the aggregated graph does not expose and the temporal signal may align with the target partition; elsewhere, the model recovers the static baseline as a special case at no systematic cost, or underperforms slightly when temporal information is uninformative or misleading to the task.

7.2.2 Synthetic temporal graphs

The regime with evolutionary community dynamics is examined next, considering the last-snapshot ground truths of 5 previously generated TADC-SBM graphs (Section 5.1).

Overall, the synthetic benchmark separates the two models more sharply than the real-world evaluation allows, and in the direction the temporal operator was designed to improve: the structural signal-to-noise ratio of the graph decreases with η, while the Hawkes-Laplacian operator reweights (decays) adjacencies to emphasize recent and persistent interactions, which are more likely to be informative about the target partition.

As shown in Table 7.2, LMoN attains higher label-based scores (AMI, ARI) than the baseline DMoN at every level of community stability η. The separation is resolvable outside one standard deviation at every $\eta < 1$; while at $\eta = 1$, where memberships are fully persistent, the two models coincide within variance. The largest absolute gap occurs at $\eta = .75$, which is the level of drift previously expected to be most informative for separating temporal from static objectives. On the structural metrics, the ordering reverses: when $\eta < 1$, DMoN attains marginally higher coverage and lower conductance — highlighting that label agreement is not a simple function of structural quality, and that the temporal operator's advantage is not a byproduct of better coverage or conductance.

Mechanically, as edges in these synthetic graphs are resampled independently at each of the $T = 8$ snapshots (Equation 5.1), the evaluation target is the last-snapshot ground truth. Under drift ($\eta < 1$), the aggregated graph DMoN operates on therefore superposes up to eight different partitions with equal weight, and the target partition is only one of them; the Hawkes-Laplacian operator reweights that same graph by interaction recency, which concentrates edge mass on the most recent snapshots and hence on precisely the partition against which both models are scored. This also accounts for the reversal on coverage and conductance, which are computed on the aggregated graph: a partition tuned to the most recent snapshot must sit slightly worse with respect to the union of all snapshots, so the small structural deficit is the cost the operator pays for the label-based gain, not a defect of the partition it recovers. At $\eta = 1$, every snapshot carries the same memberships, recency weighting has no additional structure to expose, and the two models agree within variance — the behaviour anticipated by the reduction condition $\hat{\mathbf{A}} = \mathbf{A}$ (Section 7.1.3), which is approached in effect on the examined regime.

The results are therefore consistent with the framing discussed in this research: tem-

poral signal confers an advantage only when it carries information about the target partition that a static, aggregated view of the graph does not already capture. In the synthetic graphs, this condition holds whenever $\eta < 1$ — as nodes may transition between communities, the aggregated graph superposes multiple partitions, which the temporal operator models as time-decayed weights — while at $\eta = 1$, edges are resampled but the target partition is persistent (memberships fixed over time), so both models coincide within variance. The trend across η is not monotonic, however: for $\eta \leq .5$, both DMoN and LMoN yield accuracy only slightly above random ($\approx 1/k$), so relative improvements should not be read as absolute performance gains; LMoN also scores slightly lower at $\eta = .25$ than at $\eta = 0$, with variance comparable to the difference, so its behavior in low-η regimes reflects the stochasticity of the generation process and the low signal-to-noise ratio at those levels of drift, rather than a genuine advantage. This mirrors the real-world results (Table 7.1), where *Brain* exhibits informative temporal characteristics, allowing LMoN to improve model performance accordingly, while the remaining datasets coincide within variance or are statistically indistinguishable from the baseline.

Altogether, this evaluation delimits when LMoN's mechanism is useful: it is under drift that the operator earns its advantage, and under persistent memberships that it correctly falls back to the static baseline, retrieving it on static graphs. Holding both models to the same GCN encoder is what allowed this comparison: since the encoder, the receptive field, and every shared hyperparameter are identical, the separation observed at $\eta < 1$ is produced by the operator itself — Hawkes reweighting followed by Laplacian normalization — rather than by additional representational capacity. More expressive encoders allow for more complex message-passing schemes, which may improve performance on both models; but regardless of the encoder employed, the relative advantage of LMoN over DMoN is expected to persist as long as the temporal signal is informative about the target partition. Lastly, note that other (structural) operators may still outperform both neural models in some evaluated regimes, as discussed next.

7.2.3 Ablation study and comparison with spectral baseline

By disabling one component at a time and holding the encoder and all hyperparameters fixed, the ablation study reported in Table 7.3 locates the separation observed under drift in the TADC-SBM experiments within the operator. Two observations follow from it.

First, without the Hawkes reweighting, the model coincides with DMoN on every metric and at every η: the mean-membership correction alone (Equation 4.12) confers no advantage, so disabling the decay recovers the static baseline in practice, as anticipated by the reduction condition where $\hat{\mathbf{A}}$ approximates $\mathbf{A}$ (Section 7.1.3). The gain under drift is therefore attributable to the reweighting itself, rather than to the normalization accompanying it — the Hawkes-Laplacian operator is the component that allows the model to exploit temporal information to improve label-based metrics, being mean-membership correction an insufficient condition for such improvement to be observed.

Second, fixing the retention rate at its initialization ($\rho = 0.9$) recovers only part of that gain: at intermediate levels of community stability, where the temporal signal is most informative about the target partition, this variant sits between the static baseline and the full model on the label-based metrics. Attaining the full gain therefore requires learning the retention rate rather than having it fixed: sharpening the kernel past its initialization

Dataset	Model	Accuracy ↑	AMI ↑	ARI ↑	Cov. ↑	Cond. ↓
$SBM_{\eta=1.00}$	LMoN	$\mathbf{.894 \pm .004}$	$\mathbf{.766 \pm .007}$	$\mathbf{.776 \pm .009}$	$\mathbf{.471 \pm .003}$	$\mathbf{.530 \pm .003}$
	$LMoN_{no\text{-}learn}$	$.892 \pm .007$	$.761 \pm .013$	$.771 \pm .014$	$.469 \pm .006$	$.533 \pm .005$
	$LMoN_{no\text{-}decay}$	$.887 \pm .015$	$.755 \pm .022$	$.764 \pm .027$	$.467 \pm .010$	$.535 \pm .010$
	DMoN	$.888 \pm .015$	$.755 \pm .022$	$.765 \pm .027$	$.467 \pm .010$	$.535 \pm .010$
$SBM_{\eta=.75}$	LMoN	$\mathbf{.249 \pm .014}$	$\mathbf{.056 \pm .008}$	$\mathbf{.038 \pm .006}$	$.192 \pm .002$	$.810 \pm .003$
	$LMoN_{no\text{-}learn}$	$.238 \pm .011$	$.046 \pm .005$	$.031 \pm .004$	$\mathbf{.194 \pm .002}$	$\mathbf{.807 \pm .003}$
	$LMoN_{no\text{-}decay}$	$.217 \pm .009$	$.037 \pm .003$	$.024 \pm .003$	$.193 \pm .002$	$.808 \pm .002$
	DMoN	$.216 \pm .008$	$.037 \pm .004$	$.024 \pm .003$	$.193 \pm .002$	$.808 \pm .002$
$SBM_{\eta=.5}$	LMoN	$\mathbf{.194 \pm .005}$	$\mathbf{.020 \pm .004}$	$\mathbf{.012 \pm .002}$	$.183 \pm .002$	$.818 \pm .002$
	$LMoN_{no\text{-}learn}$	$.187 \pm .006$	$.017 \pm .003$	$.010 \pm .002$	$\mathbf{.185 \pm .001}$	$\mathbf{.816 \pm .001}$
	$LMoN_{no\text{-}decay}$	$.186 \pm .005$	$.013 \pm .002$	$.008 \pm .001$	$\mathbf{.185 \pm .001}$	$\mathbf{.816 \pm .001}$
	DMoN	$.186 \pm .005$	$.013 \pm .002$	$.007 \pm .001$	$\mathbf{.185 \pm .001}$	$\mathbf{.816 \pm .001}$
$SBM_{\eta=.25}$	LMoN	$\mathbf{.177 \pm .005}$	$\mathbf{.009 \pm .002}$	$\mathbf{.005 \pm .001}$	$.180 \pm .001$	$.821 \pm .001$
	$LMoN_{no\text{-}learn}$	$.174 \pm .006$	$.005 \pm .004$	$.003 \pm .002$	$.181 \pm .001$	$.820 \pm .001$
	$LMoN_{no\text{-}decay}$	$\mathbf{.177 \pm .002}$	$.004 \pm .001$	$.003 \pm .001$	$\mathbf{.182 \pm .001}$	$\mathbf{.819 \pm .001}$
	DMoN	$\mathbf{.177 \pm .002}$	$.004 \pm .002$	$.003 \pm .001$	$\mathbf{.182 \pm .001}$	$\mathbf{.819 \pm .001}$
$SBM_{\eta=0}$	LMoN	$\mathbf{.182 \pm .010}$	$\mathbf{.013 \pm .007}$	$\mathbf{.007 \pm .004}$	$.180 \pm .001$	$.820 \pm .001$
	$LMoN_{no\text{-}learn}$	$.174 \pm .008$	$.007 \pm .004$	$.004 \pm .002$	$.181 \pm .002$	$.820 \pm .002$
	$LMoN_{no\text{-}decay}$	$.169 \pm .004$	$.003 \pm .001$	$.002 \pm .001$	$\mathbf{.183 \pm .001}$	$\mathbf{.818 \pm .001}$
	DMoN	$.169 \pm .004$	$.003 \pm .001$	$.002 \pm .001$	$\mathbf{.183 \pm .001}$	$\mathbf{.818 \pm .001}$

Table 7.3: Ablation study of the LMoN model: *no-learn* fixes the retention rate at $\rho = 0.9$; *no-decay* omits the Hawkes reweighting entirely, so the operator reduces to the static Laplacian. The full model and the static DMoN baseline are repeated from Table 7.2 as upper and lower reference points. Values are mean ± standard deviation over 5 runs, with highest means in bold.

concentrates edge mass further on the most recent snapshots, so that message-passing over the reweighted adjacencies resolves the target partition (the last-snapshot ground truth) against earlier partitions, superposed on the aggregated graph when $\eta < 1$.

Moreover, when under drift, the full model attains the lowest coverage and the highest conductance among all variants; and the more a variant attenuates decay, the closer it sits to the static baseline. This reproduces the reversal already observed between LMoN and DMoN (Table 7.2): as coverage and conductance are computed on the aggregated graph, a partition tuned to the most recent snapshots sits worse with respect to the union of all of them. Finally, at $\eta = 1$, the variants coincide within variance, since all node memberships are kept fixed instead (no drift); while at the lowest levels of stability, all variants operate near chance, so any advantages should not be read as genuine gains.

Comparison with Bethe-Hessian operator

A separate experiment was conducted to evaluate the performance of the spectral clustering pathway presented in Section 5.2, employing a static Bethe-Hessian operator.

Dataset	Model	Accuracy ↑	AMI ↑	ARI ↑	Cov. ↑	Cond. ↓
ArxivAI [99]	S(**H**)	.458 ± .005	.241 ± .002	.178 ± .003	.692 ± .001	.319 ± .001
	LMoN	**.539 ± .023**	.291 ± .018	**.260 ± .022**	**.802 ± .002**	.235 ± .010
	DMoN	.521 ± .014	**.298 ± .009**	.251 ± .010	.789 ± .003	**.216 ± .004**
ArxivMath [99]	S(**H**)	.206 ± .005	.238 ± .005	.091 ± .003	.764 ± .005	.265 ± .013
	LMoN	**.323 ± .010**	**.368 ± .006**	**.196 ± .005**	.884 ± .003	.114 ± .003
	DMoN	.319 ± .013	**.368 ± .009**	.193 ± .011	**.885 ± .002**	**.111 ± .003**
Brain [141]	S(**H**)	**.456 ± .001**	**.504 ± .000**	**.305 ± .001**	.313 ± .000	**.682 ± .000**
	LMoN	.403 ± .004	.470 ± .002	.270 ± .002	.293 ± .001	.720 ± .001
	DMoN	.333 ± .037	.397 ± .033	.233 ± .034	**.525 ± .040**	.724 ± .040
Dblp [209]	S(**H**)	.268 ± .002	.107 ± .001	.080 ± .001	.645 ± .011	.377 ± .006
	LMoN	.392 ± .021	.290 ± .013	.172 ± .009	.819 ± .004	.178 ± .004
	DMoN	**.394 ± .019**	**.293 ± .020**	**.179 ± .015**	**.821 ± .003**	**.177 ± .004**
Patent [68]	S(**H**)	**.496 ± .034**	**.289 ± .035**	**.255 ± .068**	.986 ± .008	.022 ± .014
	LMoN	.360 ± .040	.181 ± .031	.137 ± .039	.997 ± .002	.004 ± .002
	DMoN	.390 ± .065	.223 ± .080	.175 ± .080	**.998 ± .001**	**.003 ± .001**
PubMed [146]	S(**H**)	.501 ± .000	.107 ± .000	.100 ± .000	.895 ± .000	.117 ± .000
	LMoN	**.622 ± .001**	.208 ± .003	.192 ± .003	**.926 ± .000**	**.080 ± .000**
	DMoN	**.622 ± .001**	**.216 ± .001**	**.196 ± .001**	.922 ± .001	.084 ± .001
School [106]	S(**H**)	.788 ± .003	.782 ± .004	.678 ± .005	.891 ± .002	.126 ± .002
	LMoN	.981 ± .003	.959 ± .007	.956 ± .008	.933 ± .001	.072 ± .001
	DMoN	**.993 ± .002**	**.986 ± .004**	**.984 ± .004**	**.935 ± .000**	**.070 ± .001**
$SBM_{\eta=1.00}$	S(**H**)	**1.000 ± .000**	**1.000 ± .000**	**1.000 ± .000**	**.558 ± .000**	**.442 ± .000**
	LMoN	.894 ± .004	.766 ± .007	.776 ± .009	.471 ± .003	.530 ± .003
	DMoN	.888 ± .015	.755 ± .022	.765 ± .027	.467 ± .010	.535 ± .010
$SBM_{\eta=.75}$	S(**H**)	**.454 ± .001**	**.158 ± .001**	**.141 ± .001**	**.295 ± .000**	**.705 ± .000**
	LMoN	.249 ± .014	.056 ± .008	.038 ± .006	.192 ± .002	.810 ± .003
	DMoN	.216 ± .008	.037 ± .004	.024 ± .003	.193 ± .002	.808 ± .002
$SBM_{\eta=.5}$	S(**H**)	**.202 ± .003**	**.026 ± .002**	**.016 ± .001**	**.212 ± .001**	**.788 ± .001**
	LMoN	.194 ± .005	.020 ± .004	.012 ± .002	.183 ± .002	.818 ± .002
	DMoN	.186 ± .005	.013 ± .002	.007 ± .001	.185 ± .001	.816 ± .001
$SBM_{\eta=.25}$	S(**H**)	**.192 ± .009**	**.015 ± .003**	**.010 ± .002**	**.205 ± .001**	**.796 ± .000**
	LMoN	.177 ± .005	.009 ± .002	.005 ± .001	.180 ± .001	.821 ± .001
	DMoN	.177 ± .002	.004 ± .002	.003 ± .001	.182 ± .001	.819 ± .001
$SBM_{\eta=0}$	S(**H**)	**.190 ± .011**	**.020 ± .003**	**.012 ± .002**	**.203 ± .001**	**.798 ± .001**
	LMoN	.182 ± .010	.013 ± .007	.007 ± .004	.180 ± .001	.820 ± .001
	DMoN	.169 ± .004	.003 ± .001	.002 ± .001	.183 ± .001	.818 ± .001

Table 7.4: LMoN and DMoN versus spectral clustering on the Bethe-Hessian operator on the aggregated graph, denoted S(**H**). Real-world datasets have fixed community memberships, while synthetic graphs are evaluated against last-snapshot ground truths. Values are mean ± standard deviation over 5 runs. Highest means in bold; arrows indicate higher/lower value preference.

Both neural models and the spectral baseline are evaluated on the same datasets as the previous experiments, with results reported in Table 7.4. On the synthetic graphs, the spectral baseline leads on every metric at every η, and recovers the last-snapshot ground truth exactly at $\eta = 1$: neither neural model matches it, with or without temporal reweighting, although at $\eta \leq .5$ they all operate near chance; This is consistent with the argument that principled algorithmic methods for community detection may outperform neural network-based approaches, and the evaluated regime is the one that favors it the most: TADC-SBM graphs are sparse and generated from a block structure that the operator is asymptotically optimal for, even while their attribute features are mostly well separated around the community centroids (Figure 5.3). In the experiments performed, the encoder does not translate this partial signal into an improvement over the spectral pathway: the temporal operator improves on the static neural baseline under drift, but still underperforms compared to an optimal operator on the aggregated (static) graph.

This behavior, however, reverses on most of the real-world datasets, where the generative assumptions of the operator no longer hold true — as the Bethe-Hessian (Eq. 2.12) in SBM regimes is detectability-optimal [152] with proper selection of the regularizer $r \approx \sqrt{d}$, where d is the average degree of the graph. The spectral baseline leads on label agreement only on *Brain* and *Patent*, and only partially in both cases: on *Brain* it retains the best accuracy, AMI and ARI, while DMoN attains the highest coverage; and on *Patent* the neural models attain near-saturated coverage and conductance, while agreeing less with the reference labels — therefore matching partitions that are structurally clean on the aggregated graph, but aligned with a different partition than the annotated one. Notably, *Brain* is also the dataset on which the temporal operator delivers its clearest real-world gain over DMoN (Table 7.1), which narrows but does not close the distance compared the spectral pathway. On all other real-world datasets instead, both LMoN and DMoN outperform the spectral baseline on label-based and structural metrics with resolvable differences, by margins larger than the observed deviations.

Taken together, the comparison delimits where each pathway is preferable, rather than ranking them globally. Where the structural signal is directly recoverable from the aggregated topology — as it is by construction on the synthetic graphs, fully at $\eta = 1$ and partially under drift, since the target partition is located among those in the superposed snapshots — a spectral method exploiting that structure extracts more of it than the neural pathway with a Laplacian operator recovers, be it with or without temporal reweighting. Where it is not, and node attributes and degree heterogeneity carry information that the spectrum of the aggregate does not expose, the neural pathway leads on most evaluated datasets. A temporal mechanism such as LMoN therefore confers an advantage under two further conditions: its inductive bias — recency and persistence in interaction history — must track information genuinely absent from the static signal, and its encoder must be paired with node features informative enough to exploit jointly with that structure. The gain it delivers over DMoN should accordingly be read as a gain over a neural baseline, which is not in every regime the best available static method — while the spectral pathway is, likewise, not the best available method in every regime. The two pathways may therefore be read as complementary, rather than strictly comparable, and the choice between them should be informed by the characteristics of the graph at hand, as the no free lunch theorem states: no single method is expected to be optimal across all regimes, and the best available method may differ between datasets.

Part IV
Final Remarks

To build truly intelligent machines,
teach them cause and effect.

Judea Pearl (2018)

Conclusions

This thesis has explored the intersection of temporal graph learning and community detection, presenting novel methodologies that advance the state of the art by bridging network science and machine learning, while also contributing to practical applications through free and open-source software development and benchmarking frameworks.

Departing from a question at the intersection of these two literatures — whether learning on graphs and community detection reinforce one another, compete, or prove largely orthogonal in practice — this research has shown that the answer is contingent on the regime of temporal stability and attribute alignment, here analyzed through the lens of spectral theory. Although graph neural networks have reached the state of the art across many graph-related tasks, their advantage over established descriptive and inferential clustering algorithms [135] remains far less settled [102], particularly under joint demands of computational efficiency and real-world recovery accuracy [131].

This work's neural contribution, LMoN, was designed to exploit spatio-temporal and attribute information for community detection tasks. On evaluated real-world datasets, its advantage over the static baseline was resolvable outside variance on a single dataset — where interaction history is genuinely informative about the target partition — while its performance coincided within variance on the other instances. Meanwhile, on synthetic TADC-SBM graphs [129], LMoN yielded consistent improvements, with an ablation attributing them to the learned Hawkes-Laplacian operator. Yet, a principled spectral method applied to the aggregated graph still matched or exceeded the neural models on every synthetic graph, while also outperforming them on two real-world cases.

These findings amount to further evidence for the argument this thesis advances throughout — that a (temporal) neural mechanism confers an advantage only when its inductive bias is able to exploit information genuinely absent from the aggregated (static) signal. In non-attributed graph regimes, algorithmic methods remain the most appropriate choice, and the decisive obstacle is their scalability rather than their accuracy — to which this research contributed with GPU-accelerated extensions and open-source implementations of spectral and modularity-based methods. For neural models, their strength falls onto the attributed regime that, e.g., algorithmic spectral and modularity [116] methods only seldom exploit at scale; and even there, their gains are clearest where structural, attribute, and temporal signals align, rather than as a universal advantage [135] — as the 'no free lunch' theorem states for many optimization problems, compounded, in the case of community detection, by the ill-defined notion of ground truth [131], community structures [44], and the limits of detectability [51] itself.

It was empirically shown that Laplacian operators fall short of such limits, while non-backtracking (Hashimoto-type) operators retain informative spectrum even when the spatial signal becomes weak due to influence of temporal dynamics. Given this bound-

ary condition — that neural gains are contingent on signal alignment rather than universal — this research's most consequential contribution was to make the algorithmic route tractable at scale: GPU-accelerated formulations of Leiden-based modularity optimization and spectral clustering, built on the NVIDIA RAPIDS (cuGraph) ecosystem over a sparse supra-adjacency representation [147]. Against a CPU reference under an equal-work budget, the GPU-based Leiden backend for modularity optimization attained speedups of up to roughly three orders of magnitude, depending on graph density and snapshot count. For the largest graphs instead, the effect is a change in tractability, as temporal graphs that exceed practical time limits on CPU may be processed on GPU devices, employing the same codebase and data processing workflows. Exposed as a zero-code-change backend of the NetworkX-Temporal library [128], moving from a CPU- to a GPU-based pipeline is made possible by changing a single environment variable, turning it into an efficient clustering primitive for downstream tasks such as graph pooling.

However, the spectral case is non-trivial on GPUs: it has been shown that accelerated eigensolvers are currently restricted to symmetric (positive or Hermitian) operators, whereas the non-backtracking matrix that attains the detectability threshold on temporal graphs is by construction asymmetric, as it propagates information along directed edges respecting the arrow of time. Escaping this asymmetry by reformulating the eigenproblem was then proposed: solving the symmetric Bethe-Hessian, whose negative eigenvalues may recover the informative, out-of-bulk directions of the non-backtracking operator [152], provided its regularization parameter is appropriately chosen. However, the full characterization for the temporal Bethe-Hessian, in order to ensure that this property holds, remains an open problem — conjectured to depend on temporal coupling strength, number of snapshots, and degree heterogeneity of the supra-graph [29].

Notably, the development of this theoretical footing, as well as the methodological counterpart of building efficient eigensolvers for asymmetric matrices, is of interest to problems beyond community detection itself — such as for causal learning applications [142]. A number of works have shown how the timing and ordering of links shapes the causal topology of networked systems, i.e. which nodes can possibly influence each other over time. However, learning on large-scale graphs that encode such temporal dependencies is computationally expensive — to which a standard response may be coarse-graining it in a way that preserves the relevant causal structure [59]. An operator grounded in detectability theory, whose temporal supra-graph preserves community identities across time, may therefore offer a *principled pooling primitive* for such tasks.

This work therefore identifies two complementary directions for future research. The first is *neural*: improving differentiable temporal pooling by grounding it in detectability theory — principled rather than heuristic — while retaining its block-pooling form. The second is *algorithmic*: GPU eigensolvers for asymmetric (directed) matrices, dispensing with the symmetric reformulation entirely and enabling the asymptotically optimal non-backtracking operator to be employed at scale [51]. Underlying both is a question for machine learning and network science: whether community structure is an adequate basis for pooling, especially when the downstream task depends on temporally ordered dependencies rather than co-membership alone. Characterizing when community-based pooling preserves these relevant dynamics in the attributed graph regime is, in this research's conclusion, an open problem that future work may first strive to answer, and the one toward which future efforts departing from this thesis may be directed.

Bibliography

[1] S. B. Abdrabbah, R. Ayachi, and N. B. Amor. Collaborative filtering based on dynamic community detection. In *2nd International Conference on Dynamic Networks and Knowledge Discovery - Volume 1229*, DYNAK'14, pages 85–106, Aachen, DEU, 2014. CEUR-WS.org. url: https://ceur-ws.org/Vol-1229/dynak2014_paper8.pdf.

[2] N. Ahmed, V. K. Yalavarthi, and L. Schmidt-Thieme. Motif-aware graph neural networks for networked time series imputation. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press, 2025. ISBN 978-1-57735-897-8. doi: 10.1609/aaai.v39i11.33241.

[3] N. Bandara, T. Kandappu, and A. Misra. Saccadex: Directed acyclic graph-based semi-supervised learning of continuous ocular dynamics from sparse neuromorphic streams. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1384–1394, March 2026. url: https://openaccess.thecvf.com/content/WACV2026/papers/Bandara_SaccadeX_Directed_Acyclic_Graph-based_Semi-Supervised_Learning_of_Continuous_Ocular_Dynamics_WACV_2026_paper.pdf.

[4] S. Bandyopadhyay and V. Peter. Self-expressive graph neural network for unsupervised community detection, 2021. arXiv: https://arxiv.org/abs/2011.14078.

[5] M. Bastian, S. Heymann, and M. Jacomy. Gephi: An open source software for exploring and manipulating networks. *Proceedings of the International AAAI Conference on Web and Social Media*, 3(1):361–362, 3 2009. ISSN 2162-3449. doi: 10.1609/icwsm.v3i1.13937.

[6] F. Battiston, G. Cencetti, I. Iacopini, V. Latora, M. Lucas, A. Patania, J.-G. Young, and G. Petri. Networks beyond pairwise interactions: Structure and dynamics. *Physics Reports*, 874:1–92, 8 2020. ISSN 0370-1573. doi: 10.1016/j.physrep.2020.05.004.

[7] K. Berahmand, F. Daneshfar, E. S. Salehi, Y. Li, and Y. Xu. Autoencoders and their applications in machine learning: a survey. *Artificial Intelligence Review*, 57(2), 2024. ISSN 1573-7462. doi: 10.1007/s10462-023-10662-6.

[8] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3496–3507, 2022. doi: 10.1109/TPAMI.2021.3054830.

[9] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of community in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 10 2008. doi: 10.1088/1742-5468/2008/10/p10008.

[10] V. Brabant, Y. Asgari, P. Borgnat, A. Bonifati, and R. Cazabet. Longitudinal modularity, a modularity for link streams. *EPJ Data Science*, 14(1), 2 2025. ISSN 2193-1127. doi: 10.1140/epjds/s13688-025-00529-x.

[11] J. S. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *2nd NIPS*, NIPS'89, pages 211–217. MIT Press, 1989. url: https://proceedings.neurips.cc/paper/1989/hash/0336dcbab05b9d5ad24f4333c7658a0e-Abstract.html.

[12] S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks?, 2022. arXiv: https://arxiv.org/abs/2105.14491.

[13] J. J. Cai, E. Borenstein, and D. A. Petrov. Broker genes in human disease. *Genome Biology and Evolution*, 2:815–825, 1 2010. doi: 10.1093/gbe/evq064.

[14] A. Caligiuri, E. Emery, L. Ferreira, J. García-Castillo, S. D. Lindner, J. Molina-Hernández, N. A. R. de Almeida Passos, V. H. Ribeiro, M. Sartore, B. Wang, and V. Calleja-Solanas. Time-varying ecological interactions characterise equilibrium and stability, 2025. arXiv: https://arxiv.org/abs/2506.22123.

[15] G. Carcassi, C. A. Aidala, and J. Barbour. Variability as a better characterization of shannon entropy. *European Journal of Physics*, 42(4):045102, 2021. ISSN 1361-6404. doi: 10.1088/1361-6404/abe361.

[16] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. *Time-Varying Graphs and Dynamic Networks*, pages 346–359. Springer Berlin Heidelberg, 2011. ISBN 9783642224508. doi: 10.1007/978-3-642-22450-8_27.

[17] R. Cazabet, S. Boudebza, and G. Rossetti. Evaluating community detection algorithms for progressively evolving graphs, 2020. arXiv: https://arxiv.org/abs/2007.08635.

[18] Q. Chang, C. D. Giurcaneanu, R. Jia, X. Li, G. Hu, X. Cheng, J. Yang, M. Wu, and Y. Zhang. Forget less, generalize more: Unifying temporal and structural adaptation for dynamic graphs, 2026. arXiv: https://arxiv.org/abs/2605.29453.

[19] P.-Y. Chen and A. O. Hero. Deep community detection. *IEEE Transactions on Signal Processing*, 63(21):5706–5719, 11 2015. ISSN 1941-0476. doi: 10.1109/tsp.2015.2458782.

[20] D. Cheng, Y. Zou, S. Xiang, and C. Jiang. Graph neural networks for financial fraud detection: a review. *Frontiers of Computer Science*, 19(9), 2025. ISSN 2095-2236. doi: 10.1007/s11704-024-40474-y.

[21] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for

statistical machine translation. In *2014 Conference on Empirical Methods in Natural Language Processing EMNLP*, pages 1724–1734. Association for Computational Linguistics, 10 2014. doi: 10.3115/v1/D14-1179.

[22] F. Chung. Laplacians and the cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1):1–19, 4 2005. doi: 10.1007/s00026-005-0237-z.

[23] F. Chung and L. Lu. Connected components in random graphs with given expected degree sequences. *Annals of Combinatorics*, 6(2):125–145, Nov. 2002. ISSN 0218-0006. doi: 10.1007/pl00012580.

[24] F. R. K. Chung. *Spectral Graph Theory*. CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1996. doi: 10.1090/cbms/092.

[25] M. Coll, C. A. Joslyn, N. W. Landry, Q. F. Lotito, A. Myers, J. Pickard, B. Praggastis, and P. Szufel. Hif: The hypergraph interchange format for higher-order networks. *Network Science*, 13:e21, 2025. doi: 10.1017/nws.2025.10018.

[26] A. Condon and R. M. Karp. Algorithms for graph partitioning on the planted partition model. *Random Structures and Algorithms*, 18(2):116–140, 2001. ISSN 1098-2418. doi: 10.1002/1098-2418(200103)18:2<116::aid-rsa1001>3.0.co;2-2.

[27] E. Cozzo, G. F. de Arruda, F. A. Rodrigues, and Y. Moreno. Multilayer networks: Metrics and spectral properties. In *Understanding Complex Systems*, pages 17–35. Springer International Publishing, 2016. doi: 10.1007/978-3-319-23947-7_2.

[28] G. Csárdi, T. Nepusz, S. Horvát, V. Traag, F. Zanini, and D. Noom. igraph, 2025. doi: 10.5281/ZENODO.3630268.

[29] L. Dall'Amico, R. Couillet, and N. Tremblay. Community detection in sparse time-evolving graphs with a dynamical bethe-hessian. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546. Preprint available at: https://arxiv.org/abs/2006.04510.

[30] Dask Development Team. *Dask: Library for dynamic task scheduling*, 2016. url: http://dask.pydata.org.

[31] C. Data61. Stellargraph machine learning library, 2018. url: https://github.com/stellargraph/stellargraph.

[32] N. G. de Bruijn. A combinatorial problem. *Proceedings of the Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam*, 49(7): 758–764, 1946. url: https://www.dwc.knaw.nl/DL/publications/PU00018247.pdf.

[33] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering, 2017. arXiv: https://arxiv.org/abs/1606.09375.

[34] Y. Deshpande, A. Montanari, E. Mossel, and S. Sen. Contextual stochastic block models, 2018. arXiv: https://arxiv.org/abs/1807.09596.

[35] E. Dmitriev, M. W. Chekol, and S. Wang. Mgtcom: Community detection in multimodal graphs, 2022. arXiv: https://arxiv.org/abs/2211.06331.

[36] O. Duranthon and L. Zdeborová. Neural-prior stochastic block model. *Machine Learning: Science and Technology*, 4(3):035017, Aug. 2023. ISSN 2632-2153. doi: 10.1088/2632-2153/ace60f.

[37] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, 8:128–140, 1736. url: https://www.cantab.net/users/michael.behrend/repubs/maze_maths/pages/euler_en.html.

[38] A. Failla, R. Cazabet, G. Rossetti, and S. Citraro. Describing group evolution in temporal data using multi-faceted events. *Machine Learning*, 113(10), 2024. ISSN 1573-0565. doi: 10.1007/s10994-024-06600-4.

[39] H. Felippe, F. Battiston, and A. Kirkley. Network mutual information measures for graph similarity. *Communications Physics*, 7(1), 2024. ISSN 2399-3650. doi: 10.1038/s42005-024-01830-3.

[40] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. Preprint available at: https://arxiv.org/pdf/1903.02428.

[41] E. Forsyth and L. Katz. A matrix approach to the analysis of sociometric data: Preliminary report. *Sociometry*, 9(4):340, 1946. ISSN 0038-0431. doi: 10.2307/2785498.

[42] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2 2010. doi: 10.1016/j.physrep.2009.11.002.

[43] S. Fortunato and M. Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 1 2007. ISSN 1091-6490. doi: 10.1073/pnas.0605965104.

[44] S. Fortunato and D. Hric. Community detection in networks: A user guide. *Physics Reports*, 659:1–44, 11 2016. ISSN 0370-1573. doi: 10.1016/j.physrep.2016.09.002.

[45] S. Fortunato and M. E. J. Newman. 20 years of network community detection. *Nature Physics*, 18(8):848–850, 2022. ISSN 1745-2481. doi: 10.1038/s41567-022-01716-7.

[46] L. C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215–239, 1978. ISSN 0378-8733. doi: 10.1016/0378-8733(78)90021-7.

[47] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991. ISSN 1097-024X. doi: 10.1002/spe.4380211102.

[48] K. Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20(3-4):121–136, 1975. doi: 10.1007/bf00342633.

[49] L. Gallo, L. Lacasa, V. Latora, and F. Battiston. Higher-order correlations reveal complex memory in temporal hypergraphs. *Nature Communications*, 15(1), 6 2024. ISSN 2041-1723. doi: 10.1038/s41467-024-48578-6.

[50] F. Gers. Learning to forget: continual prediction with lstm. In *9th International Conference on Artificial Neural Networks: ICANN '99*, volume 1999, pages 850–855. IEE, 1999. doi: 10.1049/cp:19991218.

[51] A. Ghasemian, P. Zhang, A. Clauset, C. Moore, and L. Peel. Detectability thresholds and optimal algorithms for community structure in dynamic networks. *Physical Review X*, 6(3), 7 2016. ISSN 2160-3308. doi: 10.1103/physrevx.6.031005.

[52] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1263–1272. JMLR.org, 2017. url: https://proceedings.mlr.press/v70/gilmer17a.html.

[53] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *National Academy of Sciences*, 99(12):7821–7826, 6 2002. doi: 10.1073/pnas.122653799.

[54] C. Gkartzios, E. Pitoura, and P. Tsaparas. Modularity-fair deep community detection. In *2025 IEEE International Conference on Data Mining (ICDM)*, pages 287–296, 2025. doi: 10.1109/ICDM65498.2025.00036.

[55] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 5 2010. PMLR. url: https://proceedings.mlr.press/v9/glorot10a.html.

[56] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *IEEE International Joint Conference on Neural Networks*. IEEE, 2005. doi: 10.1109/ijcnn.2005.1555942.

[57] A. M. M. M. Gouvêa, D. A. Vega-Oliveros, M. Cotacallapa, L. N. Ferreira, E. E. N. Macau, and M. G. Quiles. Dynamic community detection into analyzing of wildfires events. In *Computational Science and Its Applications extendash ICCSA 2020*, pages 1032–1047. Springer International Publishing, 2020. doi: 10.1007/978-3-030-58799-4_74.

[58] A. Graser. Notebook-based visual analysis of large tracking datasets. In C. Costa and E. Pitoura, editors, *Proceedings of the Workshops of the EDBT/ICDT 2021 Joint Conference, Nicosia, Cyprus, March 23, 2021*, volume 2841 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021. url: https://ceur-ws.org/Vol-2841/BMDA_11.pdf.

[59] D. Grattarola, D. Zambon, F. M. Bianchi, and C. Alippi. Understanding pooling in graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–11, 2022. doi: 10.1109/TNNLS.2022.3190922.

[60] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *22nd ACM SIGKDD*. ACM, 8 2016. doi: 10.1145/2939672.2939754.

[61] P. D. Grünwald. *The Minimum Description Length Principle*. The MIT Press, 3 2007. ISBN 9780262256292. doi: 10.7551/mitpress/4643.001.0001.

[62] R. Guidotti and M. Coscia. *On the Equivalence Between Community Discovery and Clustering*, pages 342–352. Springer International Publishing, 2018. ISBN 9783319761114. doi: 10.1007/978-3-319-76111-4_34.

[63] J. Guo, Z. Han, S. Zhou, J. Li, V. Tresp, and Y. Wang. Continuous temporal graph networks for event-based graph data. In *DLG4NLP 2022*. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.dlg4nlp-1.3.

[64] Y. Guo, L. Yang, S. Hao, and J. Gao. Dynamic identification of urban traffic congestion warning communities in heterogeneous networks. *Physica A: Statistical Mechanics and its Applications*, 522:98–111, 5 2019. doi: 10.1016/j.physa.2019.01.139.

[65] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conference*, SciPy, pages 11–15. SciPy, 6 2008. doi: 10.25080/tcwv9851.

[66] L. Hagen and A. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Tr. on Computer-Aided Design of Int. Circuits and S.*, 11(9):1074–1085, 1992. doi: 10.1109/43.159993.

[67] E. Hajiramezanali, A. Hasanzadeh, N. Duffield, K. R. Narayanan, M. Zhou, and X. Qian. Variational graph recurrent neural networks, 2020. arXiv: https://arxiv.org/abs/1908.09710.

[68] B. Hall, A. Jaffe, and M. Trajtenberg. *The NBER Patent Citation Data File: Lessons, Insights and Methodological Tools*. National Bureau of Economic Research, Oct. 2001. doi: 10.3386/w8498.

[69] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. url: https://proceedings.neurips.cc/paper_files/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html.

[70] J. B. Hansen, A. Cini, and F. M. Bianchi. On time series clustering with graph neural networks. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. url: https://openreview.net/forum?id=MHQXfiXsr3.

[71] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E.

Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 9 2020. doi: 10.1038/s41586-020-2649-2.

[72] A. G. Hawkes. Point spectra of some mutually exciting point processes. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 33(3):438–443, 10 1971. ISSN 1467-9868. doi: 10.1111/j.2517-6161.1971.tb01530.x.

[73] C. He, X. Fei, Q. Cheng, H. Li, Z. Hu, and Y. Tang. A survey of community detection in complex networks using nonnegative matrix factorization. *IEEE Transactions on Computational Social Systems*, 9(2):440–457, 2022. doi: 10.1109/TCSS.2021.3114419.

[74] P. W. Holland, K. B. Laskey, and S. Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 1983. ISSN 0378-8733. doi: https://doi.org/10.1016/0378-8733(83)90021-7.

[75] P. Holme and J. Saramäki. Temporal networks. *Physics Reports*, 519(3):97–125, 10 2012. ISSN 0370-1573. doi: 10.1016/j.physrep.2012.03.001.

[76] P. Holme and J. Saramäki, editors. *Temporal Network Theory*. Springer International Publishing, 2023. ISBN 9783031303999. doi: 10.1007/978-3-031-30399-9.

[77] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[78] W. Hwang, Y. rae Cho, A. Zhang, and M. Ramanathan. Bridging centrality: Identifying bridging nodes in scale-free networks. In *12th ACM SIGKDD*. ACM, 5 2006. url: https://cse.buffalo.edu/tech-reports/2006-05.pdf.

[79] A. Jain and S. K. Sahni. Algorithms for optimal min hop and foremost paths in interval temporal graphs. *Applied Network Science*, 7(1), 8 2022. doi: 10.1007/s41109-022-00499-3.

[80] B. Karrer and M. E. J. Newman. Stochastic blockmodels and community structure in networks. *Physical Review E*, 83(1), 1 2011. ISSN 1550-2376. doi: 10.1103/physreve.83.016107.

[81] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970. doi: 10.1002/j.1538-7305.1970.tb01770.x.

[82] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. Preprint available at: https://arxiv.org/pdf/1412.6980.

[83] T. N. Kipf and M. Welling. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, 2016. url: https://arxiv.org/abs/1611.07308.

[84] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR*, 2017. url: https://arxiv.org/abs/1609.02907.

[85] M. Kivela, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter. Multilayer networks. *Journal of Complex Networks*, 2(3):203–271, 7 2014. ISSN 2051-1329. doi: 10.1093/comnet/cnu016.

[86] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. ISSN 1931-9193. doi: 10.1002/nav.3800020109.

[87] V. Lachi, F. Ferrini, A. Longa, B. Lepri, A. Passerini, and M. Jaeger. Bridging theory and practice in link representation with graph neural networks, 2025. arXiv: https://arxiv.org/abs/2506.24018.

[88] C. M. Le and E. Levina. Estimating the number of communities in networks by spectral methods, 2019. arXiv: https://arxiv.org/abs/1507.00827.

[89] C. Lee and P. Cunningham. Benchmarking community detection methods on social media data, 2013. arXiv: https://arxiv.org/abs/1302.0739.

[90] C. Lee and D. J. Wilkinson. A review of stochastic block models and extensions for graph clustering. *Applied Network Science*, 4(1), 12 2019. doi: 10.1007/s41109-019-0232-2.

[91] J. Leskovec and R. Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology*, 8(1):1–20, 7 2016. ISSN 2157-6912. doi: 10.1145/2898361.

[92] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *Trans. Sig. Proc.*, 67 (1):97–109, 1 2019. ISSN 1053-587X. doi: 10.1109/TSP.2018.2879624.

[93] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2177–2185, Cambridge, MA, USA, 2014. MIT Press. url: https://proceedings.neurips.cc/paper/2014/hash/feab05aa91085b7a8012516bc3533958-Abstract.html.

[94] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks, 2017. arXiv: https://arxiv.org/abs/1511.05493.

[95] R. Liao, Z. Zhao, R. Urtasun, and R. S. Zemel. Lanczosnet: Multi-scale deep graph convolutional networks, 2019. arXiv: https://arxiv.org/abs/1901.01484.

[96] C. D. Linhares, J. R. Ponciano, M. R. Oliveira, A. Soares, A. J. Traina, and A. Kerren. A review on python libraries for temporal network analysis. *Information and Software Technology*, 198:108208, 10 2026. ISSN 0950-5849. doi: 10.1016/j.infsof.2026.108208.

[97] F. Liu, S. Xue, J. Wu, C. Zhou, W. Hu, C. Paris, S. Nepal, J. Yang, and P. S. Yu. Deep learning for community detection: progress, challenges and opportunities. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI'20, 2021. ISBN 9780999241165. doi: 10.24963/ijcai.2020/693.

[98] H. Liu, P. Jiao, M. Gao, C. Chen, and D. Jin. Heterogeneous temporal hypergraph neural network. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, IJCAI '25, 2025. ISBN 978-1-956792-06-5. doi: 10.24963/ijcai.2025/347.

[99] M. Liu, K. Liang, Y. Liu, S. Wang, S. Zhou, and X. Liu. arxiv4tgc: Large-scale datasets for temporal graph clustering, 2023. arXiv: https://arxiv.org/abs/2306.04962.

[100] M. Liu, Y. Liu, K. Liang, W. Tu, S. Wang, S. Zhou, and X. Liu. Deep temporal graph clustering. In *The 12th International Conference on Learning Representations*, 2024. url: https://proceedings.iclr.cc/paper_files/paper/2024/hash/c14d902be45c72833018b2ccfac071e4-Abstract-Conference.html.

[101] Y. Liu, J. Li, Y. Chen, R. Wu, E. Wang, J. Zhou, S. Tian, S. Shen, X. Fu, C. Meng, W. Wang, and L. Chen. Revisiting modularity maximization for graph clustering: A contrastive learning perspective. In *Proc. of the 30th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, KDD '24, pages 1968–1979, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3671967.

[102] A. Longa, V. Lachi, G. Santin, M. Bianchini, B. Lepri, P. Lio, F. Scarselli, and A. Passerini. Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. url: https://openreview.net/forum?id=pHCdMat0gI.

[103] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 39th International Conference on Machine Learning*, page 6, Atlanta, GA, 2013. url: https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

[104] S. Mahdavi, S. Khoshraftar, and A. An. dynnode2vec: Scalable dynamic network embedding. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 12 2018. doi: 10.1109/bigdata.2018.8621910.

[105] L. Mao, M. Kwak, A. H. Kazemipour Ashkezari, Z. Li, P. Cong, Y. Chen, J. H. Phee, S. Kang, J. Li, and C. Zhu. Dynmoco: A novel ai framework to reveal modular substructures of protein from molecular dynamics. *Biophysical Journal*, 2026. ISSN 0006-3495. doi: 10.1016/j.bpj.2026.03.034.

[106] R. Mastrandrea, J. Fournet, and A. Barrat. Contact patterns in a high school: A comparison between data collected using wearable sensors, contact diaries and friendship surveys. *PLOS ONE*, 10(9):e0136497, 9 2015. ISSN 1932-6203. doi: 10.1371/journal.pone.0136497.

[107] C. Matias and V. Miele. Statistical clustering of temporal networks through a dynamic stochastic block model. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(4):1119–1141, 2017. doi: https://doi.org/10.1111/rssb.12200.

[108] M. McPherson, L. Smith-Lovin, and J. M. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27(1):415–444, 8 2001. doi: 10.1146/annurev.soc.27.1.415.

[109] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434. IEEE, 7 2017. doi: 10.1109/cvpr.2017.576.

[110] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33014602.

[111] P. J. Mucha, T. Richardson, K. Macon, M. A. Porter, and J.-P. Onnela. Community structure in time-dependent, multiscale, and multiplex networks. *Science*, 328 (5980):876–878, 5 2010. doi: 10.1126/science.1184819.

[112] R. R. Nadakuditi and M. E. J. Newman. Graph spectra and the detectability of community structure in networks. *Physical Review Letters*, 108(18), 5 2012. ISSN 1079-7114. doi: 10.1103/physrevlett.108.188701.

[113] G. M. Namata, B. London, L. Getoor, B. Huang, and U. Edu. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8. MLG, 2012. url: https://dtai.cs.kuleuven.be/events/mlg2012/papers/11_querying_namata.pdf.

[114] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003. ISSN 1095-7200. doi: 10.1137/s003614450342480.

[115] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(6), 6 2004. doi: 10.1103/physreve.69.066133.

[116] M. E. J. Newman. Modularity and community structure in networks. *National Academy of Sciences*, 103(23):8577–8582, 2006. doi: 10.1073/pnas.0601602103.

[117] M. E. J. Newman and A. Clauset. Structure and inference in annotated networks. *Nature Communications*, 7(1), 2016. ISSN 2041-1723. doi: 10.1038/ncomms11863.

[118] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001. url: https://proceedings.neurips.cc/paper_files/paper/2001/file/801272ee79cfde7fa5960571fee36b9b-Paper.pdf.

[119] NVIDIA Corporation. cuGraph Documentation, 2026. url: https://docs.rapids.ai/api/cugraph/stable/. Accessed: 2026-05-29.

[120] NVIDIA Corporation. cuML Documentation, 2026. url: https://docs.rapids.ai/api/cuml/stable/. Accessed: 2026-05-29.

[121] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis. CuPy: A NumPy-compatible library for NVIDIA GPU calculations. In *Workshop on Machine Learning Systems (LearningSys), Neural Information Processing Systems (NIPS)*, 2017. url: http://learningsys.org/nips17/assets/papers/paper_16.pdf.

[122] D. Otasek, J. H. Morris, J. Bouças, A. R. Pico, and B. Demchak. Cytoscape automation: empowering workflow-based network analysis. *Genome Biology*, 20(1), 2019. ISSN 1474-760X. doi: 10.1186/s13059-019-1758-4.

[123] J. Palowitch, A. Tsitsulin, B. Mayer, and B. Perozzi. Graphworld: Fake graphs bring real insights for gnns. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, pages 3691–3701. ACM, 8 2022. doi: 10.1145/3534678.3539203.

[124] J. Palowitch, A. Tsitsulin, B. Perozzi, and B. A. Mayer. Synthetic graph generation to benchmark graph learning. In *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*, 2022. url: https://openreview.net/forum?id=jdJtWFSC3-S.

[125] P. Panzarasa, T. Opsahl, and K. M. Carley. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology*, 60(5):911–932, 2 2009. ISSN 1532-2890. doi: 10.1002/asi.21015.

[126] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. B. Schardl, and C. E. Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs, 2019. arXiv: https://arxiv.org/abs/1902.10191.

[127] N. Passos, E. Carlini, and S. Trani. Dynamic graph clustering with graph neural networks, 2026. Poster at Learning on Graphs (LoG) Italian Meetup, 9–11 June, 2026, Pisa, Italy.

[128] N. A. Passos, E. Carlini, and S. Trani. Networkx-temporal: Building, manipulating, and analyzing dynamic graph structures. *SoftwareX*, 31:102277, 2025. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2025.102277.

[129] N. A. R. A. Passos, E. Carlini, and S. Trani. Tadc-sbm: a time-varying, attributed, degree-corrected stochastic block model. In *2025 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, 2025. doi: 10.1109/ISCC65549.2025.11326334.

[130] G. A. Pavlopoulos, D. Paez-Espino, N. C. Kyrpides, and I. Iliopoulos. Empirical comparison of visualization tools for larger-scale network analysis. *Advances in Bioinformatics*, 2017:1–8, 2017. ISSN 1687-8035. doi: 10.1155/2017/1278932.

[131] L. Peel, D. B. Larremore, and A. Clauset. The ground truth about metadata and community detection in networks. *Science Advances*, 3(5), 5 2017. ISSN 2375-2548. doi: 10.1126/sciadv.1602548.

[132] L. Peel, T. P. Peixoto, and M. De Domenico. Statistical inference links data and theory in network science. *Nature Communications*, 13(1), 11 2022. ISSN 2041-1723. doi: 10.1038/s41467-022-34267-9.

[133] T. P. Peixoto. The graph-tool python library. *figshare*, 2014. doi: 10.6084/m9.figshare.1164194.

[134] T. P. Peixoto. Hierarchical block structures and high-resolution model selection in large networks. *Physical Review X*, 4(1), 3 2014. ISSN 2160-3308. doi: 10.1103/physrevx.4.011047.

[135] T. P. Peixoto. *9.* Elements in the Structure and Dynamics of Complex Networks. Cambridge University Press, 2023. doi: 10.1017/9781009118897.

[136] T. P. Peixoto, L. Peel, T. Gross, and M. D. Domenico. Graphs are maximally expressive for higher-order interactions, 2026. arXiv: https://arxiv.org/abs/2602.16937.

[137] E. Penaloza and N. Stevens. Changepoint detection in highly-attributed dynamic graphs, 2024. arXiv: https://arxiv.org/abs/2407.06998.

[138] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *20th ACM SIGKDD*. ACM, 8 2014. doi: 10.1145/2623330.2623732.

[139] H. V. Pham, D. H. Thanh, and P. Moore. Hierarchical pooling in graph neural networks to enhance classification performance in large datasets. *Sensors*, 21(18): 6070, 9 2021. doi: 10.3390/s21186070.

[140] M. Pierrelée, A. Reynders, F. Lopez, A. Moqrich, L. Tichit, and B. H. Habermann. Introducing the novel cytoscape app timenexus to analyze time-series data using temporal multilayer networks (tmlns). *Scientific Reports*, 11(1), 7 2021. ISSN 2045-2322. doi: 10.1038/s41598-021-93128-5.

[141] M. G. Preti, T. A. Bolton, and D. Van De Ville. The dynamic functional connectome: State-of-the-art and perspectives. *NeuroImage*, 160:41–54, 10 2017. ISSN 1053-8119. doi: 10.1016/j.neuroimage.2016.12.061.

[142] L. Qarkaxhija, V. Perri, and I. Scholtes. De bruijn goes neural: Causality-aware graph neural networks for time series data on dynamic graphs. In B. Rieck and R. Pascanu, editors, *Proceedings of the First Learning on Graphs Conference*, volume 198 of *Proceedings of Machine Learning Research*, pages 51:1–51:21. PMLR, 09–12 2022. url: https://proceedings.mlr.press/v198/qarkaxhija22a.html.

[143] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In Y. Chang, C. Zhai, Y. Liu, and Y. Maarek, editors, *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, pages 459–467. ACM, 2018. doi: 10.1145/3159652.3159706.

[144] J. Reichardt and S. Bornholdt. Statistical mechanics of community detection. *Physical Review E*, 74(1), 2006. ISSN 1550-2376. doi: 10.1103/physreve.74.016110.

[145] N. A. Reis de Almeida Passos, E. Carlini, and S. Trani. Deep community detection in attributed temporal graphs: Experimental evaluation of current approaches. In *Proceedings of the 3rd Graph Neural Networking Workshop 2024*, GNNet '24, pages 1–6, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400712548. doi: 10.1145/3694811.3697822.

[146] N. A. Reis de Almeida Passos, E. Carlini, and S. Trani. Pubmed-temporal: A dynamic graph dataset with node-level features, 12 2024. doi: 10.5281/zenodo.13932075.

[147] N. A. Reis de Almeida Passos, E. Carlini, and S. Trani. Accelerating dynamic graph clustering on gpu architectures with cugraph, 2026. url: https://nelsonaloysio.github.io/files/preprint/europar2026frame.pdf. To appear in: 6th Workshop on Flexible Resource and Application Management on the Edge (FRAME '26), 24–28 August, 2026, Pisa, Italy (Accepted).

[148] N. A. Reis de Almeida Passos, E. Carlini, and S. Trani. Toward efficient community detection on time-varying attributed graphs, 2026. url: https://nelsonaloysio.github.io/files/preprint/europar2026symposium.pdf. To appear in: PhD Symposium at the 32nd International European Conference on Parallel and Distributed Computing (Euro-Par '26), 24–28 August, 2026, Pisa, Italy (Accepted).

[149] G. Rossetti and R. Cazabet. Community discovery in dynamic networks: A survey. *ACM Comput. Surv.*, 51(2), 2 2018. ISSN 0360-0300. doi: 10.1145/3172867.

[150] G. Rossetti, Pyup.Io Bot, E. T. Hoeven, U. Norman, D. Jorquera, Hanga Dormán, and M. Dorner. Giuliorossetti/dynetx: v0.3.2, 2023. doi: 10.5281/zenodo.3953118.

[151] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein. Temporal graph networks for deep learning on dynamic graphs, 2020. arXiv: https://arxiv.org/abs/2006.10637.

[152] A. Saade, F. Krzakala, and L. Zdeborová. Spectral clustering of graphs with the bethe hessian. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in NIPS*, volume 27. Curran Associates, Inc., 2014. url: https://proceedings.neurips.cc/paper_files/paper/2014/file/d8c5fabf1a4b215168274283f7c7562c-Paper.pdf.

[153] S. Sahu. Heuristic-based dynamic leiden algorithm for efficient tracking of communities on evolving graphs, 2024. arXiv: https://arxiv.org/abs/2410.15451.

[154] Y. Salehi and D. Giannacopoulos. Deep modularity networks with diversity-preserving regularization, 2025. arXiv: https://arxiv.org/abs/2501.13451.

[155] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang. DySAT. In *13th International Conference on Web Search and Data Mining*. ACM, 1 2020. doi: 10.1145/3336191.3371845.

[156] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, 2014. arXiv: https://arxiv.org/abs/1312.6120.

[157] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 1 2009. doi: 10.1109/tnn.2008.2005605.

[158] M. T. Schaub, J.-C. Delvenne, M. Rosvall, and R. Lambiotte. The many facets of community detection in complex networks. *Applied Network Science*, 2(1), 2017. ISSN 2364-8228. doi: 10.1007/s41109-017-0023-6.

[159] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. *Modeling Relational Data with Graph Convolutional Networks*, pages 593–607. Springer International Publishing, 2018. ISBN 9783319934174. doi: 10.1007/978-3-319-93417-4_38.

[160] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: A software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13(11):2498–2504, 11 2003. ISSN 1088-9051. doi: 10.1101/gr.1239303.

[161] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In *12th Int. Conference on AI and Statistics*, volume 5, pages 488–495. PMLR, 4 2009. url: https://proceedings.mlr.press/v5/shervashidze09a.html.

[162] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. doi: 10.1109/34.868688.

[163] M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs, 2017. arXiv: https://arxiv.org/abs/1704.02901.

[164] U. Singer, I. Guy, and K. Radinsky. Node embedding over temporal graphs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, pages 4605–4612. AAAI Press, 2019. ISBN 9780999241141. doi: 10.24963/ijcai.2019/640.

[165] S. Souravlas, S. Anastasiadou, and S. Katsavounis. A survey on the recent advances of deep community detection. *Applied Sciences*, 11(16), 2021. ISSN 2076-3417. doi: 10.3390/app11167179.

[166] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. url: http://jmlr.org/papers/v15/srivastava14a.html.

[167] C. L. Staudt, A. Sazonovs, and H. Meyerhenke. Networkit: A tool suite for large-scale complex network analysis, 2015. arXiv: https://arxiv.org/abs/1403.3005.

[168] L. Stephan and Y. Zhu. Community detection with the bethe-hessian, 2025. arXiv: https://arxiv.org/abs/2411.02835.

[169] J.-L. Stevens, P. Rudiger, and J. Bednar. Holoviews: Building complex visualizations easily for reproducible science. In *Proceedings of the 14th Python in Science Conference*, SciPy, pages 59–66. SciPy, 2015. doi: 10.25080/majora-7b98e3ed-00a.

[170] S. Stramaglia, L. Angelini, C. Marangi, L. Nitti, and M. Pellicoro. *Statistical Physics and the Clustering Problem*, pages 253–272. Springer Berlin Heidelberg, 2004. ISBN 9783662089682. doi: 10.1007/978-3-662-08968-2_15.

[171] X. Su, S. Xue, F. Liu, J. Wu, J. Yang, C. Zhou, W. Hu, C. Paris, S. Nepal, D. Jin, Q. Z. Sheng, and P. S. Yu. A comprehensive survey on community detection with deep learning. *IEEE T. on N.Networks and Learning Systems*, pages 1–21, 2022. doi: 10.1109/tnnls.2021.3137396.

[172] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: Extraction and mining of academic social networks. In *KDD'08*, pages 990–998, 2008. doi: 10.1145/1401890.1402008.

[173] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 5 2015. doi: 10.1145/2736277.2741093.

[174] S. Tardelli, L. Nizzoli, M. Tesconi, M. Conti, P. Nakov, G. D. S. Martino, and S. Cresci. Temporal dynamics of coordinated online behavior: Stability, archetypes, and influence, 2023. arXiv: https://arxiv.org/abs/2301.06774.

[175] W. H. Thompson, P. Brantefors, and P. Fransson. From static to temporal network theory: Applications to functional brain connectivity. *Network Neuroscience*, 1(2): 69–99, 6 2017. ISSN 2472-1751. doi: 10.1162/netn_a_00011.

[176] V. A. Traag, L. Waltman, and N. J. van Eck. From louvain to leiden: guaranteeing well-connected community. *Scientific Reports*, 9(1):5233, 3 2019. ISSN 2045-2322. doi: 10.1038/s41598-019-41695-z.

[177] V. A. Traag, L. Waltman, and N. J. van Eck. leidenalg Documentation, 2026. url: https://leidenalg.readthedocs.io/en/stable/. Accessed: 2026-05-29.

[178] V. A. Traag and L. Šubelj. Large network community detection by fast label propagation. *Scientific Reports*, 13(1):2701, 2 2023. ISSN 2045-2322. doi: 10.1038/s41598-023-29610-z.

[179] A. Tsitsulin, J. Palowitch, B. Perozzi, and E. MÃ¼ller. Graph clustering with graph neural networks. *Journal of Machine Learning Research*, 24(127):1–21, 2023. url: http://jmlr.org/papers/v24/20-998.html.

[180] F. van Merode, H. Boersma, F. Tournois, W. Winasti, N. A. Reis de Almeida Passos, and A. v. d. Ham. Using entropy metrics to analyze information processing within production systems: The role of organizational constraints. *Logistics*, 9(2), 2025. ISSN 2305-6290. doi: 10.3390/logistics9020046.

[181] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks, 2018. arXiv: https://arxiv.org/abs/1710.10903.

[182] D. Verma and M. Meilă. A comparison of spectral clustering algorithms. Technical Report UW-CSE-03-05-01, University of Washington, Department of Computer Science and Engineering, 2003. url: https://sites.stat.washington.edu/spectral/papers/UW-CSE-03-05-01.pdf.

[183] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[184] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4): 395–416, 8 2007. ISSN 1573-1375. doi: 10.1007/s11222-007-9033-z.

[185] C. Wang, S. Pan, R. Hu, G. Long, J. Jiang, and C. Zhang. Attributed graph clustering: A deep attentional embedding approach. In *Twenty-Eighth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 8 2019. doi: 10.24963/ijcai.2019/509.

[186] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks, 2020. arXiv: https://arxiv.org/abs/1909.01315.

[187] Y.-X. Wang and Y.-J. Zhang. Nonnegative matrix factorization: A comprehensive review. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1336–1353, 2013. doi: 10.1109/TKDE.2012.51.

[188] D. P. Williamson. Lecture notes in spectral graph theory, 2016. url: https://people.orie.cornell.edu/dpw/orie6334/Fall2016/.

[189] D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. doi: 10.1109/4235.585893.

[190] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021. doi: 10.1109/TNNLS.2020.2978386.

[191] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan. Inductive representation learning on temporal graphs, 2020. arXiv: https://arxiv.org/abs/2002.07962.

[192] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks?, 2019. arXiv: https://arxiv.org/abs/1810.00826.

[193] K. S. Xu and A. O. Hero. Dynamic stochastic blockmodels for time-evolving social networks. *IEEE Journal of Selected Topics in Signal Processing*, 8(4):552–562, 8 2014. ISSN 1941-0484. doi: 10.1109/jstsp.2014.2310294.

[194] C. Yan and Z. Chang. Modularized tri-factor nonnegative matrix factorization for community detection enhancement. *Physica A: S.Mech.&Ap.*, 533:122–050, 2019. ISSN 0378-4371. url: https://www.sciencedirect.com/science/article/pii/S0378437119311914.

[195] Z. Yan, J. Zhou, L. Gao, Z. Tang, and M. Zhang. An efficient subgraph gnn with provable substructure counting power. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pages 3702–3713, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3671731.

[196] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, MDS '12, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450315463. doi: 10.1145/2350190.2350193.

[197] Z. Yang, W. W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *ICML'26 Proc., Vol.48*, pages 40–48. JMLR.org, 2016. url: https://proceedings.mlr.press/v48/yanga16.html.

[198] J. You, T. Du, and J. Leskovec. Roland: Graph learning framework for dynamic graphs, 2022. arXiv: https://arxiv.org/abs/2208.07239.

[199] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977. ISSN 2153-3806. doi: 10.1086/jar.33.4.3629752.

[200] B. Zhang, C. Fan, S. Liu, K. Huang, X. Zhao, J. Huang, and Z. Liu. The expressive power of graph neural networks: A survey, 2023. arXiv: https://arxiv.org/abs/2308.08235.

[201] D. Zhang, J. Yin, X. Zhu, and C. Zhang. Attributed network embedding via subspace discovery. *Data Mining and Knowledge Discovery*, 33(6):1953–1980, 8 2019. ISSN 1573-756X. doi: 10.1007/s10618-019-00650-2.

[202] H. Zhang, B. Wu, X. Yuan, S. Pan, H. Tong, and J. Pei. Trustworthy graph neural networks: Aspects, methods, and trends. *Proceedings of the IEEE*, 112(2):97–139, 2024. doi: 10.1109/JPROC.2024.3369017.

[203] L. Zhang, Y. Zhao, T. Che, S. Li, and X. Wang. Graph neural networks for image-guided disease diagnosis: A review. *iRADIOLOGY*, 1(2):151–166, 2023. doi: https://doi.org/10.1002/ird3.20.

[204] Y. Zhang, M. Lucas, and F. Battiston. Higher-order interactions shape collective dynamics differently in hypergraphs and simplicial complexes. *Nature Communications*, 14(1), 3 2023. ISSN 2041-1723. doi: 10.1038/s41467-023-37190-9.

[205] Y. Zhao, E. Levina, and J. Zhu. Consistency of community detection in networks under degree-corrected stochastic block models. *The Annals of Statistics*, 40(4), 8 2012. ISSN 0090-5364. doi: 10.1214/12-aos1036.

[206] Y. Zhao, B. Y. Chen, F. Gao, and X. Zhu. Dynamic community detection considering daily rhythms of human mobility. *Travel B. and Society*, 31:209–222, 4 2023. doi: 10.1016/j.tbs.2022.12.009.

[207] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. ISSN 2666-6510. doi: 10.1016/j.aiopen.2021.01.001.

[208] X. Zhu. Learning from labeled and unlabeled data with label propagation, 2002. doi: 10.1.1.14.3864.

[209] Y. Zuo, G. Liu, H. Lin, J. Guo, X. Hu, and J. Wu. Embedding temporal network via neighborhood formation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18. ACM, 7 2018. doi: 10.1145/3219819.3220054.

[210] Y. Zuo, G. Liu, H. Lin, J. Guo, X. Hu, and J. Wu. Embedding temporal network via neighborhood formation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 2857–2866, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3220054.

[211] Álvaro Arroyo, A. Gravina, B. Gutteridge, F. Barbero, C. Gallicchio, X. Dong, M. Bronstein, and P. Vandergheynst. On vanishing gradients, over-smoothing, and over-squashing in gnns: Bridging recurrent and graph learning, 2025. arXiv: https://arxiv.org/abs/2502.10818.