ONE IDENTITY, ONE TRAJECTORY: TRAJECTORY-PERSISTENT DYNAMIC GRAPH CONDENSATION FOR EFFICIENT TEMPORAL GRAPH LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Dynamic graphs are ubiquitous in real-world systems, yet their temporal evolution makes repeated training of dynamic graph neural networks (DGNNs) prohibitively expensive. Dynamic graph condensation aims to compress large-scale graph sequences into smaller yet informative synthetic sequences while preserving downstream utility. However, existing snapshot-wise methods condense each snapshot in isolation, discarding the cross-snapshot node identities that DGNNs fundamentally rely on to model temporal dependencies. Accordingly, we propose **T**rajectory-**P**ersistent **D**ynamic **G**raph **C**ondensation (TP-DGC), which shifts the condensation unit from isolated snapshots to node trajectories observed across the entire temporal window. Specifically, TP-DGC compresses structure-aware trajectory states from the real graph into a compact set of persistent synthetic identities. A time-shared assignment enforces index-consistent identities across all snapshots, making persistence intrinsic to the condensed representation rather than a post-hoc alignment. These identities are refined through trajectory-level distribution matching and task-specific semantic alignment. The resulting condensed trajectories are edge-free yet encode structural information through these persistent identities, enabling direct transfer across different DGNN architectures without re-condensation. Experiments on DBLP, Reddit, Brain, and ogbn-arxiv show that TP-DGC outperforms state-of-the-art baselines on node-classification across various condensation budgets, while achieving 7.1×–56.9× speedup over the strongest baseline in total condensation time.

1 INTRODUCTION

Dynamic graphs are ubiquitous in real-world applications, including social media, academic search, recommender systems, and epidemic forecasting (Longa et al., 2023; Su et al., 2024; Zhang et al., 2025). Unlike static graphs, dynamic graphs have node attributes and topology that evolve over time, producing large-scale graph sequences (Pareja et al., 2020). Dynamic graph neural networks (DGNNs) learn from such sequences by jointly modeling structural and temporal dependencies (Sankar et al., 2020; Pareja et al., 2020). However, training DGNNs on full graph histories is prohibitively expensive. This cost has motivated graph condensation (GC) (Jin et al., 2022b; Liu et al., 2022), a data-centric paradigm that replaces large graphs with compact synthetic graphs supporting repeated GNN training while preserving downstream utility.

To adapt GC to large-scale dynamic graph learning, a straightforward strategy is to apply static GC independently to each snapshot and then align or concatenate the resulting synthetic snapshots. This snapshot-wise strategy has two key drawbacks. It severs cross-snapshot temporal dependencies and node correspondence, failing to faithfully preserve the evolutionary dynamics of the original graph. It also requires re-running condensation for each snapshot, so the construction cost scales with the number of snapshots. These limitations have motivated dynamic graph condensation (DGC) (Chen et al., 2026), which condenses the entire sequence jointly.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

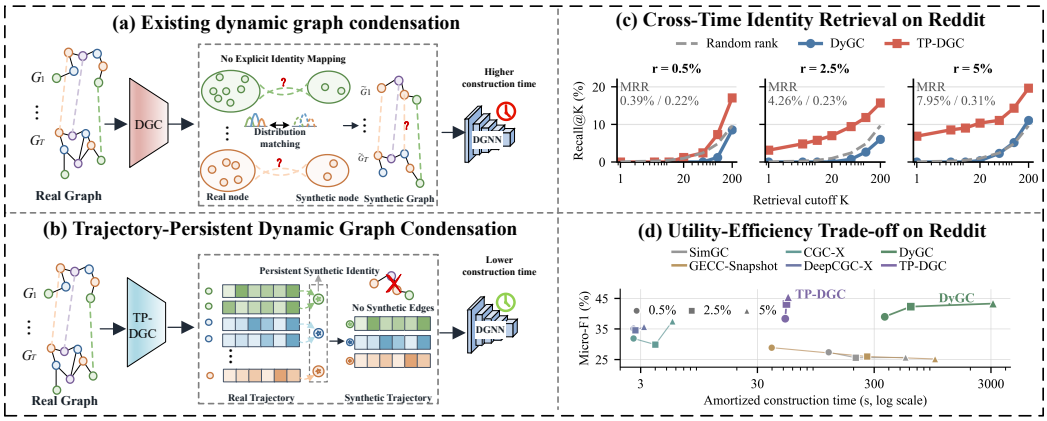

Figure 1: We illustrate the pipeline of existing dynamic graph condensation methods (Fig. 1a), which construct evolving synthetic graph sequences through coarse-grained distribution matching. TP-DGC instead condenses complete node trajectories into an edge-free sequence while maintaining persistent identities (Fig. 1b). It achieves stronger cross-time identity retrieval (Fig. 1c) and a better utility-efficiency trade-off (Fig. 1d) for dynamic graph condensation.

A representative DGC method, DyGC (Chen et al., 2026), condenses the full sequence using shared indices and aligns spatiotemporal distributions via a state-evolution field, yielding cross-snapshot correspondence. Despite this progress, existing DGC methods generally suffer from synthetic identity drift across snapshots (Fig. 1a), arising from two fundamental limitations. First, shared indices serve merely as identifiers and do not enforce semantic correspondence. Second, distribution matching aligns only aggregate statistics and leaves individual node trajectories unconstrained. As a result, the same synthetic node may correspond to different real entities across snapshots, thereby impairing the temporal modeling of DGNNs. Moreover, existing methods incur high computational overhead in optimizing time-varying synthetic edges, especially on large-scale dynamic graphs, which limits transferability across DGNN architectures (Fig. 1d). In summary, an efficient DGC method that preserves consistent synthetic node identities is essential for scalable and practical dynamic graph learning.

To this end, we propose Trajectory-Persistent Dynamic Graph Condensation (TP-DGC), which extends the notion of equivalence from snapshot-level to trajectory-level (Fig. 1b). Trajectory-level equivalence requires the condensed sequence to preserve the complete temporal evolution of each node identity, so that DGNNs can propagate information across time in the same way as on the real graph. Rather than optimizing an evolving synthetic graph sequence, TP-DGC adopts each complete node trajectory as the basic condensation unit and calibrates it under a persistent synthetic identity. Specifically, TP-DGC begins by constructing structure-aware trajectory states from the real dynamic graph. It then initializes persistent synthetic identities using class-conditional trajectory prototypes with a time-shared assignment. A joint refinement stage subsequently combines trajectory distribution matching with task-specific semantic alignment. Since the real topology is encoded into the trajectory states, the resulting condensed sequence is edge-free, which eliminates the need for time-varying synthetic edges and enables transfer across different DGNN architectures without re-condensation.

The main contributions of this work are summarized below:

- We propose TP-DGC, a trajectory-persistent dynamic graph condensation framework that extends equivalence from snapshot-level to trajectory-level. It treats each complete node trajectory as the condensation unit and maintains persistent synthetic identities across snapshots, enabling DGNNs to propagate information along correct temporal paths.
- We design a time-shared assignment scheme with joint calibration for persistent synthetic identities. Unlike per-snapshot assignments, it initializes identities via class-conditional trajectory prototypes and calibrates trajectories through joint distribution matching with semantic alignment, yielding edge-free sequences transferable across DGNN architectures without re-condensation.
- Experiments on DBLP, Reddit, Brain, and ogbn-arxiv demonstrate that TP-DGC outperforms state-of-the-art baselines on node classification across various condensation budgets, while

achieving 7.1×–56.9× speedup in total condensation time. Controlled permutation experiments further show that disrupting cross-snapshot identity correspondence degrades predictive performance.

Prior Work. Motivated by dataset condensation (Wang et al., 2018), graph condensation (GC) constructs compact synthetic graphs that preserve the utility of large graph datasets (Jin et al., 2022b; Liu et al., 2022; Zheng et al., 2023; Xiao et al., 2024). Most existing GC methods focus on static graphs and do not explicitly model temporal dependencies across snapshots. Although dynamic graph neural networks (DGNNs) capture temporal evolution, repeatedly training them on full graph histories remains computationally expensive (Zhao et al., 2020; Seo et al., 2018; Yu et al., 2018; Sankar et al., 2020). Dynamic graph condensation (DyGC) jointly condenses complete graph sequences into compact synthetic sequences for efficient downstream learning (Chen et al., 2026). GECC instead focuses on incrementally updating condensed representations as new graph data arrive (Gong et al., 2026). These methods model temporal evolution primarily at the graph or snapshot level, rather than explicitly treating persistent node trajectories as the basic condensation units. A detailed discussion is provided in **Appendix A**.

2 Problem Formulation

Consider a dynamic graph with T snapshots, a shared node index set, and labels $\mathbf{Y} \in \{1, \ldots, C\}^N$. Dynamic graph condensation maps it to a substantially smaller synthetic sequence,

$$\mathcal{G} = \{G_t(\mathbf{X}_t, \mathbf{A}_t)\}_{t=1}^T \longrightarrow \mathcal{G}' = \{G'_t(\mathbf{X}'_t, \mathbf{A}'_t)\}_{t=1}^T, \qquad M \ll N, \tag{1}$$

where $\mathbf{X}_t \in \mathbb{R}^{N\times d}$ and $\mathbf{A}_t \in \{0,1\}^{N\times N}$. The synthetic snapshots contain M nodes with labels $\mathbf{Y}' \in \{1, \ldots, C\}^M$. A DGNN trained on $\mathcal{G}'$ should achieve comparable performance on downstream tasks to one trained on $\mathcal{G}$.

Let $\{\mathbf{Z}_t\}_{t=1}^T = \Phi(\mathcal{G})$ denote the structure-aware node states extracted from $\mathcal{G}$. We formulate condensation at the trajectory level, mapping complete real-state trajectories to optimizable synthetic feature trajectories:

$$\tau_i^Z = [\mathbf{Z}_{1,i}, \ldots, \mathbf{Z}_{T,i}], \quad \mathcal{T}^Z = \{\tau_i^Z\}_{i=1}^N \longrightarrow \mathcal{T}' = \{\tau'_j\}_{j=1}^M, \quad \tau'_j = [\mathbf{X}'_{1,j}, \ldots, \mathbf{X}'_{T,j}]. \tag{2}$$

3 Trajectory-Persistent Dynamic Graph Condensation

The overall framework of the proposed TP-DGC is illustrated in Fig. 2. TP-DGC implements the trajectory-set formulation through three modules. First, it constructs structure-aware trajectories of real nodes and condenses them into persistent synthetic trajectories with explicit cross-snapshot correspondence. Second, a frozen DGNN teacher transfers task-specific knowledge from the real dynamic graph to the synthetic trajectories. Third, the synthetic trajectories are calibrated by trajectory distribution matching, semantic alignment, and prototype preservation. During calibration, the shared assignment $\mathbf{P}$, synthetic labels $\mathbf{Y}'$, and teacher parameters θ_T remain fixed, while only the synthetic feature sequence $\mathbf{X}'_{1:T}$ is optimized.

3.1 Structure-Aware Persistent Trajectory Construction

Structure-aware state trajectories. Let $\mathcal{G} = \{G_t = (\mathbf{X}_t, \mathbf{A}_t)\}_{t=1}^T$ denote the real dynamic graph. To encode both the current topology and historical node states, we recursively propagate node representations across propagation orders and snapshots (Chen et al., 2026):

$$\mathbf{H}_t^{(k)} = \alpha\mathbf{H}_{t-1}^{(k)} + (1-\alpha)\widehat{\mathbf{A}}_t\mathbf{H}_t^{(k-1)}, \qquad k = 1, \ldots, K, \quad t = 2, \ldots, T, \tag{3}$$

where $\mathbf{H}_t^{(0)} = \mathbf{X}_t$ and $\mathbf{H}_1^{(k)} = \widehat{\mathbf{A}}_1\mathbf{H}_1^{(k-1)}$ specify the order-zero and initial-snapshot boundary states, respectively. Here, $\widehat{\mathbf{A}}_t$ is the symmetrically normalized adjacency with self-loops, K is the propagation depth, and $\alpha \in [0,1]$ controls the contribution of historical states. We concatenate the propagation orders and assemble each node's states over the complete temporal window:

$$\mathbf{Z}_t = \mathbf{H}_t^{(0)} \| \mathbf{H}_t^{(1)} \| \cdots \| \mathbf{H}_t^{(K)}, \qquad \tau_i^Z = [\mathbf{Z}_{1,i}, \ldots, \mathbf{Z}_{T,i}], \qquad \mathcal{T}^Z = \{\tau_i^Z\}_{i=1}^N, \tag{4}$$

where τ_i^Z preserves the ordered structure-aware states of node i, and $\mathcal{T}^Z$ represents the real dynamic graph as a set of complete node trajectories.

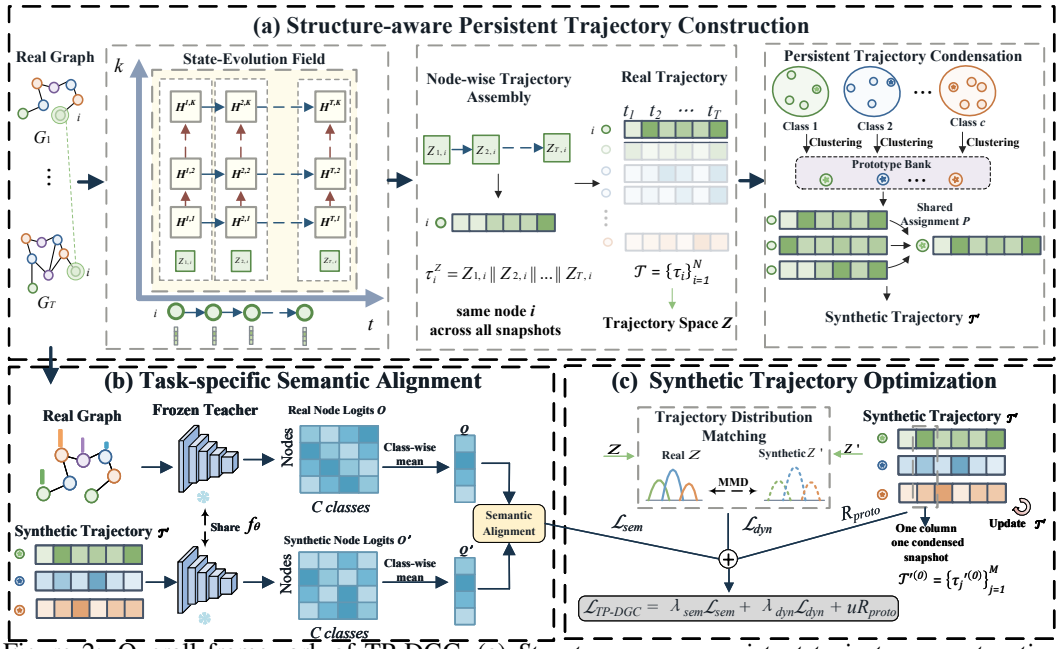

Figure 2: Overall framework of TP-DGC. (a) *Structure-aware persistent trajectory construction* forms real-node trajectories and initializes persistent synthetic identities with class-wise prototypes and a time-shared assignment. (b) *Task-specific semantic alignment* aligns synthetic class-level predictions with those of the real graph through a frozen DGNN teacher. (c) *Synthetic trajectory optimization* jointly applies trajectory-distribution matching, semantic alignment, and prototype preservation regularization.

Class-conditional prototypes and shared assignment. Let $\mathcal{I}_{\text{tr}}$ denote the labeled training-node index set, with $|\mathcal{I}_{\text{tr}}| = N_{\text{tr}}$. We compress these labeled training trajectories into M persistent synthetic trajectories. Each real trajectory is represented by:

$$\mathbf{u}_i = \text{Normalize}\big(\text{vec}(\mathbf{Z}_{1:T,i})\big), \qquad \mathbf{Z}_{1:T,i} = [\mathbf{Z}_{1,i}, \ldots, \mathbf{Z}_{T,i}], \tag{5}$$

where $\mathbf{Z}_{1:T,i}$ contains the ordered structure-aware states of node i over the complete observation window. Given per-class budgets $\{M_c\}_{c=1}^{C}$ with $\sum_c M_c = M$, we perform class-wise k-means++ clustering to select M_c anchors $\{\mathbf{c}_j\}$ for each class (Arthur & Vassilvitskii, 2007). This yields diverse, well-separated class-specific anchors at minimal preprocessing cost, without introducing a learnable clustering module.

Each real training trajectory is assigned to its nearest same-class anchor:

$$a_i = \underset{j:\, y'_j = y_i}{\arg\min} \|\mathbf{u}_i - \mathbf{c}_j\|_2^2, \qquad P_{ij} = \mathbb{I}[a_i = j], \qquad \sum_{j=1}^{M} P_{ij} = 1, \tag{6}$$

where y'_j denotes the label of synthetic trajectory j, and $\mathbf{P} \in \{0,1\}^{N_{\text{tr}} \times M}$ is shared across all snapshots. Consequently, synthetic index j denotes the same synthetic identity throughout the observation window. The assignment is determined in the trajectory-descriptor space, while the trainable synthetic features are initialized by aggregating the original node features according to $\mathbf{P}$:

$$\mathbf{x}'^{(0)}_{t,j} = \frac{\sum_{i \in \mathcal{I}_{\text{tr}}} P_{ij} \mathbf{x}_{t,i}}{\sum_{i \in \mathcal{I}_{\text{tr}}} P_{ij} + \varepsilon}, \qquad t = 1, \ldots, T, \quad j = 1, \ldots, M. \tag{7}$$

The fixed assignment determines the initial grouping of real nodes and establishes a persistent synthetic index. While subsequent calibration may move $\mathbf{X}'_{1:T}$ away from these group means, index j remains fixed across all snapshots. Dataset-specific trajectory descriptors and optimization settings are detailed in **Appendices C.1** and **D.2**.

3.2 Task-Specific Semantic Alignment

The persistent trajectory initialization captures representative temporal patterns, but it does not explicitly preserve task-specific decision information. Moreover, no one-to-one correspondence between real and synthetic nodes is established during condensation. We therefore transfer task-specific knowledge at the class level using a frozen DGNN teacher f_{θ_T} (Hinton et al., 2015). The synthetic sequence is edge-free, containing only self-loops:

$$\mathbf{A}'_t = \mathbf{I}_M, \qquad t = 1, \dots, T, \tag{8}$$

where $\mathbf{I}_M$ is the identity matrix representing self-loops. In implementation, the off-diagonal edge set is empty and self-loops are inserted once during normalization. The teacher produces real and synthetic node-level predictions:

$$\mathbf{O} = f_{\theta_T}(\mathbf{X}_{1:T}, \mathbf{A}_{1:T}), \qquad \mathbf{O}' = f_{\theta_T}(\mathbf{X}'_{1:T}, \mathbf{A}'_{1:T}), \tag{9}$$

where $\mathbf{O} \in \mathbb{R}^{N \times C}$ and $\mathbf{O}' \in \mathbb{R}^{M \times C}$ are the node-level prediction matrices produced after processing the complete sequence, and θ_T remains fixed throughout condensation.

Let $\mathcal{I}_c = \{i \in \mathcal{I}_{\text{tr}} : y_i = c\}$ and $\mathcal{I}'_c = \{j : y'_j = c\}$ denote the real training nodes and synthetic nodes of class c. Their class-level predictive distributions are:

$$\overline{\mathbf{p}}_c = \frac{1}{|\mathcal{I}_c|} \sum_{i \in \mathcal{I}_c} \operatorname{softmax}\left(\frac{\mathbf{O}_i}{\tau}\right), \qquad \overline{\mathbf{p}}'_c = \frac{1}{|\mathcal{I}'_c|} \sum_{j \in \mathcal{I}'_c} \operatorname{softmax}\left(\frac{\mathbf{O}'_j}{\tau}\right), \tag{10}$$

where the temperature $\tau > 0$ controls the smoothness of the distributions. We combine supervision from the fixed synthetic labels with class-level knowledge distillation:

$$\mathcal{L}_{\text{hard}} = -\frac{1}{M} \sum_{j=1}^{M} \log \left[\operatorname{softmax}(\mathbf{O}'_j) \right]_{y'_j}, \tag{11}$$

$$\mathcal{L}_{\text{soft}} = \frac{\tau^2}{C} \sum_{c=1}^{C} \mathrm{KL}\left(\overline{\mathbf{p}}_c \| \overline{\mathbf{p}}'_c\right), \tag{12}$$

$$\mathcal{L}_{\text{sem}} = \mathcal{L}_{\text{hard}} + \beta \mathcal{L}_{\text{soft}}, \tag{13}$$

where KL is the Kullback–Leibler divergence and β balances hard-label supervision against the transferred soft targets. Since the teacher is frozen, gradients from $\mathcal{L}_{\text{sem}}$ update only the synthetic feature sequence.

3.3 Synthetic Trajectory Optimization

The initialized synthetic trajectories establish persistent identities but have not yet been calibrated to represent the real trajectory distribution. We map both sequences into the same structure-aware state space:

$$\mathbf{Z}_{1:T} = \Phi(\mathbf{X}_{1:T}, \mathbf{A}_{1:T}), \qquad \mathbf{Z}'_{1:T} = \Phi(\mathbf{X}'_{1:T}, \mathbf{A}'_{1:T}), \tag{14}$$

where the real states encode the original topology and serve as matching targets, and the synthetic states are derived from the edge-free synthetic sequence defined in Eq. (8). For synthetic index j, let $\tau_j^{Z'} = [\mathbf{Z}'_{1,j}, \dots, \mathbf{Z}'_{T,j}]$ denote its complete structure-aware state trajectory. For class c, define:

$$\mathcal{T}_c^{Z} = \{\tau_i^{Z} : i \in \mathcal{I}_c\}, \qquad \mathcal{T}_c^{Z'} = \{\tau_j^{Z'} : j \in \mathcal{I}'_c\}, \tag{15}$$

where the real set contains labeled training trajectories only. We minimize the class-weighted trajectory-distribution discrepancy:

$$\mathcal{L}_{\text{dyn}} = \frac{\sum_{c=1}^{C} w_c \, \widehat{\mathrm{MMD}}^2(\mathcal{T}_c^{Z}, \mathcal{T}_c^{Z'})}{\sum_{c=1}^{C} w_c}, \qquad w_c = \frac{M_c}{\max_{1 \le r \le C} M_r}, \tag{16}$$

where each kernel sample is a vectorized complete trajectory rather than a single-snapshot node state. For $\mathcal{A} = \{\mathbf{a}_i\}_{i=1}^{n}$ and $\mathcal{B} = \{\mathbf{b}_j\}_{j=1}^{m}$, the biased empirical estimator is:

$$\widehat{\mathrm{MMD}}^2(\mathcal{A}, \mathcal{B}) = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{i'=1}^{n} k(\mathbf{a}_i, \mathbf{a}_{i'}) + \frac{1}{m^2} \sum_{j=1}^{m} \sum_{j'=1}^{m} k(\mathbf{b}_j, \mathbf{b}_{j'}) - \frac{2}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} k(\mathbf{a}_i, \mathbf{b}_j), \tag{17}$$

where the multi-scale RBF kernel is:

$$k(\mathbf{u}, \mathbf{v}) = \sum_{q=-2}^{2} \exp\left(-\frac{\|\mathbf{u} - \mathbf{v}\|_2^2}{2^q \sigma^2}\right). \tag{18}$$

Here, σ is the base RBF bandwidth; its dataset-specific estimation is described in Appendix C.2.

To prevent excessive drift from the class-conditional initialization, we add a prototype anchor in the propagated state space:

$$\mathcal{R}_{\text{proto}} = \frac{1}{MTD_Z} \|\mathbf{Z}'_{1:T} - \mathbf{Z}'^{(0)}_{1:T}\|_F^2, \tag{19}$$

where D_Z is the state dimension and $\mathbf{Z}'^{(0)}_{1:T}$ is obtained from the initialized synthetic sequence. The complete objective is:

$$\mathcal{L}_{\text{TP-DGC}} = \lambda_{\text{dyn}} \mathcal{L}_{\text{dyn}} + \lambda_{\text{sem}} \mathcal{L}_{\text{sem}} + \mu \mathcal{R}_{\text{proto}}. \tag{20}$$

Here, λ_{dyn}, λ_{sem}, and μ weight trajectory matching, semantic alignment, and prototype preservation, respectively. During calibration, $\mathbf{P}$, $\mathbf{Y}'$, and θ_T remain fixed, while gradients update only $\mathbf{X}'_{1:T}$. The final condensed dynamic graph is:

$$\mathcal{G}' = \{(\mathbf{X}'_t, \mathbf{I}_M, \mathbf{Y}')\}_{t=1}^{T}. \tag{21}$$

Because $\mathbf{I}_M$ contains no connections between distinct synthetic nodes, TP-DGC neither generates nor optimizes synthetic edges. The saved feature sequence and labels can be used directly by downstream temporal graph learners without re-condensation.

4 Experiments

4.1 Experimental Setup

Datasets. We conduct experiments on DBLP, Reddit, Brain, and ogbn-arxiv (denoted as Arxiv in tables). DBLP, Reddit, and ogbn-arxiv are Web-derived dynamic graphs, whereas Brain serves as a cross-domain stress test. DBLP and Reddit use $r \in \{0.5\%, 2.5\%, 5.0\%\}$, Brain uses $r \in \{1.0\%, 2.5\%, 5.0\%\}$, and ogbn-arxiv uses $r \in \{0.05\%, 0.25\%, 0.5\%\}$, where the condensation ratio is defined as $r = M/N$. Results on the full graph are reported only as an uncondensed reference. DBLP and Reddit are processed from the released DyGC files (Chen et al., 2026), Brain is obtained from the Brain10 data released with IDGNN (Zhong et al., 2024), and ogbn-arxiv is obtained from the Open Graph Benchmark (Hu et al., 2020).

Baselines. We compare TP-DGC with Random, Herding (Welling, 2009), K-Center (Sener & Savarese, 2018), SimGC (Xiao et al., 2024), GECC-Snapshot (Gong et al., 2026), CGC-X (Gao et al., 2025a), DeepCGC-X (Gao et al., 2026), and DyGC (Chen et al., 2026), covering coreset selection, static graph condensation, and dynamic graph condensation.

Implementation details. The downstream task is node classification on dynamic graphs, evaluated using Micro-F1 and Macro-F1. All node classification experiments use seeds 1–5 and report mean±standard deviation. Data splits and the student training and evaluation protocols follow DyGC. A separate teacher is trained for each seed when required by the method or experimental setting. Test labels are reserved for final reporting. Prototype assignment uses the structure-aware trajectory states $\mathbf{Z}_{1:T}$ on DBLP, Reddit, and Brain; for Arxiv, the raw feature trajectories $\mathbf{X}_{1:T}$ are used only for prototype selection and nearest-anchor assignment to reduce the assignment cost at scale, while trajectory-distribution matching and subsequent optimization remain in the structure-aware state space $\mathbf{Z}$. Dataset-specific preprocessing, prototype representations, optimization hyperparameters, MMD settings, model architectures, stopping rules, and hardware are detailed in **Appendices C and D**.

4.2 Experimental Results

To evaluate TP-DGC, we conduct experiments along five dimensions: (1) node classification; (2) cross-architecture generalization; (3) condensation efficiency; (4) effect of persistent identities; (5) trajectory geometry visualization.

Table 1: Node classification performance (%) under different condensation ratios. Results are mean ± standard deviation over five seeds. Best and second-best condensed methods are bolded and underlined. "Whole" denotes training on the original graph.

Dataset	Metric	Ratio	Random	Herding	K-Center	SimGC	GECC-Snapshot	CGC-X	DeepCGC-X	DyGC	TP-DGC	Whole
DBLP	Micro-F1	0.5%	71.1±1.4	70.4±2.3	66.9±1.8	72.4±0.7	28.6±1.1	65.3±7.1	71.0±1.4	**77.0±0.9**	75.2±0.9	78.9±0.2
		2.5%	74.1±1.2	73.6±0.5	72.2±1.6	71.8±1.4	28.5±4.2	70.4±4.3	72.4±1.2	77.2±0.8	**77.7±0.5**	
		5.0%	76.2±1.0	74.0±0.3	74.3±1.2	69.8±1.1	24.9±4.9	72.3±1.2	70.4±5.5	77.1±1.1	**78.1±0.4**	
	Macro-F1	0.5%	70.2±1.8	70.9±1.3	65.9±1.0	71.3±0.5	14.5±0.6	65.7±7.1	71.5±1.6	**76.3±0.9**	74.2±0.8	78.1±0.2
		2.5%	73.6±0.9	72.5±0.8	70.5±1.9	70.9±1.3	22.5±4.1	70.9±3.0	71.9±1.4	76.9±0.8	**77.2±0.5**	
		5.0%	75.4±1.1	73.1±0.2	73.1±0.9	70.1±0.8	19.0±4.4	72.8±0.6	69.3±7.2	76.7±0.9	**77.6±0.3**	
Reddit	Micro-F1	0.5%	30.9±2.3	32.7±0.8	31.7±1.4	27.4±2.9	28.9±1.2	31.9±1.9	35.1±1.0	**39.0±1.0**	38.4±0.7	49.8±0.6
		2.5%	36.9±1.3	36.1±1.5	35.4±2.3	25.6±1.8	26.0±0.8	29.9±3.2	34.6±3.2	42.3±0.8	**43.2±1.3**	
		5.0%	37.7±1.5	36.9±0.9	38.0±1.6	25.7±1.0	25.2±0.7	37.5±1.9	35.7±3.4	43.3±1.9	**45.4±0.9**	
	Macro-F1	0.5%	30.4±2.3	29.1±2.1	27.6±2.9	21.9±3.2	27.2±1.3	28.4±2.8	32.2±2.6	37.6±2.5	**38.1±0.9**	49.5±0.5
		2.5%	36.5±1.7	34.3±0.7	30.3±5.0	16.9±4.2	22.1±0.7	19.3±5.1	31.9±4.5	41.9±1.0	**42.8±1.4**	
		5.0%	36.9±1.2	35.5±0.7	34.6±4.1	16.8±2.0	19.5±1.3	33.2±4.1	32.7±4.4	42.9±2.6	**45.0±1.2**	
Brain	Micro-F1	1.0%	69.3±2.4	68.7±3.0	62.5±3.2	41.7±4.1	12.8±1.9	22.2±0.8	20.6±1.6	73.8±2.8	**77.7±1.2**	88.5±0.4
		2.5%	79.1±1.3	76.5±0.9	73.5±1.9	35.2±2.9	36.7±8.4	30.6±10.5	27.5±5.7	79.9±1.3	**83.1±1.5**	
		5.0%	80.9±0.4	78.5±0.6	76.4±0.9	38.6±3.4	55.7±7.1	40.1±4.1	49.3±4.1	82.4±1.3	**83.8±0.7**	
	Macro-F1	1.0%	66.2±4.5	67.7±2.8	58.4±4.1	33.9±5.1	2.8±0.8	8.0±0.6	12.4±2.3	73.7±3.5	**77.7±1.5**	88.8±0.3
		2.5%	79.2±1.6	77.1±1.1	74.6±2.1	28.1±2.7	29.2±12.0	20.5±13.4	19.9±8.8	80.0±1.5	**83.6±1.6**	
		5.0%	81.4±0.6	79.2±0.6	77.2±0.8	29.4±4.2	55.6±6.9	37.7±6.7	48.3±4.7	82.9±1.6	**83.6±0.9**	
Arxiv	Micro-F1	0.05%	45.0±2.5	46.9±1.9	40.5±2.7	54.7±2.6	59.5±0.7	59.7±0.7	59.1±0.8	**59.8±0.7**	59.1±1.5	70.1±0.1
		0.25%	51.9±2.0	57.9±0.8	48.9±1.4	57.2±1.6	62.2±0.6	61.6±1.1	62.6±0.8	65.3±0.6	**66.7±0.4**	
		0.5%	56.1±0.7	60.1±1.8	53.6±1.5	57.5±2.5	62.1±0.5	62.1±1.6	62.1±1.0	66.2±0.6	**68.0±0.3**	
	Macro-F1	0.05%	13.4±0.3	19.4±2.1	16.8±1.7	27.3±0.5	31.5±1.2	31.6±1.2	31.3±0.8	31.4±1.1	**31.9±1.4**	48.3±0.1
		0.25%	23.1±0.3	31.7±0.6	22.9±1.3	31.7±0.6	35.9±0.6	35.5±0.8	35.5±1.4	39.9±0.4	**40.2±0.4**	
		0.5%	29.1±0.8	35.3±0.5	25.4±1.1	33.7±1.7	36.7±0.5	36.9±1.1	36.9±0.7	42.3±0.4	**42.4±0.5**	

(1) Node classification results and analysis. Table 1 compares TP-DGC with the baselines across four datasets and multiple condensation ratios. Under the unified protocol, TP-DGC matches or exceeds the reproduced DyGC results in most moderate- and large-budget settings, while DyGC performs better in several extremely low-budget settings. These results demonstrate the effectiveness of trajectory-level condensation for node classification.

TP-DGC generally becomes more stable as the synthetic budget increases. Each synthetic identity must summarize a complete evolution pattern, so an extremely small number of trajectories may not cover the temporal variation within each class. Moderate and large budgets provide greater prototype capacity and represent more diverse within-class evolution patterns. Notably, the evolving variant of GECC (GECC-Evolving) substantially improves over its snapshot-wise counterpart (GECC-Snapshot), showing that the native incremental protocol materially affects its performance. Full results under both protocols are reported in **Appendix E.1**.

(2) Cross-architecture generalization. The generalization ability of TP-DGC is evaluated by transferring the condensed sequences directly to five representative DGNN architectures without re-condensation: TGCN (Zhao et al., 2020), GCRN (Seo et al., 2018), DySAT (Sankar et al., 2020), STGCN (Yu et al., 2018), and ROLAND (You et al., 2022). Table 2 reports the results on Reddit, DBLP, and Arxiv under Micro-F1 and Macro-F1. Across all three datasets and both metrics, TP-DGC achieves the highest average performance, consistently outperforming DeepCGC-X and DyGC. On DBLP, TP-DGC improves over DyGC on every individual architecture for both metrics, demonstrating robust transferability. On Reddit and Arxiv, TP-DGC also attains the best average scores, with only a few architecture-level exceptions where DyGC is stronger. Overall, the consistent average gains across diverse architectures indicate that TP-DGC captures generalizable spatiotemporal semantics, and this transferability is further underpinned by persistent synthetic identities that preserve trajectory-level semantics for different DGNN architectures.

(3) Condensation efficiency. We measure the total condensation time to generate the synthetic sequences, including teacher pretraining when required. As reported in Table 3, TP-DGC runtime remains nearly constant as the condensation ratio increases, whereas DyGC becomes progressively more expensive. Across the three Reddit ratios, TP-DGC achieves 7.1×–56.9× speedups over DyGC in total synthetic-data construction time. This difference is consistent with their parameterizations:

TP-DGC directly optimizes edge-free persistent trajectories and avoids repeatedly optimizing time-varying synthetic graph structure. TP-DGC also improves node-classification performance over CGC-X and DeepCGC-X at additional construction cost, while remaining substantially faster than DyGC. **Appendix E.4** further reports single-seed timings that amortize teacher pretraining across ratios, under which TP-DGC retains its total construction-time advantage.

Table 2: Performance(%) across different DGNN architectures.

Dataset	Metric	Method	TGCN	GCRN	DySAT	STGCN	ROLAND	Average
Reddit r = 5.0%	**Micro-F1**	DeepCGC-X	35.6±2.4	32.1±3.9	30.8±3.8	36.2±2.1	32.1±2.3	33.4±2.4
		DyGC	43.8±0.5	40.4±1.2	33.9±1.4	40.8±1.1	44.2±0.7	40.6±4.1
		TP-DGC	45.9±0.9	40.4±1.0	34.7±0.6	40.4±0.7	45.4±0.6	**41.4±4.6**
	Macro-F1	DeepCGC-X	32.4±2.6	25.4±6.1	21.9±8.1	34.3±4.0	29.2±4.3	28.6±5.1
		DyGC	43.4±0.9	40.0±1.3	32.3±2.6	40.7±0.9	44.2±0.7	40.1±4.7
		TP-DGC	45.1±1.3	40.0±1.3	34.4±0.6	40.1±0.8	45.1±0.5	**40.9±4.4**
DBLP r = 5.0%	**Micro-F1**	DeepCGC-X	69.6±7.4	62.6±7.0	73.9±1.3	59.4±4.3	75.1±0.8	68.1±6.9
		DyGC	77.6±0.5	76.2±0.6	76.8±0.3	74.4±0.4	78.3±0.4	76.7±1.5
		TP-DGC	78.2±0.5	77.3±0.7	77.4±0.4	75.1±0.5	78.5±0.4	**77.3±1.3**
	Macro-F1	DeepCGC-X	67.9±9.6	61.6±8.0	73.7±1.3	61.4±4.6	74.4±1.1	67.8±6.3
		DyGC	77.1±1.0	75.9±0.7	76.1±0.3	74.0±0.6	77.8±0.4	76.2±1.5
		TP-DGC	77.8±0.6	76.8±0.6	76.9±0.4	74.5±0.5	78.3±0.3	**76.8±1.5**
Arxiv r = 0.5%	**Micro-F1**	DeepCGC-X	61.4±0.9	54.9±0.5	61.2±0.5	40.5±1.0	59.3±0.5	55.5±8.7
		DyGC	66.6±0.4	60.7±1.8	65.6±0.4	49.1±1.6	62.8±0.2	61.0±7.0
		TP-DGC	68.3±0.2	62.5±0.2	66.4±0.4	47.3±1.1	63.1±0.2	**61.5±8.3**
	Macro-F1	DeepCGC-X	36.3±1.0	35.7±0.4	39.4±0.8	25.2±0.4	33.4±0.8	34.0±5.3
		DyGC	41.8±0.5	36.8±0.5	42.4±0.2	24.6±0.4	39.8±0.3	37.1±7.3
		TP-DGC	42.2±0.4	37.5±0.5	42.6±0.6	25.8±0.1	38.2±1.3	**37.3±6.8**

Table 3: Total synthetic-data construction time (s) on Reddit.

Ratio	SimGC	GECC-Snapshot	CGC-X	DeepCGC-X	DyGC	TP-DGC	Speedup vs. DyGC
0.5%	122.2±3.3	39.7±2.5	2.6±0.1	2.6±0.2	368.6±1.7	51.9±0.4	**7.1**×
2.5%	208.7±0.2	261.3±2.0	4.0±0.1	2.7±0.0	616.3±88.3	53.1±0.5	**11.6**×
5.0%	556.7±21.8	995.5±72.1	5.6±0.0	3.2±0.1	3122.4±438.4	54.9±1.2	**56.9**×

(4) Effect of persistent identities. To test whether cross-snapshot correspondence carries task-relevant information, we design two class-wise controls. Common permutation applies a fixed within-class permutation across all snapshots, only renaming complete trajectories, whereas time-varying permutation resamples it at each non-initial snapshot, letting one index represent different identities over time. Both are applied synchronously to features, labels, and adjacency, preserving the per-snapshot budget, class-conditional feature multiset, labels, and graph structure, and differing only in whether identity correspondence remains consistent. As shown in Table 4, common permutation changes either mean F1 by at most 0.6 points, while time-varying permutation reduces Micro-F1 by 1.8–3.8 points and Macro-F1 by 2.1–5.8 points. This indicates that preserving each snapshot alone is insufficient when identity continuity is disrupted, consistent with Proposition 1 in Appendix B.1, and the same pattern holds across five DGNN architectures (**Appendix B.2**).

Table 4: Cross-snapshot identity controls.

Dataset	Condition	Micro-F1	Macro-F1	Δ Micro-F1	Δ Macro-F1
DBLP 2.5%	Original	77.7±0.5	77.2±0.5	—	—
	Common permutation	**77.8±0.4**	**77.3±0.5**	+0.1	+0.1
	Time-varying permutation	75.5±0.6	75.1±0.6	−2.2	−2.1
Reddit 2.5%	Original	**43.2±1.3**	**42.8±1.4**	—	—
	Common permutation	42.7±0.8	42.2±1.0	−0.5	−0.6
	Time-varying permutation	39.7±0.5	37.7±1.0	−3.5	−5.1
Brain 2.5%	Original	83.1±1.5	**83.6±1.6**	—	—
	Common permutation	**83.5±0.7**	83.4±0.8	+0.4	−0.2
	Time-varying permutation	81.3±1.5	81.2±2.1	−1.8	−2.4
Arxiv 0.25%	Original	**66.7±0.4**	**40.2±0.4**	—	—
	Common permutation	66.5±0.6	39.6±0.4	−0.2	−0.6
	Time-varying permutation	62.9±1.3	34.4±1.1	−3.8	−5.8

(5) Trajectory geometry visualization. Fig. 3 examines whether the condensed sequences preserve the trajectory geometry of the full graph on DBLP at $r = 5.0\%$. Across all four statistics, TP-DGC remains closer to the full graph than DyGC, reducing excessive trajectory movement and temporal variation while better preserving directional consistency. This indicates that persistent iden-

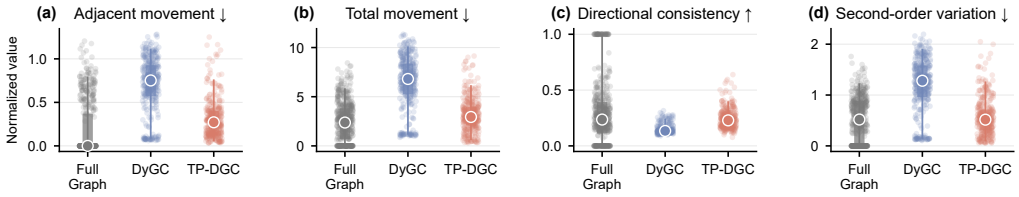

Figure 3: Native-space trajectory statistics on DBLP ($r = 5.0\%$).

tities yield smoother and more coherent temporal evolution, complementing the node-classification and permutation evidence. In additional analyses, Appendix F.1 reports an additional experiment on a cumulative ogbn-arxiv sequence constructed from publication years, where at the best tested budget TP-DGC retains 96.7% of the full-graph Micro-F1 and 93.5% of its Macro-F1; this experiment covers cumulative arrivals but does not consider deletion or re-entry.

4.3 Ablation Study and Sensitivity Analysis

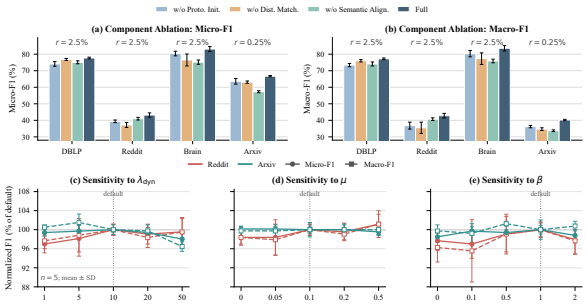

Figure 4: Ablation study and sensitivity analysis.

Ablation study. To investigate the contributions of trajectory-prototype initialization, complete-trajectory distribution matching, and Task-Specific Semantic Alignment, we conduct an ablation study on DBLP, Reddit, Brain, and Arxiv, as shown in Fig. 4(a–b). The ablated variants include *w/o Proto. Init.*, which replaces trajectory-prototype initialization; *w/o Dist. Match.*, which removes complete-trajectory distribution matching; and *w/o Semantic Align.*, which removes Task-Specific Semantic Alignment. Micro-F1 and Macro-F1 are adopted as the evaluation metrics.

As observed, the complete TP-DGC consistently outperforms all ablated variants across the four datasets. Replacing trajectory-prototype initialization causes consistent F1 degradation (2.7–6.0 points), confirming its importance for representative trajectory initialization. Distribution matching is most influential on Reddit and Brain (maximum drops of 7.3 and 6.6 points), whereas semantic alignment contributes most strongly on Brain and Arxiv (maximum drops of 7.9 and 9.4 points). These results confirm the complementary contributions of the three mechanisms. Detailed results are reported in **Appendix E.2**.

Sensitivity analysis of hyperparameters λ_{dyn}, μ, and β. Fig. 4(c–e) varies λ_{dyn}, μ, and β independently around their default values of 10, 0.1, and 1. Overall, TP-DGC exhibits stable performance across both Reddit and Arxiv, with small standard deviations and only minor fluctuations. On Reddit, the default $\lambda_{\text{dyn}} = 10$ and $\beta = 1$ achieve the best results, while $\mu = 0.5$ slightly improves over the default $\mu = 0.1$. On Arxiv, performance is largely insensitive to all three parameters, and the default settings remain competitive. Detailed numerical results are reported in **Appendix E.3**.

5 Conclusion

We propose TP-DGC, a trajectory-level framework that makes cross-snapshot correspondence intrinsic to dynamic graph condensation. By treating complete node trajectories as condensation units, TP-DGC produces compact edge-free sequences with persistent synthetic identities that can be reused across DGNN architectures. Experiments on four datasets show competitive node-classification performance and support reuse across five DGNN architectures. Controlled permutations further show that cross-snapshot correspondence carries predictive information. **Limitations and Future Work.** Our current study focuses on the widely adopted discrete-time setting with a shared node universe or cumulative arrivals. Extending TP-DGC to open-world dynamics, including node deletion, re-entry, and irregular observations, is a promising direction for future work.

AI USE STATEMENT

Generative AI tools were used only for language polishing, including improvements to grammar, wording, and readability. All scientific ideas, methods, experiments, analyses, and conclusions were developed and verified by the authors. The authors reviewed all AI-assisted edits and take full responsibility for the final content of this work.

REFERENCES

David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1027–1035, 2007.

George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A. Efros, and Jun-Yan Zhu. Dataset distillation by matching training trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10708–10717, 2022.

Dong Chen, Shuai Zheng, Yeyu Yan, Muhao Xu, Zhenfeng Zhu, and Yao Zhao. Evolution-consistent dynamic graph condensation. *Pattern Recognition*, 180:114169, 2026.

Jiawei Du, Yidi Jiang, Vincent Y. F. Tan, Joey Tianyi Zhou, and Haizhou Li. Minimizing the accumulated trajectory error to improve dataset distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3749–3758, 2023.

Junfeng Fang, Xinglin Li, Yongduo Sui, Yuan Gao, Guibin Zhang, Kun Wang, Xiang Wang, and Xiangnan He. EXGC: Bridging efficiency and explainability in graph condensation. In *Proceedings of the ACM on Web Conference 2024*, pp. 721–732, 2024.

Xinyi Gao, Tong Chen, Wentao Zhang, Yayong Li, Xiangguo Sun, and Hongzhi Yin. Graph condensation for open-world graph learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 851–862, 2024.

Xinyi Gao, Guanhua Ye, Tong Chen, Wentao Zhang, Junliang Yu, and Hongzhi Yin. Rethinking and accelerating graph condensation: A training-free approach with class partition. In *Proceedings of the ACM on Web Conference 2025*, pp. 4359–4373, 2025a.

Xinyi Gao, Junliang Yu, Tong Chen, Guanhua Ye, Wentao Zhang, and Hongzhi Yin. Graph condensation: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 37(4):1819–1837, 2025b.

Xinyi Gao, Wenjie Li, Tong Chen, Xiangyu Zhao, Quoc Viet Hung Nguyen, and Hongzhi Yin. DeepCGC: Unveiling the deep clustering mechanism of fast graph condensation. *IEEE Transactions on Knowledge and Data Engineering*, 38(3):1575–1588, 2026.

Shengbo Gong, Mohammad Hashemi, Juntong Ni, Carl Yang, and Wei Jin. Scalable graph condensation with evolving capabilities. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 314–323, 2026.

Mohammad Hashemi, Shengbo Gong, Juntong Ni, Wenqi Fan, B. Aditya Prakash, and Wei Jin. A comprehensive survey on graph reduction: Sparsification, coarsening, and condensation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 8058–8066, 2024.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems*, volume 33, pp. 22118–22133, 2020.

Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael M. Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. Temporal graph benchmark for machine learning on temporal graphs. In *Advances in Neural Information Processing Systems*, volume 36, pp. 2056–2073, 2023.

Wei Jin, Xianfeng Tang, Haoming Jiang, Zheng Li, Danqing Zhang, Jiliang Tang, and Bing Yin. Condensing graphs via one-step gradient matching. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 720–730, 2022a.

Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. Graph condensation for graph neural networks. In *International Conference on Learning Representations*, 2022b.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Mengyang Liu, Shanchuan Li, Xinshi Chen, and Le Song. Graph condensation via receptive field distribution matching. *arXiv preprint arXiv:2206.13697*, 2022.

Yang Liu, Deyu Bo, and Chuan Shi. Graph distillation with eigenbasis matching. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 30702–30717. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24d.html.

Antonio Longa, Veronica Lachi, Gabriele Santin, Monica Bianchini, Bruno Lepri, Pietro Liò, Franco Scarselli, and Andrea Passerini. Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=pHCdMat0gI.

James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset meta-learning from kernel ridge-regression. In *International Conference on Learning Representations*, 2021.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34 (4):5363–5370, 2020.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. DySAT: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pp. 519–527, 2020.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *Neural Information Processing*, pp. 362–373, 2018.

Junwei Su, Difan Zou, and Chuan Wu. PRES: Toward scalable memory-based dynamic graph neural networks. In *International Conference on Learning Representations*, 2024.

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A. Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

Max Welling. Herding dynamical weights to learn. In *Proceedings of the 26th International Conference on Machine Learning*, pp. 1121–1128, 2009.

Zhenbang Xiao, Yu Wang, Shunyu Liu, Huiqiong Wang, Mingli Song, and Tongya Zheng. Simple graph condensation. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pp. 53–71, 2024.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations*, 2020.

Zhe Xu, Yuzhong Chen, Menghai Pan, Huiyuan Chen, Mahashweta Das, Hao Yang, and Hanghang Tong. Kernel ridge regression-based graph dataset distillation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2850–2861, 2023.

Beining Yang, Kai Wang, Qingyun Sun, Cheng Ji, Xingcheng Fu, Hao Tang, Yang You, and Jianxin Li. Does graph distillation see like vision dataset counterpart? In *Advances in Neural Information Processing Systems*, volume 36, pp. 53201–53226, 2023.

Jiaxuan You, Tianyu Du, and Jure Leskovec. ROLAND: Graph learning framework for dynamic graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2358–2366, 2022.

Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pp. 3634–3640, 2018.

Guibin Zhang, Yiyan Qi, Ziyang Cheng, Yanwei Yue, Dawei Cheng, and Jian Guo. Rationalizing and augmenting dynamic graph neural networks. In *International Conference on Learning Representations*, 2025.

Yuchen Zhang, Tianle Zhang, Kai Wang, Ziyao Guo, Yuxuan Liang, Xavier Bresson, Wei Jin, and Yang You. Navigating complexity: Toward lossless graph condensation via expanding window matching. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 60379–60395. PMLR, 2024. URL `https://proceedings.mlr.press/v235/zhang24cg.html`.

Bo Zhao and Hakan Bilen. Dataset condensation with distribution matching. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 6503–6512, 2023.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. In *International Conference on Learning Representations*, 2021.

Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-GCN: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3848–3858, 2020.

Xin Zheng, Miao Zhang, Chunyang Chen, Quoc Viet Hung Nguyen, Xingquan Zhu, and Shirui Pan. Structure-free graph condensation: From large-scale graphs to condensed graph-free data. In *Advances in Neural Information Processing Systems*, volume 36, pp. 6026–6047, 2023.

Yongjian Zhong, Hieu Vu, Tianbao Yang, and Bijaya Adhikari. Efficient and effective implicit dynamic graph neural network. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4595–4606, 2024.

A Related Work

A.1 Static Graph Condensation

Dataset condensation synthesizes compact training sets through bilevel optimization, gradient matching, distribution matching, or kernel regression (Wang et al., 2018; Zhao et al., 2021; Zhao & Bilen, 2023; Nguyen et al., 2021). Graph condensation extends this idea to relational data in both node- and graph-level settings. GCond condenses a single large graph for node classification (Jin et al., 2022b), whereas DosCond primarily condenses graph datasets for graph classification and also evaluates a node-classification extension (Jin et al., 2022a). Subsequent work has explored structure-aware objectives, structure-free representations, training-trajectory matching, spectral supervision, and efficient construction (Yang et al., 2023; Zheng et al., 2023; Zhang et al., 2024; Liu et al., 2024; Xu et al., 2023; Xiao et al., 2024; Fang et al., 2024; Gao et al., 2025a). Surveys provide broader taxonomies of graph reduction and condensation (Hashemi et al., 2024; Gao et al., 2025b). Static condensers optimize individual graphs and therefore do not model cross-snapshot correspondence during condensation.

A.2 Dynamic Graph Learning and Condensation

Dynamic graph neural networks jointly model graph structure and temporal dependencies through recurrent updates, temporal convolutions, or temporal attention (Zhao et al., 2020; Seo et al., 2018; Yu et al., 2018; Sankar et al., 2020); continuous-time models instead operate on timestamped interaction events (Xu et al., 2020; Rossi et al., 2020). Temporal graph benchmarks further emphasize that model behavior and evaluation protocols vary substantially across dynamic settings (Huang et al., 2023). DyGC extends joint spatiotemporal modeling to dynamic graph condensation by optimizing time-varying synthetic node features and graph structures using a state-evolution field and spatiotemporal state alignment (Chen et al., 2026). Related evolving-graph settings include OpenGC, which targets generalization under future graph-distribution shifts, and GECC, which incrementally updates a condensed graph as new data batches arrive (Gao et al., 2024; Gong et al., 2026).

A.3 Trajectory Matching

Trajectory information has also been used in dataset condensation. Training-trajectory matching constructs synthetic samples by matching model-parameter updates over multiple optimization steps (Cazenavette et al., 2022; Du et al., 2023); GEOM adapts this optimization-trajectory view to graph condensation (Zhang et al., 2024). In TP-DGC, a trajectory instead denotes the time-ordered states of the same graph node over the observation window. TP-DGC therefore targets node-state evolution and cross-snapshot identity correspondence, differing from training-trajectory matching in both its time axis and condensation object.

B Formal Analysis of Cross-Snapshot Identity

B.1 Cross-Snapshot Permutation Ambiguity

Proposition 1 (Permutation compatibility of temporal model classes). Let $S_t = \phi(A_t, X_t)$ be a permutation-equivariant spatial encoder, and let Π_t be a within-class permutation of node indices at snapshot t. Any condensation objective that depends only on the unordered node set of each snapshot is invariant to independently chosen $\{\Pi_t\}_{t=1}^{T}$. In contrast, a node-indexed temporal model is equivariant to a common permutation $\Pi_1 = \cdots = \Pi_T = \Pi$, but is generally not equivariant to time-varying permutations. This distinction applies to the following model classes:

1. *Recurrent temporal models*, with $H_t = R(S_t, H_{t-1})$, including graph-recurrent and hierarchical recurrent updates;
2. *Temporal convolution models*, with $H_t = \sum_{\ell=0}^{L-1} S_{t-\ell} W_\ell$ or its gated nonlinear extension;
3. *Temporal self-attention models*, whose attention for index i compares query and key states associated with that same index at different snapshots.

Proof. A snapshot-set objective is a sum of permutation-invariant functions. Jointly replacing (A_t, X_t) by $(\Pi_t A_t \Pi_t^\top, \Pi_t X_t)$ therefore preserves every snapshot term, irrespective of whether the permutations agree across time. Because ϕ is equivariant, the resulting spatial state is $\widetilde{S}_t = \Pi_t S_t$.

For a recurrent model, row-wise equivariance of R gives $R(\Pi S_t, \Pi H_{t-1}) = \Pi R(S_t, H_{t-1})$ under one common permutation. With time-varying permutations, however, the update instead receives $(\Pi_t S_t, \Pi_{t-1} H_{t-1})$. Factoring out Π_t would require $\Pi_{t-1} H_{t-1} = \Pi_t H_{t-1}$, which does not hold in general when $\Pi_t \neq \Pi_{t-1}$.

For a temporal convolution:

$$\widetilde{H}_t = \sum_{\ell=0}^{L-1} \Pi_{t-\ell} S_{t-\ell} W_\ell, \tag{22}$$

whereas equivariance at time t would require $\widetilde{H}_t = \Pi_t \sum_\ell S_{t-\ell} W_\ell$. The two expressions are generally unequal if any active lag uses $\Pi_{t-\ell} \neq \Pi_t$.

For temporal self-attention, a common permutation merely reorders complete index-wise query–key sequences. Under time-varying permutations, the query at index i and time t is paired with keys that can originate from different synthetic identities at earlier times. The attention scores and values thus change in general. Consequently, independent snapshot permutations preserve snapshot-wise content but not the index-aligned temporal computation, except in degenerate cases with no effective cross-time coupling or indistinguishable node states. □

Corollary 1. A common class-wise permutation is only a relabeling of complete synthetic trajectories, whereas independently changing the permutation across snapshots can splice states from different identities and alter the predictions of recurrent, convolutional, and attention-based DGNNs. Proposition 1 characterizes permutation compatibility. The controlled studies evaluate its predictive implications on the tested datasets and across the covered model classes.

B.2 Architecture-Wise Permutation Controls

Proposition 1 predicts that the common-permutation control should behave as a consistent relabeling, whereas time-varying permutations should affect DGNNs that couple node-indexed states across time. We test this prediction on Reddit at $r = 2.5\%$ using five representative downstream architectures and the same five-seed protocol as the main permutation study. As reported in Table 5, common permutation remains close to the original condition for every architecture. Time-varying permutation lowers both metrics for all five architectures, with the largest degradation for GCRN and smaller but directionally consistent changes for STGCN and DySAT. These results support the model-class analysis across recurrent, convolutional, attention-based, and hierarchical temporal mechanisms, despite differences in effect size across architectures.

Table 5: Architecture-wise permutation controls on Reddit at $r = 2.5\%$ (Micro-F1/Macro-F1, %, mean ± standard deviation over five seeds). This table reports an independent five-seed rerun. Common permutation applies one class-wise permutation consistently across all snapshots; time-varying permutation independently permutes synthetic identities at each non-initial snapshot. Each condition reuses the same condensed sequence and paired student seeds.

Architecture	Original	Common permutation	Time-varying permutation
TGCN	43.3±1.7 / 42.8±1.8	42.2±1.4 / 41.7±1.6	40.4±0.6 / 39.3±1.0
GCRN	41.0±2.2 / 40.6±1.8	40.4±2.0 / 40.0±1.7	33.7±1.3 / 33.0±2.0
DySAT	34.8±0.3 / 34.6±0.5	34.7±1.3 / 34.4±1.4	34.0±1.2 / 33.6±1.2
STGCN	39.3±1.6 / 39.1±1.9	39.9±1.4 / 39.6±1.3	38.0±0.5 / 37.8±0.6
ROLAND	43.5±1.1 / 42.6±1.8	43.2±1.2 / 42.4±1.6	39.5±1.0 / 37.5±3.5

C Method and Optimization Details

C.1 TP-DGC Procedure

Algorithm 1 TP-DGC: Trajectory-Persistent Dynamic Graph Condensation

Input: Training graph sequence $\{(\mathbf{X}_t, \mathbf{A}_t)\}_{t=1}^{T}$, training labels, budget M, calibration steps S.
Output: Edge-free condensed sequence $\mathcal{G}' = \{(\mathbf{X}'_t, \mathbf{I}_M, \mathbf{Y}')\}_{t=1}^{T}$.
Pretrain or load the DGNN teacher f_{θ_T}; freeze θ_T and compute real class-level semantic targets.
Construct $\mathbf{Z}_{1:T} = \Phi(\mathbf{X}_{1:T}, \mathbf{A}_{1:T})$ and complete real trajectories.
Set $\mathbf{u}_i = \mathrm{Normalize}(\mathrm{vec}(\mathbf{R}_{1:T,i}))$ for training nodes, where $\mathbf{R} = \mathbf{Z}$ for DBLP, Reddit, and Brain, and $\mathbf{R} = \mathbf{X}$ for Arxiv.
for each class c **do**
    Select M_c trajectory anchors by k-means++ and assign each training node to its nearest same-class anchor.
end for
Form the time-shared assignment $\mathbf{P} \in \{0,1\}^{N_{\mathrm{tr}} \times M}$ and synthetic labels $\mathbf{Y}'$.
for $t = 1, \ldots, T$ **do**
    $\mathbf{x}'^{(0)}_{t,j} = \dfrac{\sum_{i \in \mathcal{I}_{\mathrm{tr}}} P_{ij} \mathbf{x}_{t,i}}{\sum_{i \in \mathcal{I}_{\mathrm{tr}}} P_{ij} + \varepsilon}, \quad j = 1, \ldots, M.$
end for
for $s = 1, \ldots, S$ **do**
    $\mathbf{Z}'_{1:T} = \Phi(\mathbf{X}'_{1:T}, \mathbf{I}_M)$; evaluate $\mathcal{L}_{\mathrm{dyn}}$, $\mathcal{L}_{\mathrm{sem}}$, and $\mathcal{R}_{\mathrm{proto}}$.
    $\mathcal{L} = \lambda_{\mathrm{dyn}}\mathcal{L}_{\mathrm{dyn}} + \lambda_{\mathrm{sem}}\mathcal{L}_{\mathrm{sem}} + \mu\mathcal{R}_{\mathrm{proto}}$.
    Update the complete $\mathbf{X}'_{1:T}$ by Adam on $\mathcal{L}$; keep $\mathbf{P}$ and θ_T fixed.
end for
Apply the run-specific stopping and checkpoint-selection protocol.
return $\mathcal{G}'$ using the selected $\mathbf{X}'_{1:T}$.

Assignment representation. DBLP, Reddit, and Brain use normalized structure-aware state trajectories; the large-graph Arxiv implementation uses normalized raw feature trajectories with chunked cosine matching. Both compute one assignment over the complete window and share it across snapshots, while all datasets use structure-aware states in the trajectory-distribution objective.

C.2 Objective and Complexity

For class c, trajectory preservation uses the biased squared empirical MMD:

$$\widehat{\mathrm{MMD}}_c^2 = \frac{1}{n_c^2}\sum_{i,i'} k(\tau_i, \tau_{i'}) - \frac{2}{n_c m_c}\sum_{i,j} k(\tau_i, \tau'_j) + \frac{1}{m_c^2}\sum_{j,j'} k(\tau'_j, \tau'_{j'}), \tag{23}$$

$$k(u,v) = \sum_{q=-2}^{2} \exp\left(-\frac{\|u-v\|_2^2}{2^q\sigma^2}\right), \tag{24}$$

where σ^2 is the mean off-diagonal squared distance of pooled real and synthetic trajectories for DBLP, Reddit, and Brain. For Arxiv, it is precomputed from real training trajectories and kept fixed during calibration, avoiding repeated large-scale bandwidth estimation. The Arxiv solver omits the real–real kernel term, which is constant with respect to the synthetic features, and optimizes the remaining gradient-bearing terms. With frozen-teacher probabilities p_T, hard supervision is $\mathcal{L}_{\mathrm{hard}} = -\mathbb{E}_j \log p_T(y'_j \mid \tau'_j)$ and the soft term is $\mathcal{L}_{\mathrm{soft}} = \tau^2 C^{-1} \sum_c \mathrm{KL}(q_c^{\mathrm{r}} \| q_c^{\mathrm{s}})$.

For Arxiv, prototype preservation regularization uses the scale-normalized feature-space regularizer:

$$\mathcal{R}_{\mathrm{proto}}^{\mathrm{Arxiv}} = \frac{\|\mathbf{X}' - \mathbf{X}'^{(0)}\|_F^2/(MTD)}{\|\mathbf{X}'^{(0)}\|_F^2/(MTD) + \varepsilon}. \tag{25}$$

This replaces the propagated-state regularizer used for DBLP, Reddit, and Brain while retaining the same prototype-preservation role.

With N_{tr} real trajectories, M synthetic trajectories, propagation depth K, and S calibration steps, time complexity is:

$$O\left(K\sum_{t=1}^{T}|E_t|d + N_{\text{tr}}MTd + SC_{\text{cal}}\right), \tag{26}$$

where C_{cal} denotes the cost of one calibration step, including trajectory-distribution evaluation and a frozen-teacher forward pass on the compact synthetic sequence. The hard assignment is stored as one prototype index per real trajectory, requiring $O(N_{\text{tr}})$ persistent memory; only an active $B \times M$ distance block is materialized. Peak state-and-score storage is $O(NTd_z + MTd_z + BM)$, where $d_z = (K+1)d$. The one-time assignment dominates before calibration, while later optimization is restricted to the compact synthetic sequence and does not parameterize a time-varying edge generator. This profile is consistent with Table 3.

D Reproducibility and Experimental Setup

D.1 Dataset Construction and Preprocessing

Table 6: Dataset statistics and implementation settings. #Edges counts stored edge entries across all snapshots. Teacher-related columns report layers, hidden dimension, maximum steps, and early-stopping patience; Student Steps reports downstream training steps.

Dataset	#Snapshots	#Nodes	#Edges	#Classes	#Features	Train/Val/Test	Teacher	L/H	K	Max/ES Pat./Student Steps	Task Type
DBLP	10	6,606	42,815	5	100	3,303/1,321/1,982	TGCN	2/128	2	1,000/150/1,000	Transductive NC
Reddit	10	8,291	264,050	4	20	4,145/1,658/2,488	TGCN	1/128	2	1,000/150/1,000	Transductive NC
Brain	12	5,000	1,955,488	10	20	2,500/1,000/1,500	TGCN	1/128	2	1,000/150/1,000	Transductive NC
Arxiv	6	169,343	4,113,046	40	128	90,941/29,799/48,603	TGCN-L	3/256	3	100/100/2,000	Transductive NC

DBLP, Reddit, and Brain use fixed 50/20/30 transductive node splits; Arxiv uses the official OGB split (Hu et al., 2020). Reddit and DBLP are processed from the released DyGC raw files into ten snapshots. Brain is obtained from the Brain10 data released with IDGNN (Zhong et al., 2024) and contains twelve snapshots, while Arxiv is organized into six snapshots under the released large-graph protocol. Arxiv is downloaded through OGB and converted to the temporal format used by the official evaluation pipeline. Preprocessing creates processed snapshots and fixed split files without modifying the source data.

We evaluate persistent synthetic identities under a fixed node universe in the main benchmark and under cumulative timestamp-based node arrivals in Section F.1. The edge count in Table 6 sums stored entries over snapshots; a recurring edge is counted once per timestamp. Model-side normalization adds self-loops rather than storing them as dataset edges. Synthetic budgets are fractions of the full node set for DBLP, Reddit, and Brain. Arxiv follows the class-wise large-graph allocation and rounds upward when needed to retain rare classes.

D.2 Baseline Adaptation and Hyperparameters

Static condensers are applied independently to each training snapshot, $\widetilde{G}_t = \mathcal{M}(G_t; r)$, without changing their original objectives. Because independently generated indices need not denote the same within-class mode over time, each current synthetic snapshot is aligned class-wise to the preceding already aligned snapshot. We set $(\widehat{\mathbf{X}}_1, \widehat{\mathbf{A}}_1, \widehat{\mathbf{y}}_1) = (\widetilde{\mathbf{X}}_1, \widetilde{\mathbf{A}}_1, \widetilde{\mathbf{y}}_1)$ and define $\mathbf{Q}_t(i,j) = 1$ to mean that aligned position i takes row j of the current unaligned snapshot. For $t \geq 2$:

$$\mathbf{Q}_t^* = \arg\min_{\mathbf{Q}\in\mathcal{P}_{\text{class}}} \sum_{i,j} Q_{ij}\, d\big(\widehat{\mathbf{x}}_{t-1,i}, \widetilde{\mathbf{x}}_{t,j}\big), \tag{27}$$

where $d(\mathbf{x},\mathbf{y}) = 1 - \mathbf{x}^\top\mathbf{y}/(\|\mathbf{x}\|_2\|\mathbf{y}\|_2)$ is the cosine distance used by the implementation and $\mathcal{P}_{\text{class}}$ restricts assignments to the same class. Features, adjacency, and labels are synchronously reindexed as:

$$\widehat{\mathbf{X}}_t = \mathbf{Q}_t^*\widetilde{\mathbf{X}}_t, \qquad \widehat{\mathbf{A}}_t = \mathbf{Q}_t^*\widetilde{\mathbf{A}}_t(\mathbf{Q}_t^*)^\top, \qquad \widehat{\mathbf{y}}_t = \mathbf{Q}_t^*\widetilde{\mathbf{y}}_t. \tag{28}$$

Only training information is used for this matching. Sequential Hungarian matching minimizes adjacent-snapshot feature costs within each class. Its objective contains no non-adjacent or higher-order temporal terms.

This protocol is used for GECC-Snapshot, SimGC, CGC-X, DeepCGC-X, Random, Herding, and K-Center. GECC-Snapshot is run independently per snapshot so that adapted static methods receive the same input scope. TP-DGC adopts k-means++ initialization following GECC and computes one class-constrained trajectory assignment shared across snapshots.

The main setting weights trajectory distribution matching by $\lambda_{\text{dyn}} = 10$, semantic alignment by $\lambda_{\text{sem}} = 1$, and prototype regularization by $\mu = 0.1$. Within $\mathcal{L}_{\text{sem}}$, the soft-target coefficient is $\beta = 1$ and the temperature is $\tau = 2$. We set the temporal history coefficient α to 0.1 for DBLP, Reddit, and Brain, and to 0.5 for Arxiv; teacher hidden-state loss is disabled. Adam (Kingma & Ba, 2015) uses learning rate 0.005. The large-graph solver uses assignment chunks of $B = 2{,}048$, and Arxiv evaluation uses the official 100K batch.

D.3 Training and Evaluation Protocol

Main performance and component ablations use seeds 1–5 and report mean ± standard deviation under the official DyGC student-test protocol (Chen et al., 2026). The frozen teacher follows the official TGCN/TGCN-L training and validation-checkpoint protocol. All condensation, alignment, stopping, and checkpoint-selection procedures exclude test labels. Validation labels are used only for candidate selection, and test labels are reserved for final reporting. Every TP-DGC artifact has zero off-diagonal synthetic adjacency entries. Official GCN normalization adds self-loops, giving $\widehat{\mathbf{A}}'_t = \mathbf{I}$; spatial layers reduce to node-wise transformations, and the temporal module processes each synthetic identity across snapshots. Real-graph propagation supplies structure-aware reference trajectories for trajectory-distribution matching and Task-Specific Semantic Alignment. Experiments ran on one NVIDIA RTX 4060 Laptop GPU (8 GB), an Intel i9-14900HX CPU, and 32 GB RAM using Python 3.12.7, PyTorch 2.5.1, and CUDA 12.1.

The certificate is evaluated every five steps, and adaptive stopping becomes eligible from step 20 onward. A closed-form ridge probe fitted on synthetic representations evaluates a class-balanced training subset and forms the train-only certificate $\frac{1}{2}(\text{MicroF1} + \text{MacroF1})$. It affects stopping, not the condensation gradient. Calibration stops when objective saturation, small feature updates, and certificate stagnation coincide, subject to a 200-step maximum. After calibration, the small-graph implementation selects among periodically saved candidates using a combined score over complete trajectories, final states, temporal differences, and teacher logits.

E Additional Quantitative Results

E.1 GECC-Snapshot and GECC-Evolving

Table 7: GECC adaptation study (Micro-F1/Macro-F1, %, mean ± standard deviation over five seeds). GECC-Snapshot follows the common static-baseline protocol; GECC-Evolving uses incremental graph accumulation. The best Micro-F1 and Macro-F1 in each row are bolded.

Dataset	Ratio	GECC-Snapshot	GECC-Evolving	TP-DGC
DBLP	0.5%	28.6±1.1 / 14.5±0.6	72.7±1.2 / 73.2±0.9	**75.2±0.9 / 74.2±0.8**
	2.5%	28.5±4.2 / 22.5±4.1	70.2±0.8 / 71.0±0.7	**77.7±0.5 / 77.2±0.5**
	5.0%	24.9±4.9 / 19.0±4.4	75.8±0.7 / 75.7±0.8	**78.1±0.4 / 77.6±0.3**
Reddit	0.5%	28.9±1.2 / 27.2±1.3	30.5±1.5 / 28.0±1.7	**38.4±0.7 / 38.1±0.9**
	2.5%	26.0±0.8 / 22.1±0.7	34.4±2.2 / 33.1±2.7	**43.2±1.3 / 42.8±1.4**
	5.0%	25.2±0.7 / 19.5±1.3	33.7±0.3 / 28.8±1.1	**45.4±0.9 / 45.0±1.2**
Brain	1.0%	12.8±1.9 / 2.8±0.8	16.6±2.8 / 8.6±2.1	**77.7±1.2 / 77.7±1.5**
	2.5%	36.7±8.4 / 29.2±12.0	24.5±4.2 / 16.7±4.6	**83.1±1.5 / 83.6±1.6**
	5.0%	55.7±7.1 / 55.6±6.9	39.7±2.2 / 33.7±3.5	**83.8±0.7 / 83.6±0.9**
Arxiv	0.05%	**59.5±0.7** / 31.5±1.2	59.4±1.5 / 31.7±1.2	59.1±1.5 / **31.9±1.4**
	0.25%	62.2±0.6 / 35.9±0.6	59.8±0.9 / 33.1±0.8	**66.7±0.4 / 40.2±0.4**
	0.5%	62.1±0.5 / 36.7±0.5	62.5±1.1 / 35.7±0.8	**68.0±0.3 / 42.4±0.5**

Evaluation protocol for static condensers. To evaluate static condensers under a unified protocol, we condense each snapshot independently and use the Hungarian algorithm (Munkres, 1957) to establish class-wise synthetic-node correspondence between adjacent snapshots. CGC-X and

DeepCGC-X are the official graphless variants of CGC and DeepCGC, respectively. Under our dynamic evaluation protocol, they are executed per snapshot and aligned post hoc. The main results use GECC-Snapshot so that all adapted static methods receive the same per-snapshot input. GECC-Evolving, which operates on cumulative graphs, is reported separately below.

GECC-Evolving substantially outperforms GECC-Snapshot on DBLP and Reddit, showing that GECC benefits from its native cumulative-input protocol. TP-DGC remains stronger at the moderate and large budgets, while Arxiv at the smallest budget is comparable across the three protocols. Brain remains challenging for both GECC protocols.

E.2 Component Ablations

Table 8: Ablation study of TP-DGC. Results are Micro-F1/Macro-F1 (%), reported as mean ± standard deviation over five seeds. The full configuration is bolded and shaded.

Variant	DBLP $r = 2.5\%$	Reddit $r = 2.5\%$	Brain $r = 2.5\%$	Arxiv $r = 0.25\%$
w/o trajectory prototype init.	74.1±1.4 / 73.4±1.0	39.4±0.8 / 36.8±2.1	80.4±1.4 / 80.3±1.9	63.6±1.7 / 36.2±0.7
w/o trajectory-distribution matching	76.8±0.6 / 76.0±0.7	37.2±1.5 / 35.5±3.4	76.5±3.6 / 77.3±3.5	63.1±0.7 / 34.7±0.8
w/o task-specific semantic alignment	75.1±0.9 / 74.1±1.2	41.0±0.8 / 40.7±0.8	75.2±1.3 / 75.9±1.1	57.3±0.6 / 33.7±0.5
Full	**77.7±0.5 / 77.2±0.5**	**43.2±1.3 / 42.8±1.4**	**83.1±1.5 / 83.6±1.6**	**66.7±0.4 / 40.2±0.4**

The three ablations independently replace trajectory-prototype initialization, remove complete-trajectory distribution matching, or remove Task-Specific Semantic Alignment while preserving the student evaluation protocol. Each tested module improves mean F1 relative to its removal, with dataset-dependent differences in effect size.

E.3 Parameter Sensitivity

Because the sensitivity analyses are independent five-seed reruns, their standard deviations differ slightly from those in Table 1. Table 9 varies λ_{dyn}, μ, and β one at a time around their main values of 10, 0.1, and 1, respectively; $\lambda_{\text{sem}} = 1$ and $\tau = 2$ remain fixed. The Reddit and Arxiv sweeps show broadly stable behavior around these defaults, although the preferred neighborhood differs slightly with graph scale.

Table 9: Sensitivity to the trajectory-distribution weight λ_{dyn}, prototype-regularization weight μ, and semantic soft-target coefficient β on Reddit ($r = 2.5\%$) and Arxiv ($r = 0.25\%$). Each parameter is varied independently over five seeds, with $\lambda_{\text{sem}} = 1$, $\tau = 2$, and all other parameters fixed at their default values. Bold rows denote the main-experiment defaults.

Parameter	Value	Reddit		Arxiv	
		Micro-F1	Macro-F1	Micro-F1	Macro-F1
λ_{dyn}	1	41.9±0.8	41.8±0.7	66.2±0.5	39.8±0.2
	5	42.4±1.6	42.3±1.5	66.4±0.8	40.2±0.7
	10	**43.2±0.5**	**42.8±0.5**	**66.6±0.3**	**39.6±0.4**
	20	42.8±0.9	42.1±0.9	66.2±0.5	39.5±0.5
	50	43.0±1.3	42.6±1.2	65.3±0.8	38.2±0.4
μ	0	42.5±0.6	42.1±0.7	66.6±0.3	39.6±0.4
	0.05	42.5±1.6	41.9±1.4	66.6±0.3	39.6±0.5
	0.1	**43.2±0.5**	**42.8±0.5**	**66.5±0.5**	**39.7±0.6**
	0.2	43.0±0.8	42.4±0.5	66.5±0.8	39.7±0.4
	0.5	43.7±0.9	43.3±1.2	66.2±0.5	39.7±0.4
β	0	42.2±0.9	41.2±1.3	65.5±0.8	39.6±0.5
	0.1	41.9±1.3	40.9±2.8	66.3±0.6	39.4±0.8
	0.5	42.8±1.8	42.4±1.6	66.1±0.4	40.2±0.2
	1	**43.2±0.6**	**42.8±0.7**	**66.5±0.6**	**39.7±0.8**
	2	42.3±1.3	41.8±1.2	65.7±0.8	40.0±0.4

E.4 Amortized Total Construction Efficiency

Table 10 reports teacher pre-training, ratio-specific condensation, and their sum for one randomly selected seed, chosen before timing. Teacher pre-training is amortized over the three ratios evaluated per dataset and is applied identically to TP-DGC and DyGC. Consistent with Table 3, all reported times exclude downstream student training and testing. Under this construction-only protocol, TP-DGC reduces total time by 2.3×–115.1× relative to DyGC in these representative runs.

Table 10: Amortized total synthetic-data construction time (seconds; one randomly selected seed). Pre-training is shared by DyGC and TP-DGC and amortized over three ratios; student training and testing are excluded.

Method	**Reddit** (r = 5.0%)			**DBLP** (r = 5.0%)			**Brain** (r = 5.0%)			**Arxiv** (r = 0.5%)		
	Pre-train	**Cond.**	**Total**	**Pre-train**	**Cond.**	**Total**	**Pre-train**	**Cond.**	**Total**	**Pre-train**	**Cond.**	**Total**
SimGC	0.0	567.7	567.7	0.0	292.5	292.5	0.0	261.3	261.3	0.0	2134.9	2134.9
GECC-Snapshot	0.0	952.1	952.1	0.0	169.1	169.1	0.0	146.7	146.7	0.0	800.4	800.4
CGC-X	0.0	3.7	3.7	0.0	8.4	8.4	0.0	3.7	3.7	0.0	32.9	32.9
DeepCGC-X	0.0	2.0	2.0	0.0	2.8	2.8	0.0	2.5	2.5	0.0	21.6	21.6
DyGC	15.9	2827.0	2842.9	13.6	844.0	857.6	19.4	575.2	594.6	495.3	699.1	1194.4
TP-DGC	15.9	8.8	24.7	13.6	16.2	29.8	19.4	7.1	26.5	495.3	22.3	517.6

F Additional Experiments and Extensions

F.1 Extension to Real Timestamp-Based Node Arrivals

Motivation and construction. The primary benchmarks follow the fixed-universe convention used by existing dynamic graph condensation protocols. To test whether persistent synthetic identities depend on this convention, we construct a cumulative ogbn-arxiv sequence from actual publication years. Using cutoffs 2010, 2012, 2014, 2016, 2018, and 2020 yields six snapshots with 12,974 → 23,836 → 41,125 → 69,499 → 120,740 → 169,343 active nodes. For paper i with publication year b_i and cutoff c_t, availability is $m_i^t = \mathbb{I}(b_i \le c_t)$. The snapshot sequence and edge sets are then defined by:

$$V_1 \subseteq \cdots \subseteq V_T, \qquad E_t = \{(i,j) \in E : m_i^t = m_j^t = 1\}. \tag{29}$$

Thus an unpublished paper contributes neither features nor incident edges to an earlier snapshot. Once published, a paper remains present in all subsequent snapshots.

Masked trajectories and leakage control. In this extension, a real trajectory is represented as $\tau_i = \{(m_i^t, Z_{t,i})\}_{t=1}^{T}$, and only positions with $m_i^t = 1$ are valid observations. Pre-birth positions are treated as missing rather than as semantic zero vectors. The persistent identity constraint remains attached to the synthetic prototype, while real nodes may enter that prototype's class-conditional trajectory statistics after arrival. Lifecycles are derived only from publication years. Pre-birth features and edges are hidden; feature normalization, prototype selection, trajectory statistics, semantic matching, and adaptive stopping use official training nodes only. Validation selects the checkpoint, and test labels are accessed only for final reporting.

Protocol. For each seed in $\{1, 2, 3, 4, 5\}$, we retrain the TGCN-L teacher under the arrival protocol instead of reusing a fixed-universe checkpoint. The teacher uses three layers, hidden dimension 256, and at most 100 epochs. TP-DGC follows the official class-wise large-graph budget $M_c = \lceil rN_c \rceil$ at $r \in \{0.05\%, 0.25\%, 0.5\%\}$, producing 69, 248, and 477 synthetic nodes, respectively. Students are trained for 2,000 epochs and selected by validation Micro-F1. Each condensed run is paired with the teacher of the same seed, and the final synthetic sequence is edge-free. Table 11 reports the resulting five-seed mean and standard deviation.

Table 11: Timestamp-based node-arrival results on ogbn-arxiv (%, mean ± standard deviation over five seeds). Fidelity is measured relative to the full-graph teacher under the same arrival protocol. The best condensed setting is bolded.

Setting	**Syn. nodes**	**Micro-F1**	**Macro-F1**	**Micro fid.**	**Macro fid.**
Full graph	169,343	70.0±1.4	45.8±0.9	100.0%	100.0%
0.05%	69	55.5±3.4	33.4±1.2	79.2%	73.0%
0.25%	248	64.5±2.1	39.7±0.9	92.2%	86.7%
0.5%	477	**67.7±0.6**	**42.8±1.3**	96.7%	93.5%

Despite a more than 13× increase in active nodes, the 0.5% sequence retains 96.7% and 93.5% of the full-graph Micro-F1 and Macro-F1, respectively. Performance improves with the synthetic budget. This experiment covers one dataset and monotone arrivals; deletion, re-entry, and arbitrary missingness remain outside its scope.