EURO-PAR – Doctoral Symposium – August 27, 2026

Toward Efficient Community Detection on Time-Varying Attributed Graphs

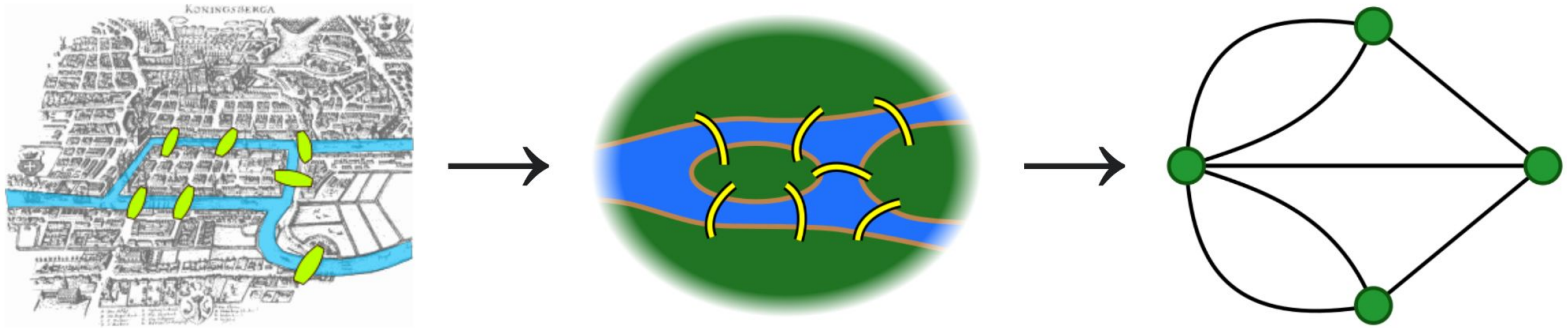
Nelson Aloysio Reis de Almeida Passos[1,2],
Emanuele Carlini[2], Salvatore Trani[2]

[1] University of Pisa, Dep. of Computer Science
[2] National Research Council (ISTI-CNR)

1. Background

The problem and the science

- Many real-world systems can be modeled and studied as **networks** (or **graphs**).
- Elements are **nodes**, connections are **edges** – *often* mathematically constructed as **matrices**.

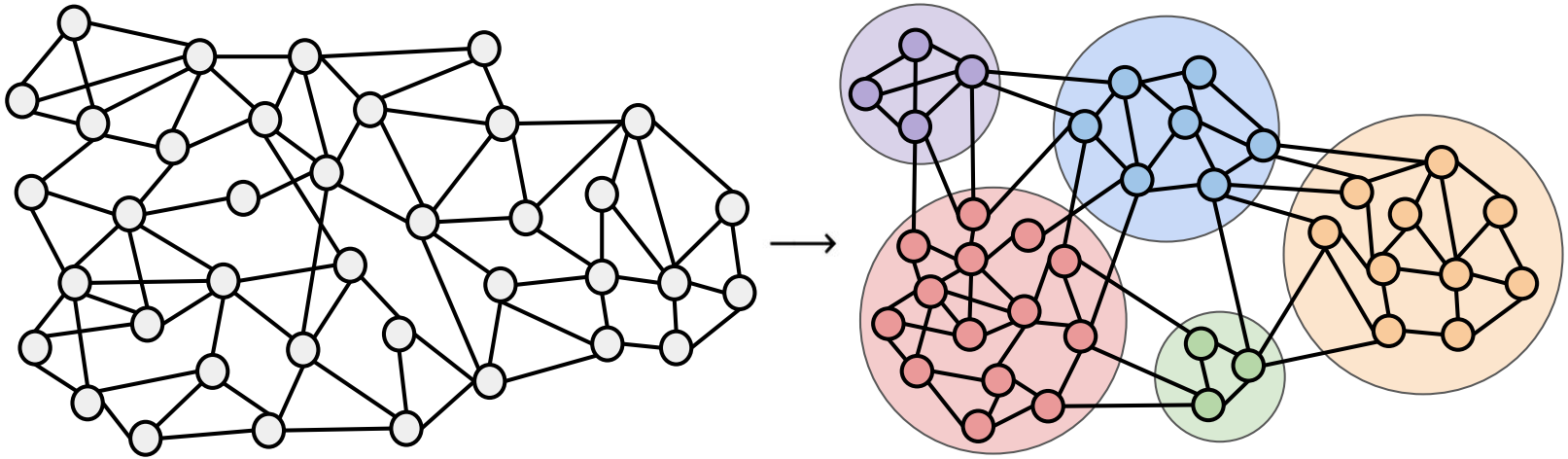


- But the systems we observe are rarely fixed *(static)* in time; instead, they *change* or *evolve.*
- A **temporal network** can be modeled as a **dynamic graph** – with *time-varying* nodes and edges.

1. Background

The problem and the science

- **Our problem: how can we efficiently cluster nodes *(detect communities)* in a graph…**

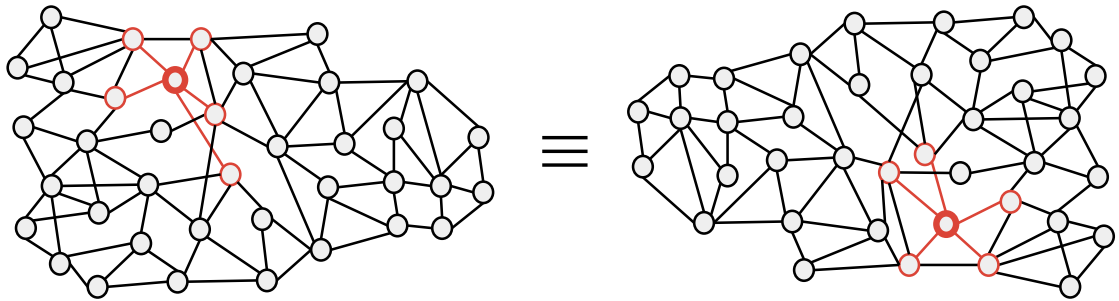


- **…when the graph changes over time and is associated (*attributed*) with additional metadata?**

1. Background

The problem and the science

- Machine learning (ML) models are very good in approximating (*learning* from) lots of data.
- **But** classical ML models can <u>not</u> learn from relational (*non-Euclidean*) datasets directly...



- ...unless first *mapped* to a real-dimensional (*Euclidean*) space – the sort of space *we live in*.
- In the last ten years, a new kind of neural network has emerged that <u>can</u> learn from graphs.

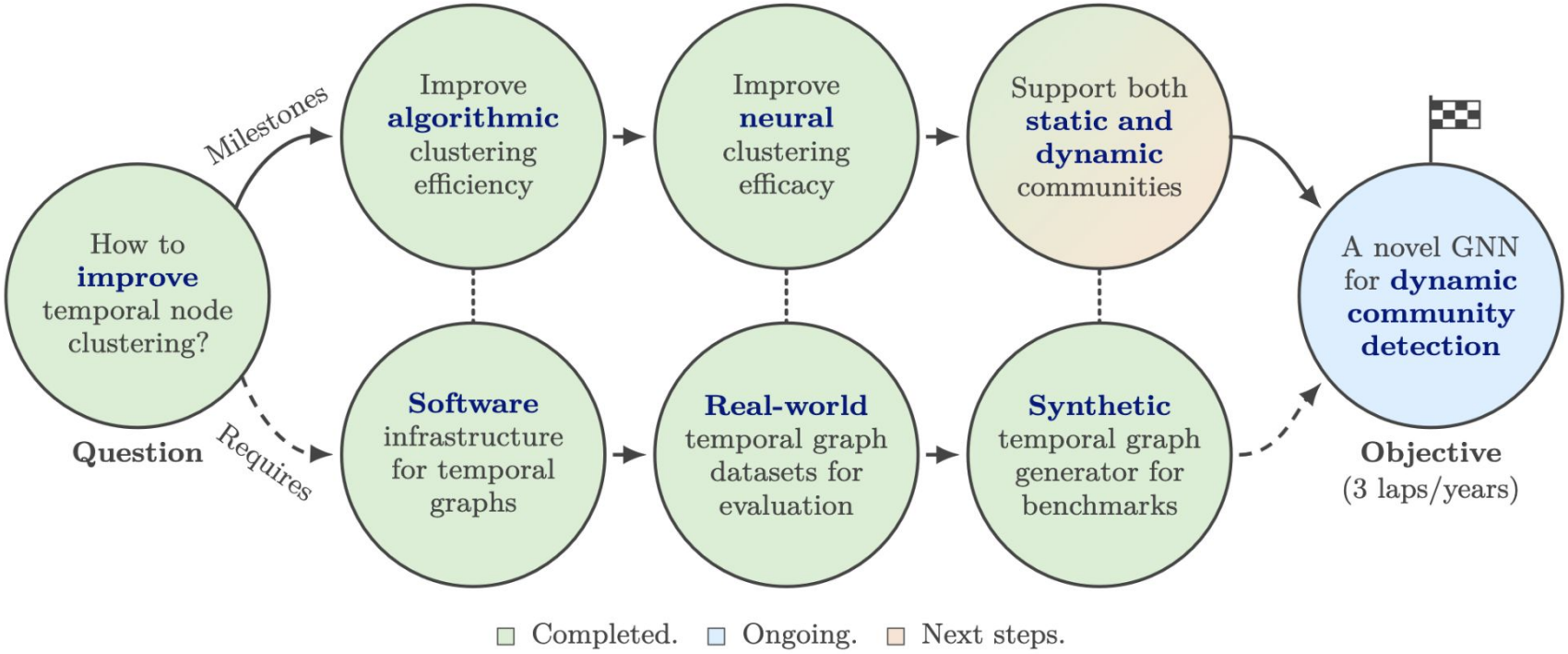
1. Background

The problem and the science

- **How can we efficiently cluster nodes *(detect communities)* in an attributed temporal graph?**
- **State of the art**: node clustering is performed with **algorithms** or **neural** models for graphs.
  - ***Algorithms**: spectral clustering, heuristic optimization, statistical inference...*
  - ***Neural models**: graph neural networks (GNNs), which learn directly on graph-based structures.*
- Methods from both approaches have been adapted **from static to dynamic** settings.
- But another question may be raised: ***when** are neural models **better** than algorithms for the task?*

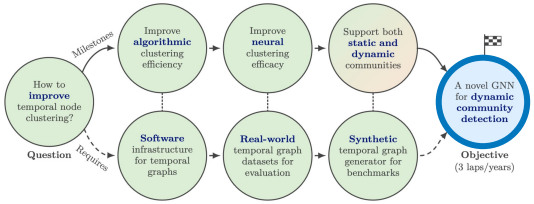
2. Schematic

Research trajectory



3. Contributions

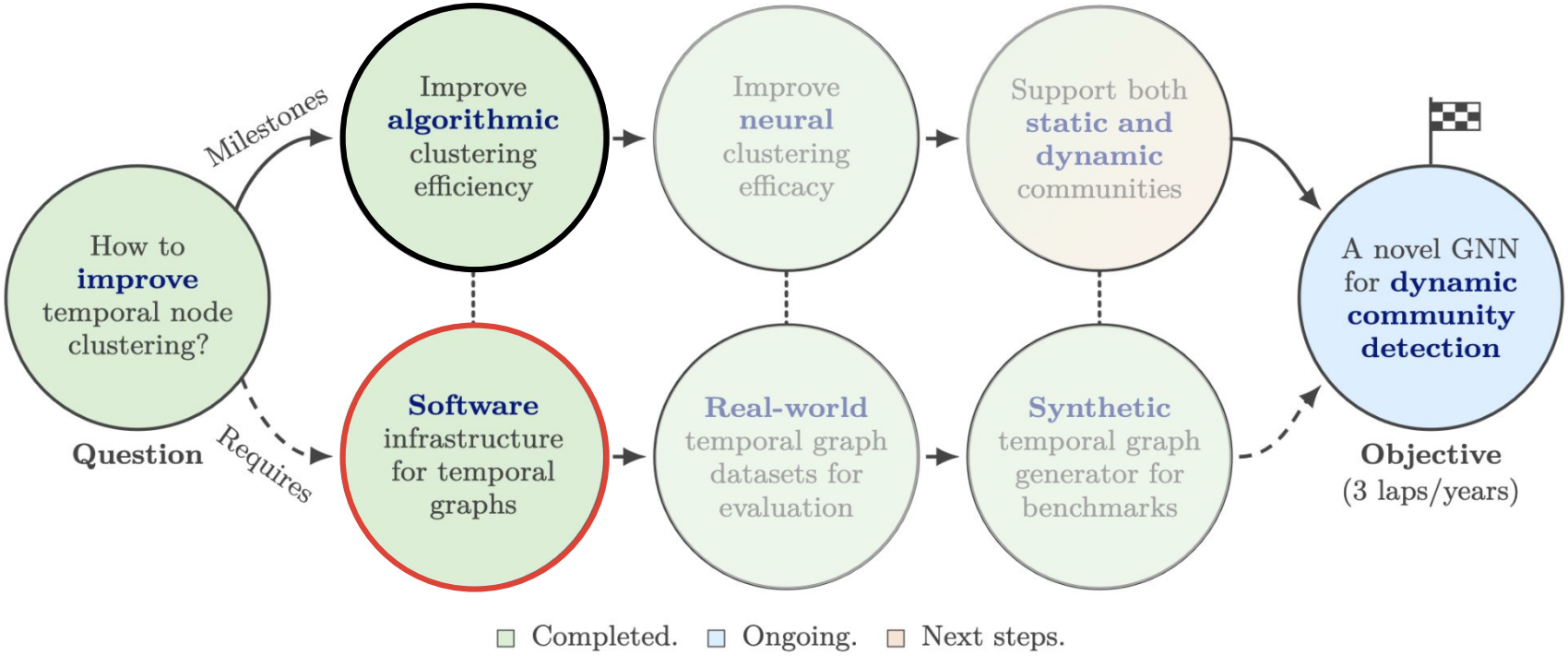
Requirements → Milestones



- Following this **research plan**, our contributions are threefold:
  - Improving algorithmic clustering efficiency (*scalability*)
  - Improving neural clustering efficacy (*performance accuracy*)
  - Supporting static and dynamic community detection (*in temporal graphs*)
- Altogether, they answer the **central research problem**, as well as the *underlying question*:
  - **How to improve temporal node clustering?**
  - *When are neural models better than algorithms for the task?*

3.1. Improve algorithmic clustering efficiency

Requires: *Software infrastructure for temporal graphs*



3.1. Improve algorithmic clustering efficiency

Requires: Software infrastructure for temporal graphs

- At the start of this research, there was **no maintained software ecosystem** for temporal graphs.

3.1. Improve algorithmic clustering efficiency

Requires: Software infrastructure for temporal graphs

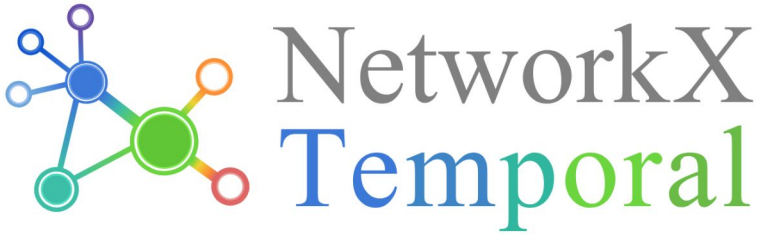
- At the start of this research, there was **no maintained software ecosystem** for temporal graphs.

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.

3.1. Improve algorithmic clustering efficiency

Requires: Software infrastructure for temporal graphs

- The answer became **building** a new one, while **integrating** all other software shown before.



NetworkX-Temporal is a Python package for the creation, manipulation, and analysis of the structure and dynamics of temporal graphs — extending NetworkX with support for time-varying relational data.

Install from PyPI:

```
$ pip install networkx-temporal
```

🔗 networkx-temporal.org

3.1. Improve algorithmic clustering efficiency

Requires: Software infrastructure for temporal graphs

- Open source and freely distributed (3-clause BSD license), the library includes functions for:
  - **Easy snapshot slicing**

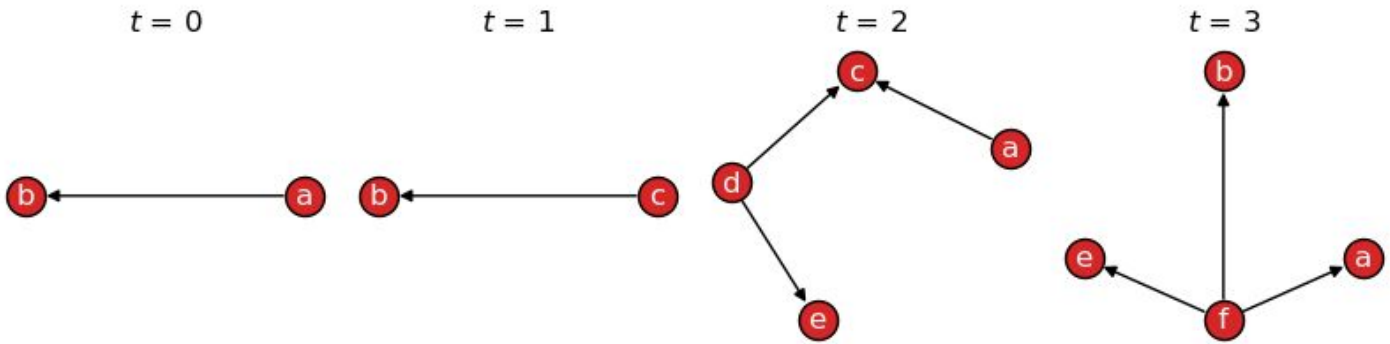
```
>>> import networkx_temporal as tx
>>>
>>> TG = tx.temporal_graph() # TG = tx.TemporalMultiGraph()
>>>
>>> TG.add_edge("a", "b", time=0)
>>> TG.add_edge("c", "b", time=1)
>>> TG.add_edge("d", "c", time=2)
>>> TG.add_edge("d", "e", time=2)
>>> TG.add_edge("a", "c", time=2)
>>> TG.add_edge("f", "e", time=3)
>>> TG.add_edge("f", "a", time=3)
>>> TG.add_edge("f", "b", time=3)
>>>
>>> TG = TG.slice(attr="time")
>>>
>>> print(TG)

TemporalMultiGraph (t=4) with 6 nodes and 8 edges
```

3.1. Improve algorithmic clustering efficiency

Requires: Software infrastructure for temporal graphs

- Open source and freely distributed (3-clause BSD license), the library includes functions for:
  - Easy snapshot slicing
  - **Drawing functions**

```
>>> tx.draw(TG, layout="kamada_kawai", figsize=(8, 2))
```

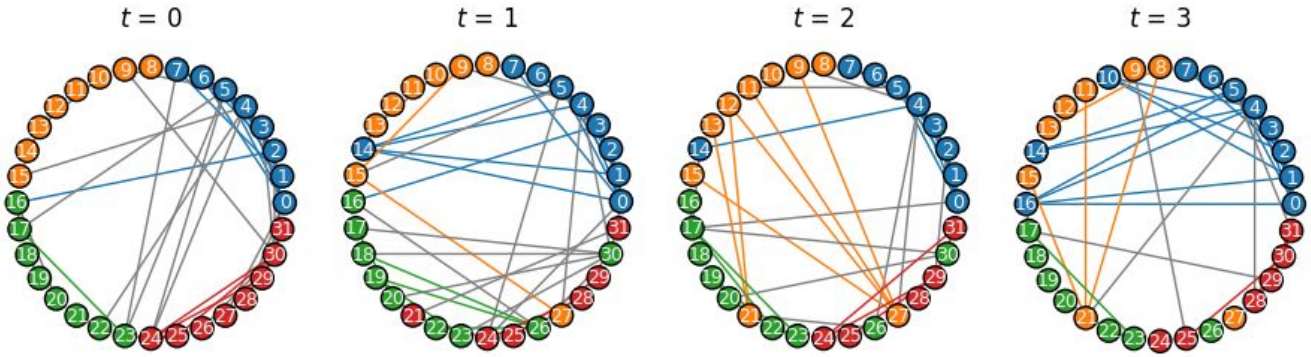


3.1. Improve algorithmic

Requires: Software infrastructure for temporal grap

- Open source and freely distributed (3-claus
  - Easy snapshot slicing
  - Drawing functions
  - **Data loaders/generators**

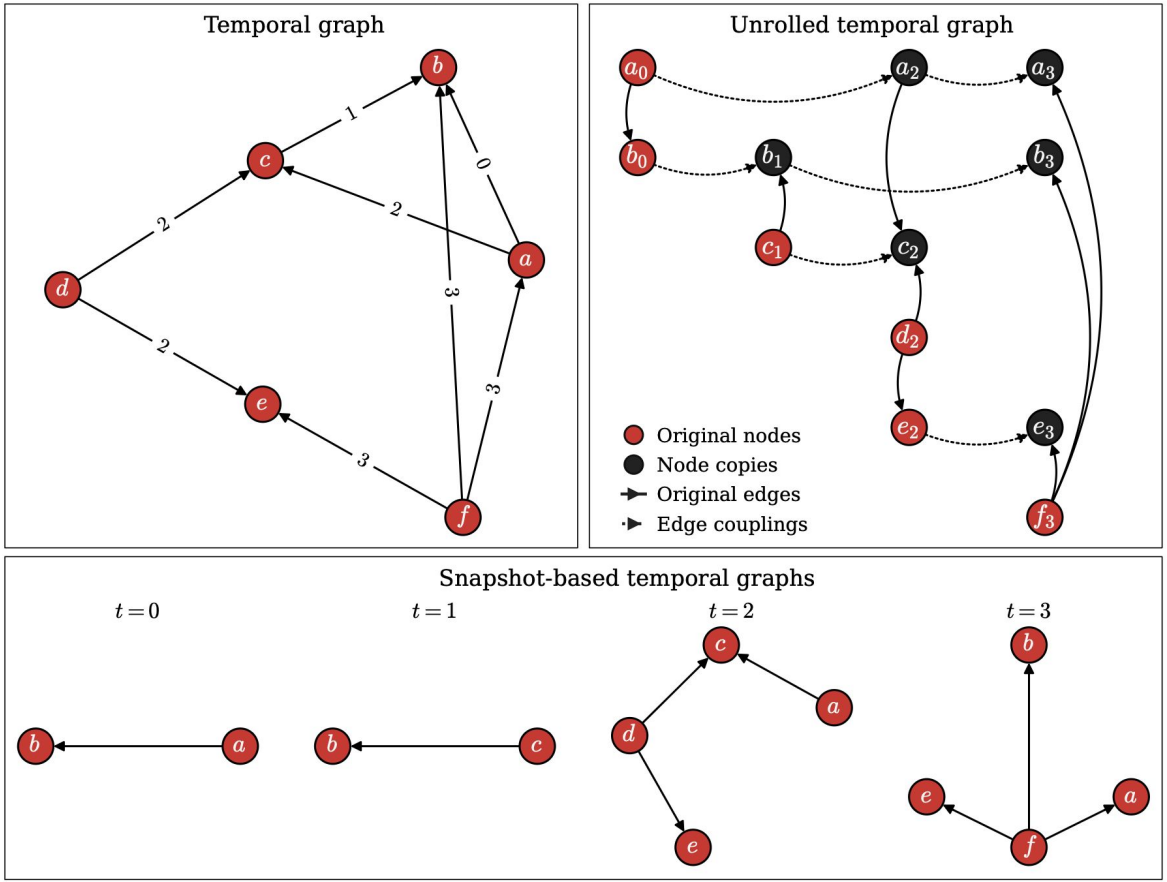
```
>>> import networkx_temporal as tx
>>>
>>> n = 8          # Nodes per community.
>>> k = 4          # Number of communities.
>>> t = 4          # Number of temporal snapshots.
>>> p = 0.8        # Within-community edge probability.
>>> eta = 0.9      # Stability of community memberships.
>>> alpha = 2      # Exponent for degree distribution (Zipf).
>>>
>>> B = tx.generate_block_matrix(k, p)
>>> z = tx.generate_community_vector(n, k)
>>> d = tx.generate_degree_vector([n]*k, max_degree=n, alpha=alpha, seed=0)
>>> tau = tx.generate_transition_matrix(k, eta)
>>>
>>> TG = tx.generators.dynamic_sbm(
>>>     B=B, z=z, d=d, t=t,
>>>     transition_matrix=tau,
>>>     fix_transition_prob=False,
>>>     seed=0
>>> )
>>> print(TG)

TemporalMultiGraph (t=4) with 32 nodes and 97 edges
```



3.1. Improve algorit

Requires: Software infrastructure for te

- Open source and freely distribu
  - Easy snapshot slicing
  - Drawing functions
  - Data loaders/generators
  - **Transform**/Conversion



3.1. Improve algorithmi

Requires: Software infrastructure for temporal g

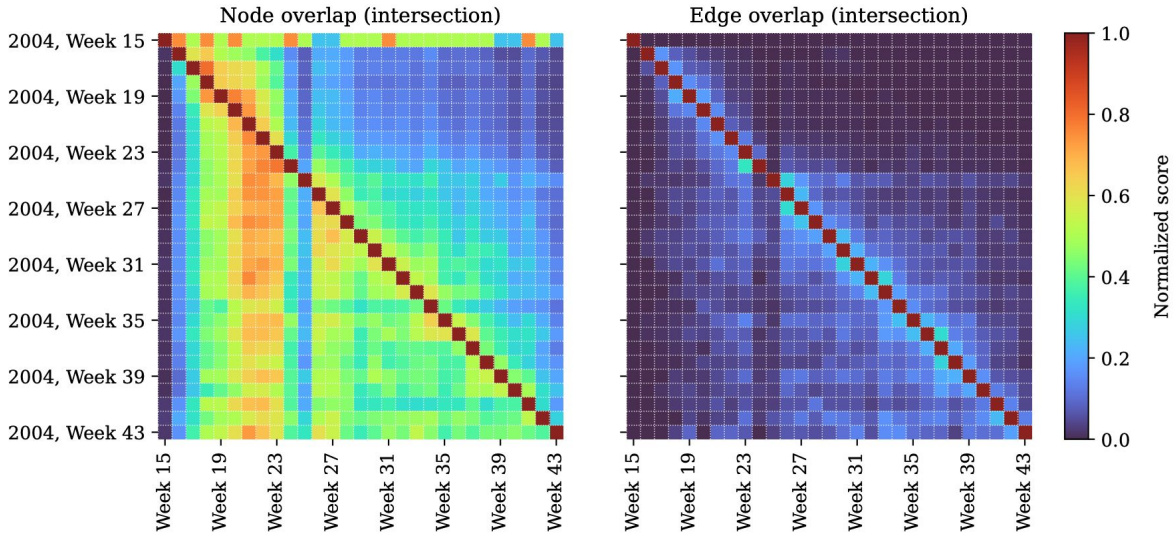
- Open source and freely distributed (3-c
  - Easy snapshot slicing
  - Drawing functions
  - Data loaders/generators
  - Transform/**Conversion**

Format	Parameter (Package)	Calls (Function)
CuGraph	`'cugraph'`	`to_cugraph()`
CuPy	`'cupy'`	`to_cupy()`
Deep Graph Library	`'dgl'`	`to_dgl()`
DyNetX	`'dynetx'`	`to_dynetx()`
graph-tool	`'graph_tool'`	`to_graph_tool()`
igraph	`'igraph'`	`to_igraph()`
NetworKit	`'networkit'`	`to_networkit()`
NumPy	`'numpy'`	`to_numpy()`
Pandas	`'pandas'`	`to_pandas()`
PyTorch Geometric	`'torch_geometric'`	`to_torch_geometric()`
SciPy	`'scipy'`	`to_scipy()`
SNAP	`'snap'`	`to_snap()`
StellarGraph	`'stellargraph'`	`to_stellargraph()`
Teneto	`'teneto'`	`to_teneto()`

3.1. Improve algorithmic clustering efficiency

Requires: Software infrastructure for temporal graphs

- Open source and freely distributed (3-clause BSD license), the library includes functions for:
  - Easy snapshot slicing
  - Drawing functions
  - Data loaders/generators
  - Transform/Conversion
  - **Utility functions**



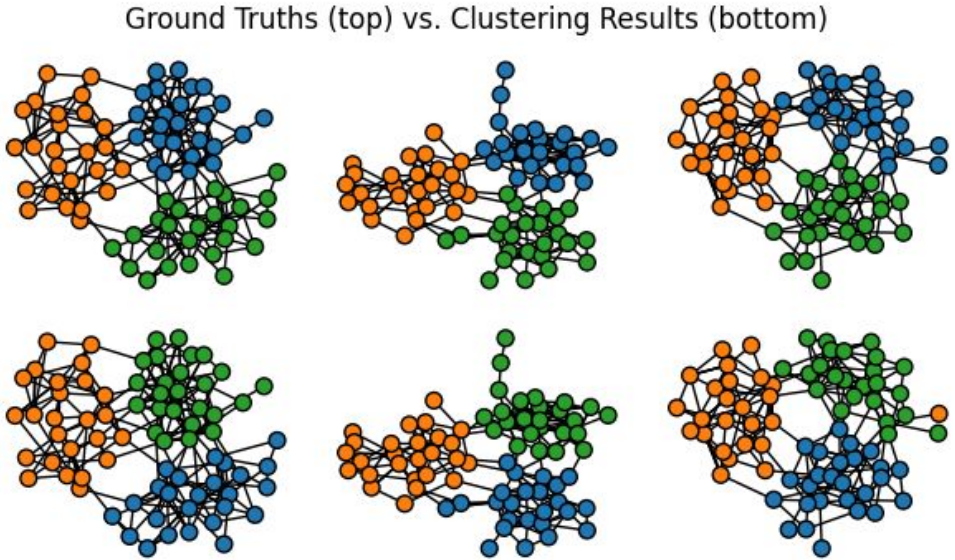
(Weekly node and edge overlap in the CollegeMsg dataset)

3.1. Improve algorithmic clustering efficiency

Requires: Software infrastructure for temporal graphs

- Open source and freely distributed (3-clause BSD license), the library includes functions for:
  - Easy snapshot slicing
  - Drawing functions
  - Data loaders/generators
  - Transform/Conversion
  - Utility functions
  - Community detection and other algorithms (CPU/GPU)

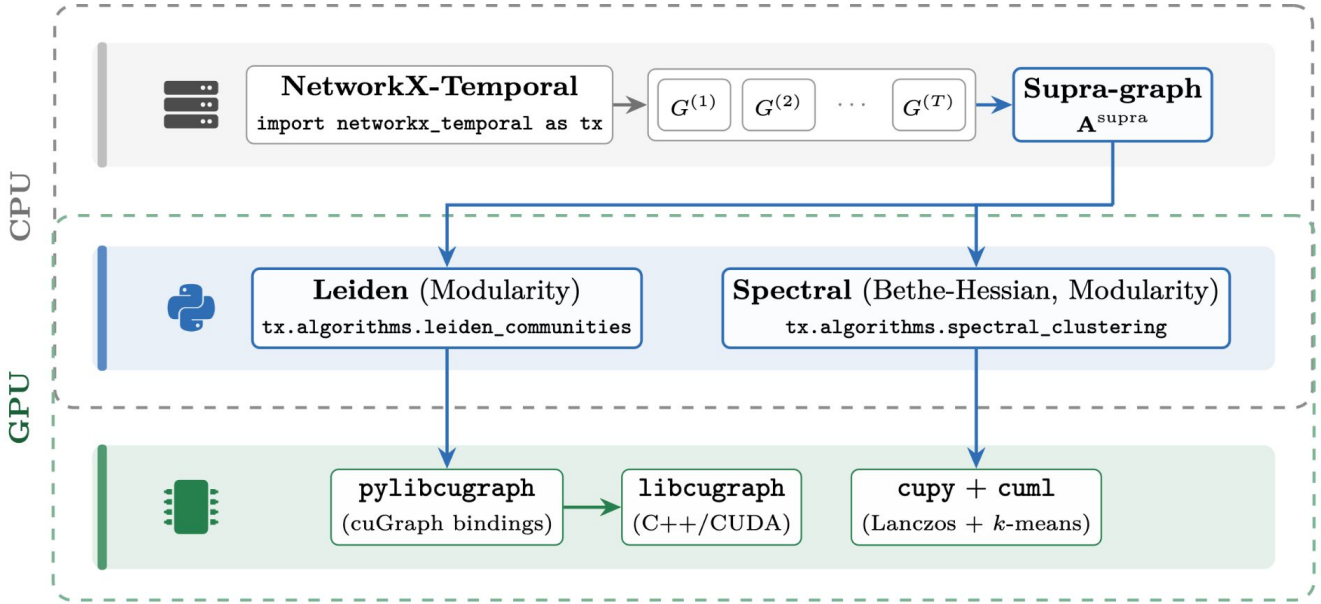
```
>>> y_pred = tx.spectral_clustering(TG, k=3, operator="laplacian")
```



3.1. Improve algorithmic clustering efficiency

Requires: Software infrastructure for temporal graphs

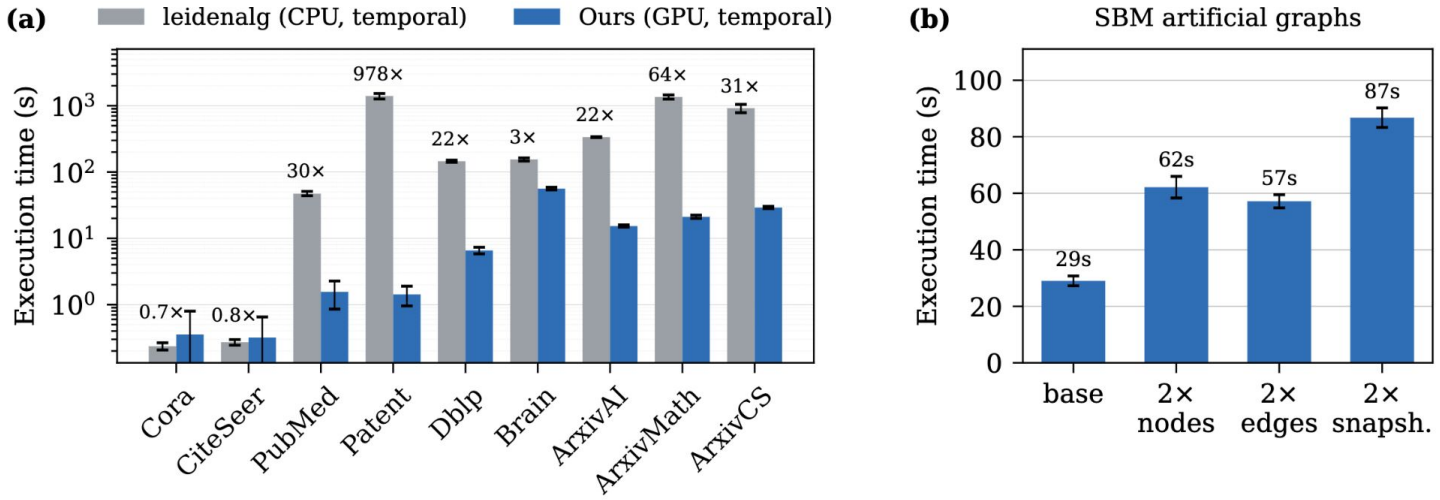
- **Result:** state-of-the-art performance on GPU (both **NVIDIA CUDA** and **AMD ROCm**) devices, with *zero-code-change acceleration* by setting an environment var `NX_GPU_AUTOCONFIG=1`.



3.1. Improve algorithmic clustering efficiency

Requires: Software infrastructure for temporal graphs

- **Result:** state-of-the-art performance on GPU (both **NVIDIA CUDA** and **AMD ROCm**) devices, with *zero-code-change acceleration* by setting an environment var `NX_GPU_AUTOCONFIG=1`.



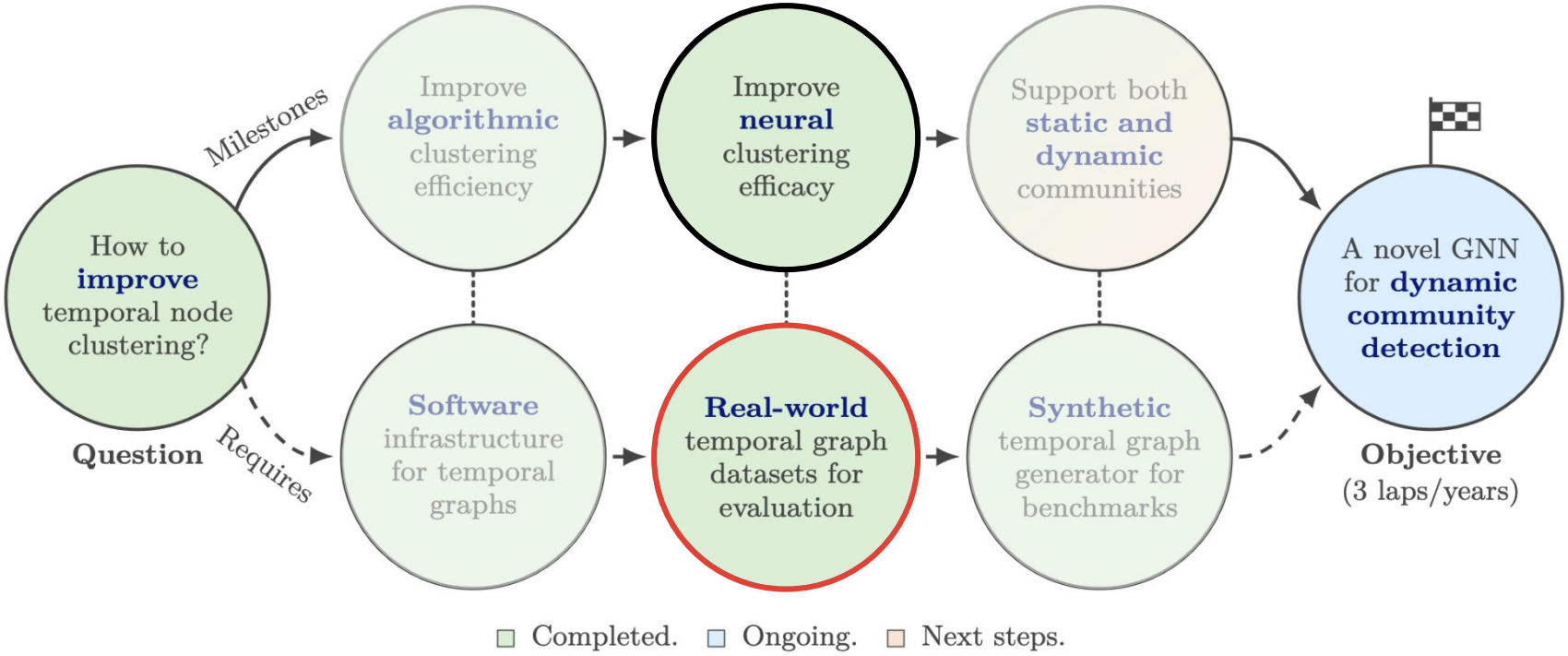
- Speedups of up to ~3 orders of magnitude vs. CPU baselines (Intel Xeon Gold 6330/NVIDIA 100).

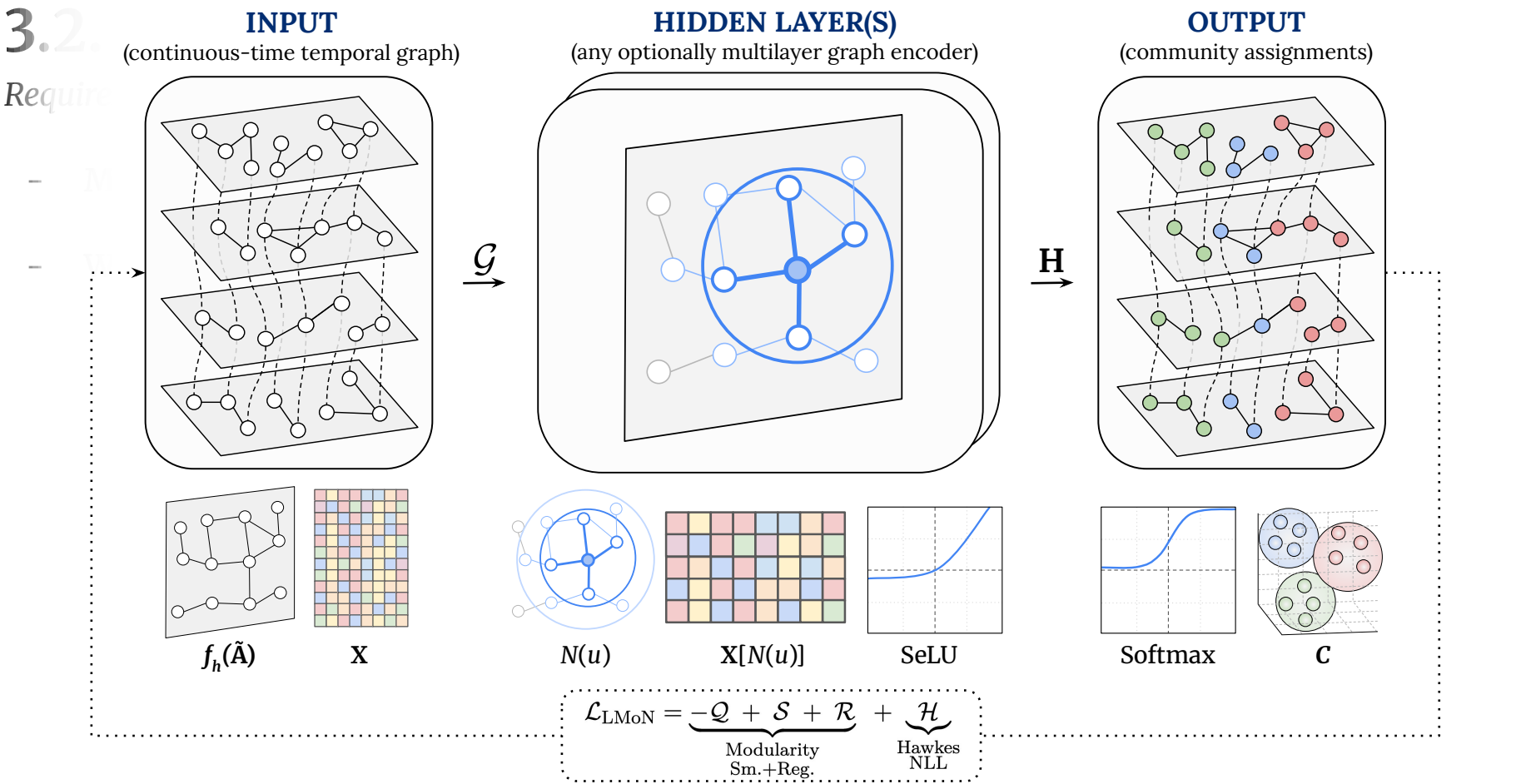
Dataset	$\lvert\mathcal{V}\rvert$	$\Sigma(\mathcal{V})$	$\lvert\mathcal{E}\rvert$	$\Sigma(\mathcal{E})$	$\lvert T\rvert$	$\lvert C\rvert$
CiteSeer† [46]	3 279	3 279	4 552	4 552	1	6
Cora† [46]	2 708	2 708	5 278	5 278	1	7
Brain [33]	5 000	60 000	878 207	947 744	12	10
Dblp [49]	28 085	101 797	153 822	222 165	27	10
Patent [14]	12 214	41 529	41 916	41 916	891	6
PubMed [27]	19 717	37 003	44 324	44 324	42	3
School [19]	327	309 006	7 004	188 508	7 375	9
ArxivAI [17]	69 854	142 316	696 819	696 819	27	5
ArxivCS [17]	169 343	374 433	1 157 799	1 157 799	29	40
ArxivLarge‡ [17]	1 324 064	4 998 391	13 649 351	13 649 351	40	172
ArxivMath [17]	270 013	614 578	783 165	783 165	31	31
SBM [29]	50 000	250 000	1 250 539	1 251 057	5	10
SBM-N [29]	100 000	500 000	2 500 515	2 501 000	5	10
SBM-E [29]	50 000	250 000	2 498 067	2 500 045	5	10
SBM-T [29]	50 000	500 000	2 495 376	2 497 636	10	10

Table 1. Dataset properties. $\lvert\mathcal{V}\rvert$ and $\lvert\mathcal{E}\rvert$ denote unique nodes and edges; $\Sigma(\mathcal{V})$ and $\Sigma(\mathcal{E})$, the sum of nodes and edges over time, i.e., counting each occurrence across snapshots; $\lvert T\rvert$ and $\lvert C\rvert$ are the number of snapshots and communities; † marks static graphs used as single-snapshot baselines; and ‡, the largest dataset, where a single CPU run allowed to exceed the time budget completed in ≈6 h, versus ≈10 min on GPU [24].

3.2. Improve neural clustering efficacy

Requires: *Real-world temporal graph datasets for evaluation*




INPUT
(continuous-time temporal graph)
HIDDEN LAYER(S)
(any optionally multilayer graph encoder)
OUTPUT
(community assignments)
$\mathcal{G}$
$\mathbf{H}$
$f_h(\tilde{\mathbf{A}})$
$\mathbf{X}$
$N(u)$
$\mathbf{X}[N(u)]$
SeLU
Softmax
$\mathbf{C}$
$\mathcal{L}_{\text{LMoN}} = \underbrace{-\mathcal{Q} + \mathcal{S} + \mathcal{R}}_{\text{Modularity Sm.+Reg.}} + \underbrace{\mathcal{H}}_{\text{Hawkes NLL}}$

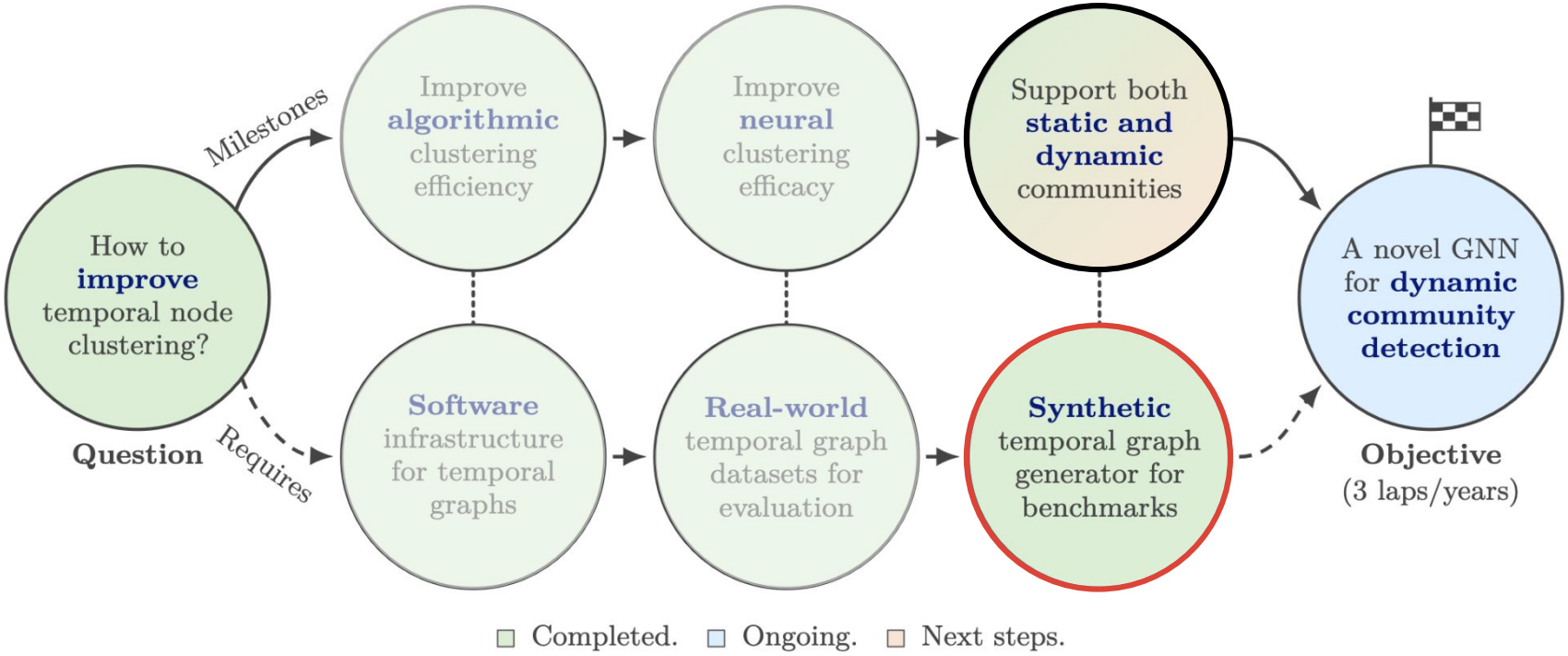
3.2. Improve neural clustering efficacy

Requires: Real-world temporal graph datasets for evaluation

- Most real-world systems have **attributes** at node/edge levels → leading to neural approaches.
- We introduce 🍋 **LMoN** – an *encoder-agnostic* Longitudinal Modularity optimization Network.
- Our model builds upon a SOTA model while maintaining its best-in-class scalability $O(d^2n+m)$:
  - **Objective:** a novel spectral relaxation (*continuous values*) of *longitudinal modularity*.
  - **Operator:** a Hawkes-Laplacian mechanism that *learns* how to reweight the graph.
- **Result:** *end-to-end differentiable* community detection on **attributed temporal** graphs.
- However, **two main issues** compound temporal graph clustering in real-world applications.

3.3. Support static and dynamic communities

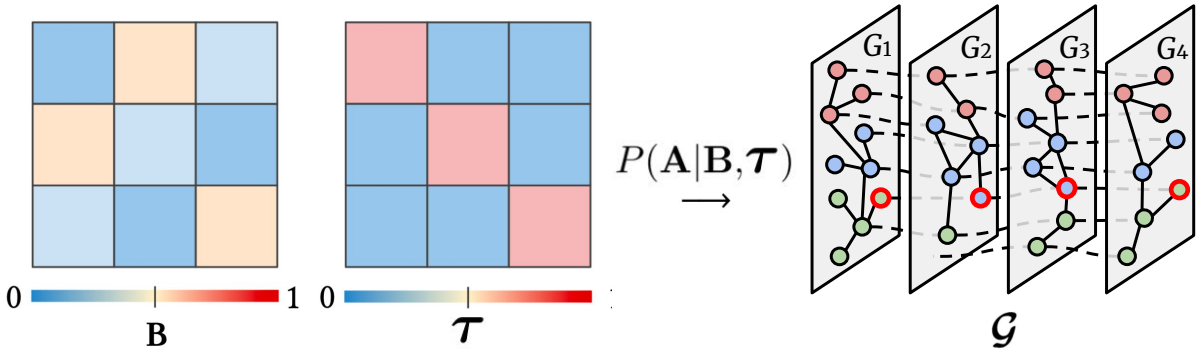
Requires: *Synthetic temporal graph generator for benchmarks*



3.3. Support static and dynamic communities

Requires: Synthetic temporal graph generator for benchmarks

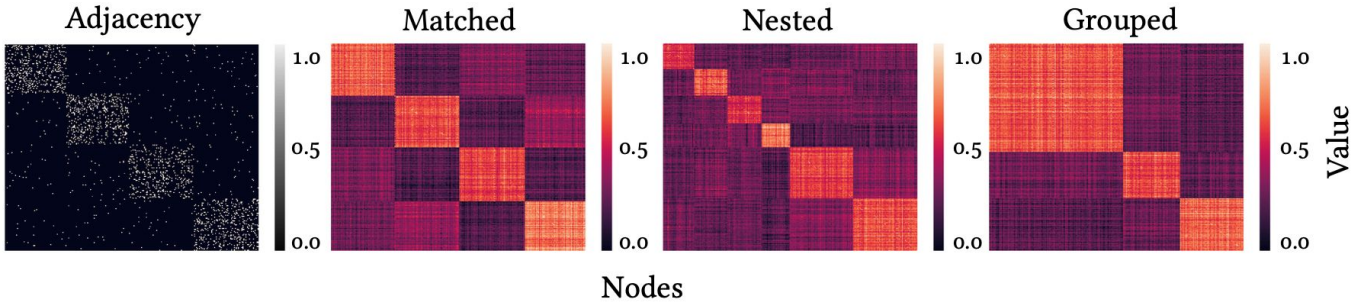
- However, two main issues compound temporal graph clustering in real-world applications.
  - **Real-world data** is *messy*: there is no guarantee of attribute and topological alignment.
  - The **ground truths** (*clusters*) themselves may not be aligned with the graph topology/attr.
- Solution: a *principled* benchmarking framework to compare and evaluate clustering models...



3.3. Support static and dynamic communities

Requires: Synthetic temporal graph generator for benchmarks

- However, two main issues compound temporal graph clustering in real-world applications.
  - **Real-world data** is *messy*: there is no guarantee of attribute and topological alignment.
  - The **ground truths** (*clusters*) themselves may not be aligned with the graph topology/attr.
- Solution: a *principled* benchmarking framework to compare and evaluate clustering models…
- …that generates graphs with *known ground truths* and *community-aligned topology and attributes.*



3.3. Support static and dynamic communities

Requires: Synthetic temporal graph generator for benchmarks

- However, two main issues compound temporal graph clustering in real-world applications.
  - **Real-world data** is *messy*: there is no guarantee of attribute and topological alignment.
  - The **ground truths** (*clusters*) themselves may not be aligned with the graph topology/attr.
- Solution: a *principled* benchmarking framework to compare and evaluate clustering models...
- ...that generates graphs with *known ground truths* and *community-aligned topology and attributes.*
- **Result:** we can evaluate exactly *when and under which conditions* **neural** > **algorithm** methods.

  → Our evaluation revealed: in some real-world applications, **algorithms still outperform GNNs**!

4. Future work

Where do we go from here?

- Two publications “in the oven” on temporal community detection (1 algorithmic, 1 neural method).
- A new research direction for **causal learning at** using *principled pooling primitives* for learning on graphs.
- Our code is available and **production-ready** at networkx-temporal.org (with documentation + examples).
- Questions and feedback appreciated! Thank you for your attention :-)

EURO-PAR CONFERENCE 2026

Toward Efficient Community Detection on Time-Varying Attributed Graphs

nelsonaloysio.github.io

@nelsonaloysio.bsky.social

nelson.reis@phd.unipi.it