Multi-Frequency Associative Memory for Continual Graph Learning through Nested Optimization

Shuvam Kundu

Multi-Frequency Associative Memory for Continual Graph Learning through Nested Optimization

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Computer Science

by

Shuvam Kundu

[Roll No: CS-2429]

under the guidance of

Dr. Swagatam Das

Professor

Electronics and Communication Sciences Unit

Indian Statistical Institute
Kolkata-700108, India

June 2026

CERTIFICATE

This is to certify that the dissertation entitled **"Multi-Frequency Associative Memory for Continual Graph Learning through Nested Optimization"** submitted by **Shuvam Kundu** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Dr. Swagatam Das
Professor,
Electronics and Communication Sciences Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Acknowledgements

I sincerely thank my advisor, *Prof. Swagatam Das*, of the Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata, for his steadfast mentorship and invaluable support throughout this dissertation. His expert guidance and encouragement have played a pivotal role in shaping the direction and quality of my research.

I gratefully acknowledge *Kushal Bose*, Senior Research Fellow under Prof. Das, for his insightful discussions, technical suggestions, and timely feedback that enriched the depth of this work.

My appreciation also extends to the faculty and staff of the Indian Statistical Institute for fostering a vibrant academic environment and for providing the facilities essential for completing this research.

I am thankful to my friends and batchmates for their consistent encouragement, knowledge sharing, and collaborative mindset during this journey.

Most importantly, I am profoundly grateful to my parents, family, and my partner, Pronomita, for their unwavering love, patience, and faith in me throughout this academic journey.

Shuvam Kundu
Indian Statistical Institute
Kolkata : 700108, India.

Abstract

Graph Neural Networks struggle to learn new tasks without forgetting old ones—a problem known as catastrophic forgetting. In graph domains, this is compounded by structural shift, where newly added edges corrupt the learned representations of historical nodes even when model weights remain unchanged. We present **CAM-Titans**, a continual graph learning framework built around a two-buffer associative memory to address both parametric and structural forgetting. Our architecture operates across three timescales of adaptation: a slow base memory updated via ordinary gradient descent, an intermediate task buffer re-encoded after every task using the delta-rule, and a transient in-context state for rapid within-pass adaptation. To ensure historical class prototypes remain retrievable as the network backbone evolves, memory retrieval is anchored in a dynamically maintained prototype coordinate system. Furthermore, a cosine classifier mitigates magnitude imbalance, preventing older classes from dominating predictions. Empirical evaluations across diverse continual learning benchmarks demonstrate that CAM-Titans effectively mitigates catastrophic forgetting, achieving superior stability and accuracy in both Task-Incremental and Class-Incremental settings.

Keywords: Continual Graph Learning, Associative Memory, Catastrophic Forgetting, Structural Shift, Graph Neural Networks, Class-Incremental Learning

Contents

List of Figures

List of Tables

List of Algorithms

List of Notations

$G = (V, E)$	Graph with node set V and edge set E
$N = \|V\|$	Number of nodes
$\hat{A} = D^{-1/2}AD^{-1/2}$	Symmetrically normalised adjacency matrix
$X^{(k)} = \hat{A}^k X$	k-hop smoothed feature matrix
S	Number of hops in multi-hop encoding
d	Hidden model dimension
T	Total number of tasks
C	Total number of classes seen so far
Θ	All learnable model parameters
M_{base}	Persistent base memory (updated by SGD)
M_{task}	Task-level memory buffer (updated per task)
M_{new}	Transient in-context memory state
$\eta_t,\ \alpha_t$	Per-token learning rate and retention gate
$k_t,\ v_t,\ q_t$	Key, value, and query vectors
p_c	Prototype vector for class c
$P \in \mathbb{R}^{C \times d}$	Matrix of all class prototypes
r_i	Residual target for anchor i in Phase 3
σ_{eff}	Cosine classifier temperature
$\lambda_{\text{kv}},\ \lambda_{\text{prox}},\ \lambda_{\text{kd}}$	Loss weighting coefficients
AP	Average Performance (mean accuracy over all tasks)
AF	Average Forgetting (mean accuracy drop from peak)
$M^p_{t,j}$	Accuracy on task j after training on task t

Chapter 1

Introduction

Graph Neural Networks (GNNs) [1, 2] have become the typical tool for learning on relational data – social networks, citation graphs, molecular structures, and e-commerce systems. Virtually all deployed GNNs assume that the data is *static* and is available at training time. Real-world graphs are dynamic: nodes and edges are added continuously, and naïve retraining of the model is performed. Computing from scratch is impractical and sometimes illegal.

Continual Learning (CL) [3, 4] addresses this by training models one after another over a sequence of tasks and keeping performance on earlier ones. The main difficulty is *catastrophic forgetting* [5, 6]: gradient updates for a new task will overwrite parameters that are important for previous tasks.

Applying CL to graphs adds another non-Euclidean challenge. New nodes and edges are added, linked to the old nodes and modify the multi-hop neighbourhoods, shifting old-task input distributions *even with frozen parameters*. This is referred to as *structural shift* [7], and makes standard CL methods essentially useless for graphs.

Continual Graph Learning (CGL) [8, 9] is the paradigm that tackles both parametric and structural forgetting simultaneously. Current CGL techniques focus on either one or the other: Topology is not considered by regularisation methods and nodes are treated as isolated by replay methods. Task identity at test time is required for architecture methods.

This dissertation introduces a framework called **CAM-Titans** that addresses all three gaps in a principled two-frequency associative memory based on the *Nested Learning* framework [10].

1.1 Continual Learning on Graphs

1.1.1 Problem Setting

A CGL system is given a series of graph tasks $\{T_1, T_2, \ldots, T_\tau\}$. For each task T_k there is a graph $\mathcal{G}_{T_k} = (\mathcal{V}_{T_k}, \mathcal{E}_{T_k}, X_{T_k}, Y_{T_k})$. The cumulative graph evolves as

$G_t = G_{t-1} \cup \Delta G_t$. The learning objective is to find parameters Θ_t minimising the joint loss over all tasks seen so far:

$$\mathcal{L}_\Theta = \sum_{k=1}^{t} \ell(\Theta_t;\, A_k, X_k, Y_k), \tag{1.1}$$

using only the current task data ΔG_t and a small memory buffer. Performance is evaluated using the **Average Performance (AP)** which is the mean accuracy over all tasks,**Average Forgetting (AF)** : decline in per-task accuracy.

1.1.2 Incremental Learning Scenarios

Following Van de Ven and Tolias, 2019 [11], three scenarios are distinguished by what information is available at test time.

- **Task-IL**: task identity provided at test time; model uses separate output heads per task. Least challenging.
- **Domain-IL**: task identity withheld; label space is fixed but input distribution shifts.
- **Class-IL**: task identity withheld *and* the label set grows with each task. Most challenging and practically relevant. Standard regularisation methods collapse to near-random accuracy here.

1.1.3 Structural Catastrophic Forgetting

In a GNN, the embedding of node v is based on its multi-hop neighbourhood. $\mathcal{N}(v)$. If new task edges are attached to old nodes, the receptive field of old nodes expands. Liu et al. [7] prove that under unbounded structural shift the error rate on old tasks approaches that of a random classifier, irrespective of the expressivity of the hypothesis space. Therefore arameter preservation alone is not enough, methods need to be stabilised as well as the topological aggregation context of historical nodes.

1.1.4 Problem Statement

The main goal of this dissertation is to develop a continuous graph learning system that is able to learn new tasks, $\{T_1, T_2, \ldots, T_\tau\}$, incrementally while maintaining the following strict operational constraints:

- **No Task Identifiers (Class-IL):** The model should be able to dynamically distinguish among all classes it has seen during inference without the use of oracle task labels.

- **Bounded Memory:** The system does not support unbounded replay of historical data, but instead has a small, fixed size memory buffer.
- **Dual-Axis Stability:** The architecture should ensure that gradient updates do not destroy historical parameters and newly added edges do not corrupt historical multi-hop topological representations of historical nodes.

These requirements cannot be met by any existing regularisation, replay, and isolated-architecture methods, so this work proposes a multi-timescale memory system that decouples and protects both structural and parametric knowledge of previous tasks.

1.2 Thesis Organisation

- **Chapter 2** reviews related work and states thesis objectives.
- **Chapter 3** covers notation, GNNs, and the Titans delta-rule derivation.
- **Chapter 4** validates Parallel Titans on static benchmarks.
- **Chapter 5** presents the full CAM-Titans architecture and training pipeline.
- **Chapter 6** gives pseudocode and complexity analysis.
- **Chapter 7** reports experimental results on CGLB.
- **Chapter 8** gives conclusion, limitations and future work.

Chapter 2

Related Work

2.1 Continual Learning

Regularisation-based methods protect important parameters from large updates. EWC [12] penalises changes to parameters with high Fisher Information; MAS [13] estimates importance from output sensitivity without labels; SI [14] accumulates importance online along the gradient trajectory. All share a common failure in long task sequences: the cumulative constraints over-restrict the parameter space.

Replay-based methods re-expose the model to historical data during new task training. ER [15] stores raw samples in a fixed buffer; GEM [16] constrains new-task gradients so that old-task loss cannot increase; DER++ [17] stores both samples and output logits for distillation-based replay; LwF [18] achieves replay without storing data by distilling from a pre-task model snapshot.

Architecture-based methods isolate parameters per task. PNN [19] instantiates a new column per task with lateral connections; PackNet [20] prunes and re-allocates capacity iteratively; HAT [21] learns binary task masks; L2P and DualPrompt [22, 23] keep a frozen pre-trained backbone and learn lightweight task-specific prompts.

2.2 Continual Graph Learning

Structural shift : Liu et al., 2021 [7] formalise structural shift in the Node-wise Graph Incremental Learning (NGIL) setting and prove that unbounded structural shift drives old-task error to random-classifier level. This motivates topology-aware methods beyond standard parameter regularisation.

Regularisation-based CGL : TWP [7] augments EWC with topology-aware importance scores that account for each parameter's contribution to GNN attention and aggregation, rather than classification loss alone. GraphSAIL [24] distils local structure, global structure, and self-information from a frozen teacher to mitigate

forgetting in recommender systems. SSRM [25] minimises the latent-space divergence caused by structural shift; applying the constraint before the final hidden layer allows the output head to adapt flexibly while stabilising representations.

Replay-based CGL : ER-GNN [26] stores historical nodes and replays them alongside new-task batches; node selection uses mean-of-feature, coverage, or influence maximisation. SSM [27] preserves topology in the buffer by expanding a subgraph around each selected node via random walk. SEA-ER [28] populates the buffer by maximising structural similarity between selected samples and the remainder of the graph, with theoretical guarantees on learnability. CaT [29] condenses each incoming graph into a small synthetic graph $\tilde{G}_k$ by minimising $\text{Dist}(\text{GNN}_\theta(G_k), \text{GNN}_\theta(\tilde{G}_k))$, storing the condensate instead of raw nodes. PUMA [30] extends CaT to unlabelled nodes via pseudo-label guided distribution matching and replaces GNN-based condensation with edge-free MLPs, reducing condensation cost from $O(N^2)$ to $O(N)$. PUMA is the strongest prior method and our primary baseline.

Architecture-based CGL : HPNs [31] extract multi-level prototypes (atom, node, class) via Atomic Feature Extractors; memory consumption is proven to be bounded regardless of task count. FGN [32] converts node classification to graph classification by treating features as nodes, enabling standard lifelong learning isolation techniques. TPP [33] freezes the base GNN and learns a lightweight task-specific prompt per task; forgetting is zero by construction but requires reliable task-identity inference. DyMoE [34] instantiates a dedicated expert sub-network per task block, with sparse top-k routing at inference for efficiency.

Benchmarks : CGLB [8] is the primary benchmark used in this work: four node-level datasets (CoraFull-CL, Arxiv-CL, Reddit-CL, Products-CL) with standardised Class-IL and Task-IL splits.

2.3 Graph Transformer Backbones

NAGphormer [35] encodes multi-hop features via the *Hop2Token* strategy: the 0-hop to S-hop smoothed features of each node form a sequence of $S+1$ tokens fed to a Transformer encoder. Topological propagation is precomputed once, decoupled from task-specific training—aligned with the SIGN paradigm [36]. This is the primary backbone for CAM-Titans. **GraphGPS** [37] alternates local MPNN and global linear attention at $O(N+E)$ complexity. **SAT** [38] concatenates k-hop subgraph representations to node features before computing attention, explicitly encoding local topology. Both GPS and SAT are used in the static validation experiments of Chapter 4, where their global attention is replaced by Parallel Titans.

2.4 Motivation

The survey above reveals four open gaps that motivate this work.

1. **Two-axis forgetting.** Current literature lacks an unified framework that simultaneously handles both parametric forgetting and structural shift without requiring task identity at test time.

2. **Address-drift problem.** Methods that store prototypes (e.g. PUMA, HPNs) do not account for the fact that prototype coordinates shift as backbone parameters evolve. A prototype written under task-k parameters is misaddressed by task-$(k + j)$ queries—an issue not explicitly identified or resolved in the literature.

3. **Magnitude imbalance in Class-IL.** PUMA and CaT use standard inner-product classifiers. Old-class weight vectors accumulate gradient over many tasks; new-class vectors receive gradient for only one, biasing predictions toward recently seen classes.

4. **Absence of multi-frequency memory.** All existing GNN continual learners use a single weight matrix for both within-task adaptation and cross-task consolidation, providing no architectural separation between fast transient knowledge and slow durable knowledge.

Chapter 3

Preliminaries

This Chapter sets out the notations and theory that are used throughout the dissertation. Readers familiar with graph neural networks may skip Section 3.1 and proceed directly to Section 3.2.

3.1 Notations and Graph Basics

A *graph* is a pair $G = (V, E)$ where V is a finite set of $N = |V|$ nodes and $E \subseteq V \times V$ is a set of edges. In this dissertation, we will use undirected and unweighted graphs. The adjacency matrix is $A \in 0, 1^{N \times N}$, the degree matrix is D, and node features are represented by $X \in \mathbb{R}^{N \times F}$. The symmetrically normalised adjacency $\hat{A} = D^{-1/2} A D^{-1/2}$ has eigenvalues are in $[-1, 1]$, and will be used in the following standard practice [1]. Multi-hop smoothed features are precomputed as $X^{(k)} = \hat{A}^k X$ for $k = 0, \ldots, S$, forming the stack $\boldsymbol{X} \in \mathbb{R}^{N \times (S+1) \times F}$ used by the SIGN paradigm [36] and followed in this work.

We use NAGphormer [35] that encodes each node as a sequence of $S + 1$ hop tokens through this precomputed stack and uses a Transformer encoder to process them. No further GNN-specific background is required beyond the message-passing update.

$$\mathbf{h}_v^{(\ell)} = \mathrm{UPDATE}^{(\ell)}\Big(\mathbf{h}_v^{(\ell-1)},\ \mathrm{AGGREGATE}^{(\ell)}\big(\{\mathbf{h}_u^{(\ell-1)} : u \in \mathcal{N}(v)\}\big)\Big), \tag{3.1}$$

which are the basis of GCN [1] and GAT [2], and all variations as baseline in Chapter 4.

3.2 Associative Memory and Delta Rule

The Nested Learning (NL) framework [10] treats neural networks as hierarchical systems of *associative memories*, each solving an online regression problem on their own context flow and update frequency. An associative memory is an operator $\mathcal{M}(\cdot)$

mapping keys to values, optimised as

$$\mathcal{M}^* = \arg\min_{\mathcal{M}} \; \tilde{\mathcal{L}}\big(\mathcal{M}(K);\, V\big). \tag{3.2}$$

Choosing an ℓ_2-regression objective and applying one proximal gradient step yields the **Delta Gradient Descent (DGD)** objective:

$$W_{t+1} = \arg\min_{W} \; \tfrac{1}{2}\big\|W\mathbf{x}_t - \mathbf{u}_t\big\|_2^2 + \tfrac{1}{2\eta_t}\|W - W_t\|_2^2, \tag{3.3}$$

whose closed-form solution (via the Sherman–Morrison lemma, with effective rate $\eta_t' = \eta_t/(\lambda^2\eta_t + 1)$) is

$$W_{t+1} = W_t\big(I - \eta_t' \,\mathbf{x}_t\mathbf{x}_t^\top\big) - \eta_t' \,\nabla_{y_t}\mathcal{L}(W_t; \mathbf{x}_t)\,\mathbf{x}_t^\top. \tag{3.4}$$

Applying DGD to a pure associative memory storing pairs $(\mathbf{k}_t, \mathbf{v}_t)$ under the loss $\frac{1}{2}\|\mathcal{M}\mathbf{k}_t - \mathbf{v}_t\|_2^2$ gives the **Titans delta rule** [10]:

$$\mathcal{M}_t = \mathcal{M}_{t-1}\big(\alpha_t I - \eta_t \mathbf{k}_t\mathbf{k}_t^\top\big) - \eta_t\big(\mathcal{M}_{t-1}\mathbf{k}_t - \hat{\mathbf{v}}_t\big)\mathbf{k}_t^\top, \tag{3.5}$$

where $\alpha_t \in (0, 1)$ is a retention gate enabling *selective forgetting* and $\hat{\mathbf{v}}_t$ is a self-referentially generated value target. The update has two interpretable components: (i) a *decay term* $\mathcal{M}_{t-1}(\alpha_t I - \eta_t\mathbf{k}_t\mathbf{k}_t^\top)$ that projects out the direction of $\mathbf{k}_t$ to free capacity, and (ii) a *residual write* $-\eta_t(\mathcal{M}_{t-1}\mathbf{k}_t - \hat{\mathbf{v}}_t)\mathbf{k}_t^\top$ that writes only the prediction error, leaving already-correct associations unchanged. Reading from memory is a simple matrix-vector product $\mathbf{y}_t = \mathcal{M}_t\mathbf{q}_t$, giving $\mathcal{O}(d^2)$ cost per token versus $\mathcal{O}(T^2d)$ for softmax attention.

Self-referential gating. The scalars η_t and α_t are generated by auxiliary memory cells $\mathcal{M}_\eta$ and $\mathcal{M}_\alpha$. The model is updated using the same delta rule, so the model learns its own plasticity schedule from the data. Another cell $\mathcal{M}_v$ generates the value target $\hat{\mathbf{v}}_t = \mathcal{M}_{v,t-1}(\mathbf{x}_t)$, replacing the static projection $W_v\mathbf{x}_t$ of standard attention and allowing key-value associations to co-evolve with the memory state in context.

3.3 Continuum Memory System

The *Continuum Memory System* (CMS) [10] generalises the short-term/long-term memory dichotomy by organising feedforward blocks into a spectrum of update frequencies $f_1 > f_2 > \cdots > f_k$. High-frequency blocks change quickly to the immediate context; low-frequency blocks encode persistent, cross-context knowledge. The forward pass links these blocks together as

$$\mathbf{y}_t = \mathrm{MLP}^{(f_k)}\big(\cdots \mathrm{MLP}^{(f_1)}(\mathbf{x}_t)\big), \tag{3.6}$$

with parameters $\boldsymbol{\theta}^{(f_\ell)}$ updated every $C^{(\ell)}$ steps. It is a multi-frequency design that offers two complementary safeguards against catastrophic forgetting: (i) If knowledge is overwritten in a fast block, it may still be in slower block. (ii) the shared low-frequency gradient path allows partial recovery of forgotten associations.

3.4 Cosine Classifier for Incremental Learning

In Class-IL settings, the weight matrix $W \in \mathbb{R}^{C\times d}$, Increases with the addition of new classes. The rows that contain early classes accumulate gradient updates over all tasks, while rows corresponding to recent classes receive updates only from the latest task, creating a *magnitude imbalance* problem. Old-class weight-vectors have significantly larger ℓ_2-norms than the vectors of the new class. Under a standard inner-product classifier $f(\mathbf{r}) = W\mathbf{r}$, this creates a systematic bias in favour of older classes.

To reduce magnitude dependence, the cosine classifier normalises Weight rows as well as feature vectors:

$$f(\mathbf{r}) \;=\; \sigma_{\text{eff}} \cdot \frac{W}{\|W\|_2} \cdot \frac{\mathbf{r}}{\|\mathbf{r}\|_2}, \tag{3.7}$$

where $\sigma_{\text{eff}} > 0$ is a temperature parameter controlling sharpness of the class-logit distribution. The dot product now only measures cosine similarity, mitigating the magnitude bias regardless of how many gradient updates each class row has received.

Chapter 4

Titans on Static Graphs

This chapter demonstrates that the Titan associative memory module is a drop-in alternative to quadratic self attention for graphs. We embedd Parallel Titans into graph transformers,evaluate on standard node and graph level benchmarks and show that it acheives accuracy $\mathcal{O}(Bd^2)$ complexity instead of $\mathcal{O}(B^2d)$, where B is the batch size. This static validation clearly distinguishes between the *architectural* contribution (associative memory for graphs) from the *continual-learning* contribution developed in Chapter 5:

4.1 From Attention to Associative Memory

The sequence modeling problem is a form of online regression problem, as done by Titans [39]. We can start from the linear-attention recurrence

$$M_t = M_{t-1} + \phi(\mathbf{k}_t)\mathbf{v}_t^\top, \tag{4.1}$$

where $M_t \in \mathbb{R}^{d\times d}$ contains all key-value. associations observed thus far. Titans replace this accumulation with the *delta rule*:

$$M_t = \alpha_t M_{t-1} - \eta_t\big(M_{t-1}\mathbf{k}_t - \hat{\mathbf{v}}_t\big)\mathbf{k}_t^\top, \tag{4.2}$$

where α_t is a retention gate whose value is $(0, 1)$ and allows *selective forgetting* and $\eta_t > 0$ is a per-token learning rate. Both scalars are created *self-referentially* by auxiliary memory cells M^α and M^η that are updated by the delta rule. So, the model learns its own plasticity from data, instead of fixed value. The third auxiliary cell M^v adds an interpolated value target $\hat{\mathbf{v}}_t = M_t^v\mathbf{x}_t$, replacing the static projection $W_v\mathbf{x}_t$ of standard attention. Full derivations are given in Chapter **??**. For our implementation we chose $\eta_{\max} = 0.05$. $\alpha_{\min} = 0.94$, $\alpha_{\max} = 0.99$ across all experiments.

4.2 Parallel Titans for Graph-Structured Data

4.2.1 The Ordering Problem

If we apply the sequential delta rule (4.2) directly to a mini-batch of nodes, it requires an arbitrary node ordering. Because the delta update is non-commutative, processing node i before node j yields a different memory state than the reverse order. This violates the *permutation equivariance* required of graph neural networks: relabelling nodes should relabel outputs identically without changing predictions.

4.2.2 Batch-Level Formulation

We solve the ordering problem by traversing all B nodes in a mini-batch *simultaneously*. The delta rule generalises naturally to a batch optimisation objective:

$$M^* = \arg\min_M \sum_{n=1}^{B} \left\| M\mathbf{k}_n - \hat{\mathbf{v}}_n \right\|^2 + \tfrac{1}{2\eta} \|M - M_{\text{prev}}\|_F^2, \tag{4.3}$$

Whose closed-form solution is the **Parallel Titans** update:

$$M_{\text{new}} = \alpha M_{\text{prev}} - \eta\, M_{\text{prev}} K K^\top - \eta\, (M_{\text{prev}} K - \hat{V})\, K^\top, \tag{4.4}$$

where $K = [\mathbf{k}_1, \ldots, \mathbf{k}_B] \in \mathbb{R}^{d \times B}$ and $\hat{V} = [\hat{\mathbf{v}}_1, \ldots, \hat{\mathbf{v}}_B] \in \mathbb{R}^{d \times B}$. Similarly, reading from memory is parallel:

$$Y = M_{\text{new}}\, Q, \qquad Q = [\mathbf{q}_1, \ldots, \mathbf{q}_B] \in \mathbb{R}^{d \times B}. \tag{4.5}$$

All matrix products are done in a single GPU pass (no loop over). This mechanism is both efficient and permutation-equivariant.

Mechanism	Time (per layer)	Space
Softmax attention	$\mathcal{O}(B^2 d)$	$\mathcal{O}(B^2)$
Linear attention	$\mathcal{O}(Bd^2)$	$\mathcal{O}(d^2)$
Parallel Titans (ours)	$\mathcal{O}(Bd^2)$	$\mathcal{O}(d^2)$

4.3 Integration into Graph Transformer Architectures

We apply Parallel Titans into three graph transformer architectures by replacing their global self-attention layer, with all other elements of the model unchanged, i.e. the (MPNN layers, positional encodings, readout heads) remain the same.

NAGphormer → NAG-Titans. NAGphormer [35] encodes each node as a sequence of $S+1$ hop tokens $\{\mathbf{h}_n^{(k)}\}_{k=0}^{S}$ via the The Hop2Token strategy and processes them using Transformer encoder. In NAG-Titans, each Transformer layer replaces its self-attention with the Parallel Titans update over the $S+1$ hop tokens of each node:

$$M_{\text{new}}^n = \alpha M_{\text{prev}}^n - \eta\, M_{\text{prev}}^n K^n K^{n\top} - \eta\,(M_{\text{prev}}^n K^n - \hat{V}^n) K^{n\top}, \tag{4.6}$$

where $K^n, \hat{V}^n$ are the key and value matrices formed from the $S+1$ hop tokens of node n.

GraphGPS → GPS-Titans. GraphGPS [37] embeds a local MPNN and a Self-attention throughout the network in linear time globally across all layers. GPS-Titans replaces the global attention with Parallel Titans applied to all the nodes in the (sub)graph.

SAT → SAT-Titans. The Structure-Aware Transformer [38] is a model that concatenates k-hop subgraph embeddings (produced by an auxiliary GNN) to node features before computing attention, explicitly encoding local features into the queries and keys. SAT-Titans replaces SAT's self-attention with Parallel Titans. The subgraph-embedding pipeline is retained without modification.

4.4 Experiments

4.4.1 Setup

Node classification (NAG-Titans). We test on six node classification tasks: Pubmed, CoraFull (from PyTorch Geometric), and the four Amazon/Co-author graphs Computer, Photo, CS, and Physics. The datasets come in a variety of scales (2.7K–172K nodes) and feature dimensionalities (500–8189). All models are fitted using Adam for 1500 epochs with early stopping on validation loss at $lr = 10^{-3}$. Accuracy is reported as mean ± std over 10 random seeds. Titans hyperparameters (H=4, d=128, $\eta_{\max}$=0.05, $\alpha_{\min}$=0.94, $\alpha_{\max}$=0.99) are held fixed across all datasets.

Graph-level benchmarks (GPS-Titans, SAT-Titans). We use the same training procedures as GraphGPS and SAT on five benchmarks from the Benchmarking GNN [40]: The datasets used are: ZINC (MAE↓), MNIST, CIFAR10, PATTERN, and CLUSTER. (Acc↑).

4.4.2 Results

Tables 4.1 and 4.2 report results on node-classification and graph-level benchmarks.

Table 4.1: Test accuracy (%, mean ± std over 10 seeds) on static node-classification benchmarks. **Bold**: best overall; underline: best among transformer-based methods; OOM: out-of-memory on a 24 GB GPU.

Method	Pubmed	CoraFull	Computer	Photo	CS	Physics
GCN	86.54 ±0.12	61.76 ±0.14	89.65 ±0.52	92.70 ±0.20	92.92 ±0.12	96.18 ±0.07
GAT	86.32 ±0.16	64.47 ±0.18	90.78 ±0.13	93.87 ±0.11	93.61 ±0.14	96.17 ±0.08
GraphSAINT	88.96 ±0.16	67.85 ±0.21	90.22 ±0.15	91.72 ±0.13	94.41 ±0.09	96.43 ±0.05
GPRGNN	89.34 ±0.25	67.12 ±0.31	89.32 ±0.29	94.49 ±0.14	95.13 ±0.09	96.85 ±0.08
APPNP	88.43 ±0.15	65.16 ±0.28	90.18 ±0.17	94.32 ±0.14	94.49 ±0.07	96.54 ±0.07
GRAND+	88.64 ±0.09	71.37 ±0.11	88.74 ±0.11	94.75 ±0.12	93.92 ±0.08	96.47 ±0.04
GT	88.79 ±0.12	61.05 ±0.38	91.18 ±0.17	94.74 ±0.13	94.64 ±0.13	97.05 ±0.05
Graphormer	OOM	OOM	OOM	92.74 ±0.14	OOM	OOM
SAN	88.22 ±0.15	59.01 ±0.34	89.83 ±0.16	94.86 ±0.10	94.51 ±0.15	OOM
GraphGPS	88.94 ±0.16	55.76 ±0.23	OOM	95.06 ±0.13	93.93 ±0.12	OOM
NAGphormer	**89.70** ±0.19	71.51 ±0.13	**91.22** ±0.14	**95.49** ±0.11	**95.75** ±0.09	**97.34** ±0.03
NAG-Titans (ours)	89.67 ±0.43	**72.21** ±0.78	91.07 ±0.47	95.43 ±0.46	95.53 ±0.22	96.95 ±0.20

4.4.3 Discussion

NAG-Titans. NAG-Titans matches or surpasses NAGphormer on five of six datasets, with the best overall score on CoraFull (+0.70%). Slightly higher standard deviations are expected given the frozen hyperparameter protocol; the competitive accuracy under these conditions indicates mechanism robustness rather than careful fitting. NAG-Titans also avoids the OOM failures seen for Graphormer, SAN, and GraphGPS on larger graphs, a direct consequence of the $\mathcal{O}(Bd^2)$ footprint.

GPS-Titans and SAT-Titans. On classification benchmarks, the GPS-Titans out performs GPS on MNIST. (98.459 vs. 98.051), CIFAR10 (72.700 vs. 72.298), and PATTERN (86.805 vs. 86.685), while CLUSTER is essentially tied. SAT-Titans Almost recovers the performance of K-subtree SAT on PATTERN (86.855 vs. 86.865) and CLUSTER (77.722 vs. 77.751). The sole systematic degradation is on ZINC regression (GPS-Titans MAE = 0.099, SAT-Titans = 0.121 vs. GPS = 0.070), where the fixed $\eta_{\max}/\alpha$ schedule is not enough for predicting heterogeneous bond-lengths; This is considered to be a limitation and is left as a future work. Importantly, this failure is task specific and does not affect the node-classification objectives of the continual learning experiments in Chapter 5.

Summary. In all three architectures and eleven benchmarks, Parallel Titans is always competitive with the attention baseline in a fixed, non-tuned hyperparameter protocol, conforming three conclusions: **(i)** *Feasibility*: Associative memory is a viable alternative to quadratic self-attention for graph transformers; **(ii)** *Scalability*: The

Table 4.2: Performance on graph-level benchmarks (Benchmarking GNNs suite). MAE↓ for ZINC; accuracy (%)↑ for all others. **Bold**: best; underline: second best. Baseline results from original papers; "—" indicates not reported.

Method	ZINC MAE↓	MNIST Acc↑	CIFAR10 Acc↑	PATTERN Acc↑	CLUSTER Acc↑
Message-passing GNNs					
GCN	0.367 ±0.011	90.705 ±0.218	55.710 ±0.381	71.892 ±0.334	68.498 ±0.976
GIN	0.526 ±0.051	96.485 ±0.252	55.255 ±1.527	85.387 ±0.136	64.716 ±1.553
GAT	0.384 ±0.007	95.535 ±0.205	64.223 ±0.455	78.271 ±0.186	70.587 ±0.447
GatedGCN	0.282 ±0.015	97.340 ±0.143	67.312 ±0.311	85.568 ±0.088	73.840 ±0.326
PNA	0.188 ±0.004	97.940 ±0.120	70.350 ±0.630	—	—
DGN	0.168 ±0.003	—	72.838 ±0.417	86.680 ±0.034	—
GSN	0.101 ±0.010	—	—	—	—
Higher-order / subgraph methods					
CIN	0.079 ±0.006	—	—	—	—
CRaWI	0.085 ±0.004	97.944 ±0.050	69.013 ±0.259	—	—
GIN-AK+	0.080 ±0.001	—	72.190 ±0.130	86.850 ±0.057	—
Graph transformers					
SAN	0.139 ±0.006	—	—	86.581 ±0.037	76.691 ±0.650
EGT	0.108 ±0.009	98.173 ±0.087	68.702 ±0.409	86.821 ±0.020	**79.232 ±0.348**
GT	0.226 ±0.014	—	—	84.808 ±0.068	73.169 ±0.622
GPS	**0.070 ±0.004**	98.051 ±0.126	72.298 ±0.356	86.685 ±0.059	78.016 ±0.180
K-subtree SAT	0.102 ±0.005	—	—	**86.865 ±0.043**	77.751 ±0.121
Ours (Parallel Titans)					
GPS-Titans	0.099 ±0.007	**98.459 ±0.023**	72.700 ±0.219	86.805 ±0.021	78.012 ±0.357
SAT-Titans	0.121 ±0.009	—	—	86.855 ±0.054	77.722 ±0.417

$\mathcal{O}(Bd^2)$ complexity eliminates $\mathcal{O}(B^2)$ OOM failures on large graphs; **(iii)** *The scope of limitations*: degradation is restricted to regression task and does not affect the classification setting

Chapter 5

Proposed Methodology: CAM-Titans

The architectural design and key algorithmic contributions of CAM-Titans are given in this chapter. Standard graph neural networks are vulnerable to catastrophic forgetting, where the knowledge gained by the previous tasks is easily overwritten by the new gradient updates.

We address this through the lens of Nested Learning [10] by organising model parameters into *three update timescales*. This gives the model the ability to adapt rapidly within a forward pass, consolidate task-level knowledge after each task, and accumulate slow cross-task structure through ordinary backpropagation. The overall architecture is shown in Figure 5.1.

5.1 Memory Requirements in Continual Graph Learning

Suppose that a model has been trained to classify nodes over T tasks. At any moment, three qualitatively different types of memory are useful, distinguished by how frequently they are updated. In the context of Nested Learning [10], this refers to optimisation hierarchies of fast versus slow weights, not any structural nesting of subgraphs.

1. **In-context (batch-level) memory (M_{new}) :** During one forward pass, it captures local neighbourhood statistics. This memory is *transient*: it is created fresh at the start of every forward pass and discarded afterward.
2. **Task-level memory (M_{task}).** Retains class-prototype associations from all tasks observed so far. It is updated once per task via the delta-rule consolidation step (Section 5.6.3).
3. **Cross-task memory (M_{base}).** Stores slow, persistent structural knowledge accumulated over all tasks through standard gradient descent update.

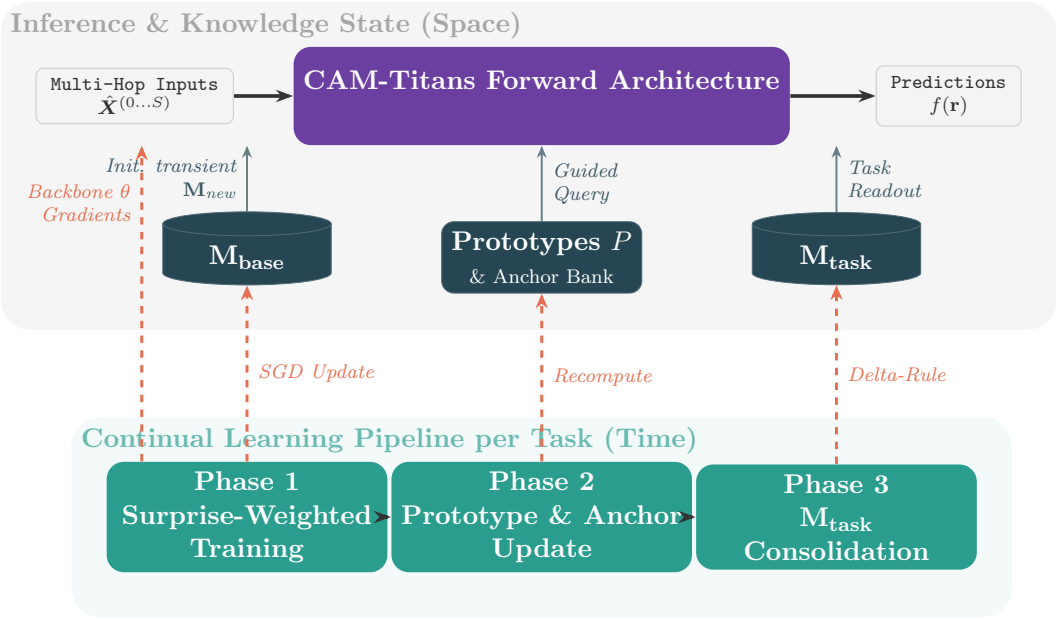


Figure 5.1: Flowchart of the Forward Architecture

Memory component	Update trigger	Frequency
Transient in-context state (M_{new})	every forward pass	highest
Task-level buffer (M_{task})	after every task	intermediate
Base memory (M_{base})	every SGD step	lowest

This is a **three-timescale** system in the sense of the NL frequency hierarchy. From an implementation perspective only M_{base} and M_{task} *persist* across forward passes. We therefore refer to the persistent storage as a **two-buffer architecture**.

5.2 Input Representation: Multi-Hop Topology Encoding

We use the SIGN paradigm [36] and precompute the multi-hop feature stack $\{X^{(0)}, X^{(1)}, \ldots, X^{(S)}\}$. This decouples topological propagation from task specific gradient updates: the topology of the graph remains fixed among all tasks while only the parameters at the downstream end are updated in continual training. The precomputed stacks are the only input representation used across the framework, and are linear in the number of nodes, irrespective of the number of hops.

5.3 The CAM Block: Two-Buffer Associative Memory

5.3.1 Persistent Two-Buffer Architecture

There are two persistent components in each memory cell:

$$M_{\text{base}} \in \mathbb{R}^{d_o \times d_i}, \tag{5.1}$$

$$M_{\text{task}} \in \mathbb{R}^{d_o \times d_i}. \tag{5.2}$$

M_{base} is a learnable `nn.Parameter` updated by backpropagation across all task. M_{task} is a registered buffer that is updated once per task by the delta-rule consolidation step (Section 5.6.3). At the start of each forward pass, M_{base} is expanded over the batch and is used as the starting point for the in-context delta-rule. M_{task} is read separately in the task-memory path (Section 5.3.4).

5.3.2 In-Context Delta-Rule Update

Given the initial memory M_{base}, the transient in-context state M_{new} is produced by one step of the DGD update (Equation (3.4)) applied to the associative memory objective:

$$M_{\text{new}} \;=\; \alpha\, M_{\text{base}} \;-\; \eta \underbrace{\big(M_{\text{base}}\mathbf{k} - \hat{\mathbf{v}}\big)\mathbf{k}^\top}_{\text{residual write}} \;-\; \eta\, M_{\text{base}}\, \mathbf{k}\mathbf{k}^\top. \tag{5.3}$$

This state M_{new} is used only for reading during the current forward pass and is not persisted.

5.3.3 Five-Cell Head Design

A single head of the CAM Block contains five cells, each a complete two-buffer associative memory ($M_{\text{base}} + M_{\text{task}}$). Instead of linear projections, the memory learns to produce It dynamically updates its own keys, values and hyper parameters from the initial memory. The per-token learning rate η_t and retention rate α_t are first computed from the *pre-update* memory state, then refined after self-modification of the η and α cells. Table 5.1 summarises the functional mapping of these cells.

Table 5.1: Functional mapping of the 5-cell head design in the CAM block

Cell	Notation	Generated Output (Dynamic Projection)
Key cell	M^k	Context-dependent key: $\mathbf{k}_t = M^k_{\text{base}}\,\mathbf{x}_t$
Value cell	M^v	Context-dependent value: $\mathbf{v}_t = M^v_{\text{base}}\,\mathbf{x}_t$
η-cell	M^η	Per-token learning rate: $\eta_t = \eta_{\max} \cdot \sigma(\bar{e}_t)$
α-cell	M^α	Per-token retention: $\alpha_t = \alpha_{\min} + (\alpha_{\max} - \alpha_{\min}) \cdot \sigma(\bar{a}_t)$
Main cell	M^{main}	Primary associative store, modified by Equation (3.5)

5.3.4 Two-Path Confidence-Adaptive Read

We break the retrieval into two parallel tracks to isolate noisy in-context evidence from combined task memory. M_{new} are read using the learned query $\mathbf{q}_\ell$ and M_{base} and M_{task} are read separately using the prototype-stabilised query:

$$\mathbf{o}_{\text{ic}} = M_{\text{new}}\,\mathbf{q}_\ell\,/\sqrt{d_i}, \qquad \mathbf{o}_{\text{base}} = M_{\text{base}}\,\mathbf{q}_{\text{proto}}\,/\sqrt{d_i}, \qquad \mathbf{o}_{\text{task\_mem}} = M_{\text{task}}\,\mathbf{q}_{\text{proto}}\,/\sqrt{d_i}$$

These are mixed together through a scalar gate α_{fast}, which is calculated from the signal strength of M_{task}:

$$\alpha_{\text{fast}} = \tanh\Big(\text{RMS}(M_{\text{task}})\Big), \qquad \mathbf{o}_{\text{task}} = \mathbf{o}_{\text{base}} + \alpha_{\text{fast}}\,\mathbf{o}_{\text{task\_mem}}. \tag{5.4}$$

Each path has a level of trust associated with it that varies according to its output confidence

$$\text{conf}(\mathbf{o}) = 1 - \frac{\mathcal{H}\big(\text{softmax}(\mathbf{o})\big)}{\log d_o} \;\in [0,1], \tag{5.5}$$

$$\mathbf{o} = c_{\text{ic}}\,\mathbf{o}_{\text{ic}} + (1 - c_{\text{ic}})\,\mathbf{o}_{\text{task}}, \tag{5.6}$$

where $c_{\text{ic}} = \text{conf}(\mathbf{o}_{\text{ic}})$ and $\mathcal{H}(\cdot)$ is Shannon entropy normalised by $\log d_o$.

5.3.5 Prototype-Guided Query

When backbone parameters are updated from one task to another, the learned query $\mathbf{q}_\ell = W_q\mathbf{h}$ drifts in representation space. Since M_{task} stores the associations written under previous calls parameter states, querying it with a drifted $\mathbf{q}_\ell$ causes an *address-drift* misalignment. We resolve this by anchoring retrieval in a dynamic prototype coordinate system. The prototype for class c after each task τ is the uniform mean of all stored anchor representations:

$$\mathbf{p}_c = \frac{1}{|\mathcal{A}_c|}\sum_{n\in\mathcal{A}_c}\mathbf{h}_n. \tag{5.7}$$

This prototype-stabilised query is then:

$$\mathbf{q}_{\text{proto}} = P^\top \text{softmax}\left(\frac{P\mathbf{q}_\ell}{\sqrt{d_h}}\right), \qquad P = [\mathbf{p}_1;\cdots;\mathbf{p}_C] \in \mathbb{R}^{C\times d_h}, \tag{5.8}$$

A weighted convex combination of class prototypes weighted by their similarity. to $\mathbf{q}_\ell$.

5.4 The Learned Titans Gate

We substitute the hard residual with a learned per-dimension sigmoid gate:

$$\mathbf{g} = \sigma(W_g\,\mathbf{h} + \mathbf{b}_g) \;\in\; (0,1)^{B\times(S+1)\times d}, \tag{5.9}$$

and define the fused representation as:

$$\tilde{\mathbf{h}} = \mathrm{LayerNorm}\big(\mathbf{g}\odot\mathbf{t} + (1-\mathbf{g})\odot\mathbf{h}\big). \tag{5.10}$$

5.5 Forward Architecture: End-to-End Pass

Figure 5.2 summarises the six-step forward pass. Each step is described in detail in the sections above.

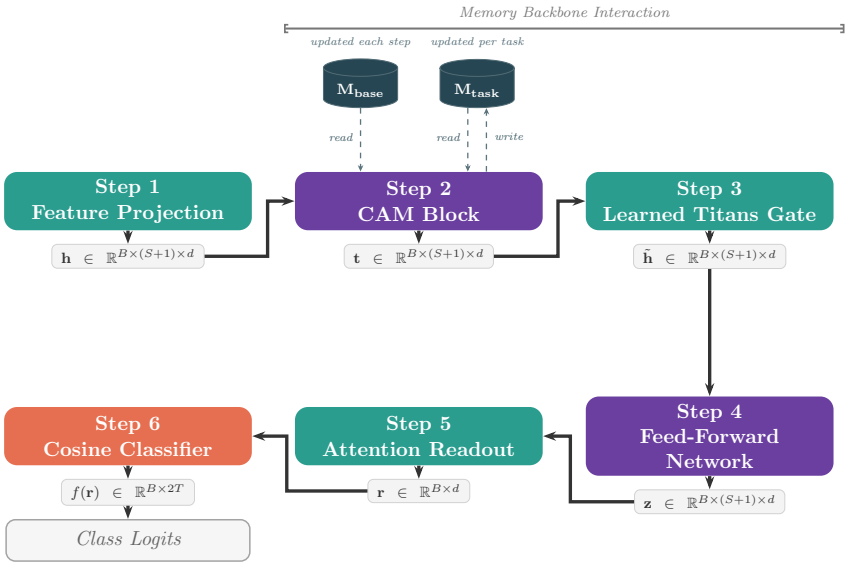


Figure 5.2: Flowchart of the CAM-Titans continual learning pipeline

5.6 The Continual Learning Pipeline

The above are all aspects of the model's *architecture*. We now describe the pipeline optimises the architecture above in three sequential phase per task.

5.6.1 Phase 1: Surprise-Weighted Backbone Training

Current-task loss. The surprise factor of sample n is the square-root-normalised CE loss:

$$\bar{S}_n = \sqrt{\frac{\mathcal{L}_n^{\mathrm{CE}}}{\bar{\mathcal{L}}+\varepsilon}}. \tag{5.11}$$

Up-weighting on high loss samples focuses the gradient budget on both boundary-adjacent nodes, which are most informative for the current task. This is because they share their boundary with a different node type. They are most likely to forget. The current-task loss is:

$$\mathcal{L}_{\text{curr}} = \frac{1}{|B|} \sum_{n \in B} \bar{S}_n \cdot \mathcal{L}_n^{\text{CE}}\big(f_\theta(x_n) + \delta^{\text{curr}},\, y_n\big), \tag{5.12}$$

where $\delta_c^{\text{curr}} = \tau_{\text{scale}} \log \hat{\pi}_c^{\text{curr}}$ is a term for correcting the within-task class frequency in the logit-adjustment method imbalance [41].

Anchor replay loss. Forgetting is represented by two signals: misclassification of old anchors and representation drift even if the argmax is still correct. The combined surprise score is:

$$u_n = \mathcal{L}_n^{\text{CE}} + \|\mathbf{r}_n^{\text{now}} - \mathbf{r}_n^{\text{old}}\|^2, \qquad S_n^{\text{anc}} = \sqrt{\frac{u_n}{\bar{u} + \varepsilon}}. \tag{5.13}$$

$$\mathcal{L}_{\text{anc}} = \frac{1}{|B_a|} \sum_{n \in B_a} S_n^{\text{anc}} \cdot \mathcal{L}_n^{\text{CE}}\big(f_\theta(x_n) + \delta^{\text{mem}},\, y_n\big) + \lambda_{\text{kd}}\, \text{KL}\big(\mathbf{p}_n^{\text{old}} \parallel \mathbf{p}_n^{\text{new}}\big). \tag{5.14}$$

5.6.2 Phase 2: Post-Task Anchor and Prototype Update

After backbone training on task τ class-balanced anchors are selected by greedy centroid matching: for each class c the n_{anc} nodes which are closest to the class centroid in feature space are chosen as anchor nodes. space are retained. The variance is minimised with centroid-matched anchors. The anchor bank was used to reconstruct the prototype. All class prototypes are then recomputed from the full anchor bank via uniform averaging (Equation (5.7)).

5.6.3 Phase 3: Delta-Rule Consolidation of M_{task}

In-context delta-rule steps may have been partially overwritten M_{task} associations from previous tasks after Phase 1. Consolidation is an explicit re-encoding step: the backbone is frozen and M_{task} is optimised to encode the *residual correction* that M_{base} fails to remember.

Residual target. For every anchor i whose key is $\mathbf{k}_i$ and whose class prototype is $\mathbf{p}_{y_i}$, the base-memory prediction is first computed under the assumption is that M_{base} is frozen:

$$\mathbf{r}_i \;=\; \mathbf{p}_{y_i} \;-\; M_{\text{base}}\, \mathbf{k}_i. \tag{5.15}$$

This is what M_{base} is not already aware of. When the class is recalled correctly by M_{base} already, $\mathbf{r}_i \to \mathbf{0}$ and M_{task} receives no gradient for it.

Consolidation objective. M_{task} is trained to predict $\mathbf{r}_i$ using the same anchor keys, with a proximal term anchored to the pre-consolidation state. M_{prev}:

$$\mathcal{L}_{\text{consol}} = \lambda_{\text{kv}} \sum_i \Big\| M_{\text{task}}\, \mathbf{k}_i - \underbrace{\big(\mathbf{p}_{y_i} - M_{\text{base}}\, \mathbf{k}_i\big)}_{\mathbf{r}_i} \Big\|^2 + \lambda_{\text{prox}} \big\| M_{\text{task}} - M_{\text{prev}} \big\|_F^2. \tag{5.16}$$

Only M_{task} buffers are updated; all backbone parameters and M_{base} are not changed during the process.

Chapter 6

Algorithms and Complexity

Algorithm 1 presents the complete CAM-Titans training loop. Algorithms 2 - 4 define the three phase subroutines; each is self-contained and can be read indepen dently.

6.1 Main Training Loop

Algorithm 1 CAM-Titans: Main Training Loop

Require: Task sequence $\mathcal{T}$; hop tensors $\boldsymbol{X}^{(0:S)}$
Ensure: Trained model θ; performance matrix $\mathbf{M}^p$
1: Initialise θ, anchor bank $\mathcal{B} \leftarrow \emptyset$, $M_{\text{task}} \leftarrow 0$
2: **for** each task $\tau \in \mathcal{T}$ **do**
3: Expand cosine classifier by $|\mathcal{Y}_\tau|$ new class rows
4: $\theta \leftarrow \textsc{Phase1}(\theta, \mathcal{D}_\tau, \mathcal{B})$
5: $\mathcal{B}, \{\mathbf{p}_c\} \leftarrow \textsc{Phase2}(\theta, \mathcal{B}, \mathcal{Y}_\tau)$
6: $M_{\text{task}} \leftarrow \textsc{Phase3}(\theta, M_{\text{task}}, \mathcal{B})$
7: $\mathbf{M}^p_{\tau,j} \leftarrow \text{Acc}(f_\theta, \mathcal{D}_j^{\text{val}})$ for $j = 0, \ldots, \tau$
8: **end for**
9: **return** θ, $\mathbf{M}^p$

Complexity. $\mathcal{O}(B \cdot S \cdot d^2)$ per forward pass in Phase 1; $\mathcal{O}(K \cdot n_c \cdot d)$ for Phase 2 anchor selection (no backward pass); $\mathcal{O}(n_{\text{steps}} \cdot B_c \cdot d^2)$ for Phase 3 consolidation. All three phases are linear in batch size and independent of total graph size N beyond the precomputed hop stack.

6.2 Phase 1: Surprise-Weighted Backbone Training

Algorithm 2 Phase 1: Surprise-Weighted Backbone Training

Require: Model θ; task data $\mathcal{D}_\tau$; anchor bank $\mathcal{B}$
Ensure: Updated θ

for each epoch **do**
  Sample mini-batch $(x, y) \sim \mathcal{D}_\tau$
  Compute per-sample CE loss ℓ_n with logit adjustment $\delta_c = \tau_{\text{scale}} \log \hat{\pi}_c$
  Weight each sample by its surprise: $S_n = \sqrt{\ell_n / \bar{\ell}}$
  $\mathcal{L}_{\text{curr}} \leftarrow \frac{1}{|B|} \sum_n S_n \cdot \ell_n$
  if $|\mathcal{B}| > 0$ **then**
    Sample anchor batch $(x_a, h_a, y_a) \sim \mathcal{B}$
    Score anchors by CE loss and representation drift: $u_n = \text{CE}_n + \|r_{\text{new}} - r_{\text{old}}\|^2$
    $\mathcal{L}_{\text{anc}} \leftarrow$ surprise-weighted CE $+ \lambda_{\text{kd}}$ KL$(\mathbf{p}^{\text{old}} \| \mathbf{p}^{\text{new}})$
  end if
  $\theta \leftarrow \theta - \eta \nabla(\mathcal{L}_{\text{curr}} + \mathcal{L}_{\text{anc}})$
end for
return θ

Complexity. $\mathcal{O}(E \cdot (B + B_a) \cdot S \cdot d^2)$ over E epochs, linear in batch size and task data; no graph-size dependence beyond the precomputed hop stack.

6.3 Phase 2: Anchor Bank and Prototype Refresh

Algorithm 3 Phase 2: Anchor Bank and Prototype Refresh

Require: Model θ; anchor bank $\mathcal{B}$; new classes $\mathcal{Y}_\tau$
Ensure: Updated $\mathcal{B}$; class prototypes $\{\mathbf{p}_c\}$

for each class $c \in \mathcal{Y}_\tau$ **do**
  Select n_{sel} anchors by greedy centroid herding: pick nodes whose running mean stays closest to the class centroid
  Append selected anchors to $\mathcal{B}$
end for
for each class c in $\mathcal{B}$ **do**
  $\mathbf{p}_c \leftarrow \frac{1}{|\mathcal{A}_c|} \sum_{n \in \mathcal{A}_c} \mathbf{h}_n$
end for
return $\mathcal{B}$, $\{\mathbf{p}_c\}$

Complexity. $\mathcal{O}(K \cdot n_c \cdot d + |\mathcal{B}| \cdot d)$ for anchor selection and prototype averaging; no backward pass, negligible relative to Phases 1 and 3.

6.4 Phase 3: Delta-Rule Consolidation

Algorithm 4 Phase 3: Delta-Rule Consolidation of M_{task}

Require: Frozen backbone θ; M_{task}; anchor bank $\mathcal{B}$
Ensure: Updated M_{task}
$M_{\text{prev}} \leftarrow M_{\text{task}}$
while not converged **do**
  Sample anchor batch from $\mathcal{B}$
  set $\mathbf{r}_i = \mathbf{p}_{y_i} - M_{\text{base}}\mathbf{k}_i$
  $\mathcal{L}_{\text{consol}} = \lambda_{\text{kv}} \sum_i \|M_{\text{task}}\mathbf{k}_i - \mathbf{r}_i\|^2 + \lambda_{\text{prox}}\|M_{\text{task}} - M_{\text{prev}}\|_F^2$
  $M_{\text{task}} \leftarrow M_{\text{task}} - lr\, \nabla_{M_{\text{task}}} \mathcal{L}_{\text{consol}}$
end while
return M_{task}

Complexity. $\mathcal{O}(n_{\text{steps}} \cdot B_c \cdot d^2)$; independent of graph size since it operates only on the stored anchor bank.

6.5 Space Complexity

Table 6.1 summarises the memory footprint of each persistent component in CAM-Titans.

Table 6.1: Space complexity of CAM-Titans components. d: model dimension; H: number of heads; τ: tasks seen so far; K: classes per task; n_{anc}: anchors per class.

Component	Space	Notes
M_{base} (per head)	$\mathcal{O}(d^2)$	Learnable `nn.Parameter`; fixed size, does not grow with tasks.
M_{task} (per head)	$\mathcal{O}(d^2)$	Registered buffer; fixed size, overwritten each consolidation.
Five-cell head ($\times H$)	$\mathcal{O}(H \cdot d^2)$	Each of the 5 cells holds one M_{base} and one M_{task}.
Anchor bank $\mathcal{B}$	$\mathcal{O}(\tau \cdot K \cdot n_{\text{anc}} \cdot d)$	Grows linearly with tasks; bounded in practice by fixed n_{anc}.
Prototype matrix P	$\mathcal{O}(\tau \cdot K \cdot d)$	One vector per class; recomputed after every task.
Hop-feature stack	$\mathcal{O}(N \cdot S \cdot d)$	Precomputed once; read-only during training.

Chapter 7

Experimental Results

This chapter provides a thorough empirical assessment of CAM-Titans on the Continual Graph Learning Benchmark (CGLB) [8]. We begin with the experimental setup (datasets, baselines, implementation details and evaluation protocol), then report main results across all four evaluation. configurations .The results of ablation, which require additional runs, are deferred to Section 7.2.5.

7.1 Experimental Setup

7.1.1 Datasets

All experiments are performed on the four Node-level Continual Graph Learning. The datasets used in this work are from (N-CGL) provided by CGLB [8]. Table 7.1 summarises their statistics. The data sets cover three application areas. (academic citation, social media and e-commerce) and two orders of magnitude in scale from 19K nodes (CoraFull-CL) to 2.4M nodes (Products-CL).

Table 7.1: Statistics of the four CGLB N-CGL datasets used in this work. All The default task construction is 2-class-per-task ($K = 2$), which is followed by experiments.

Dataset	#Nodes	#Edges	#Classes	#Tasks	#Features	Domain
CoraFull-CL	19,793	130,622	70	35	8,710	Citation
Arxiv-CL	169,343	1,166,243	40	20	128	Citation
Reddit-CL	227,853	114,615,892	40	20	602	Social
Products-CL	2,449,028	61,859,036	46	23	100	E-commerce

Task construction. The C node classes are divided into $\lfloor C/2 \rfloor$ tasks of two classes each, according to the CGLB protocol. At task τ only the nodes belonging to the

two classes of that task are available for training, rest are inaccessible. Nodes from different tasks are linked by *inter-task edges* in the original graph; we consider both with and without to separate the effect of cross-task topological signal on these edges.

7.1.2 Baselines

We compare CAM-Titans with the following eight baselines and one idealised upper bound, covering the major classes of continual learning strategies.

Note that CaT and PUMA are only assessed in the Class-IL setting. For the Task-IL evaluations, the best baselines are MAS and ER-GNN.

Table 7.2: Baseline methods with citations and CL strategy.

Method	Strategy	Description
Bare model	—	Standard GNN/MLP trained sequentially with no anti-forgetting mechanism; lower bound.
EWC [12]	Regularisation	Elastic Weight Consolidation: penalises changes to parameters that were important for previous tasks using the Fisher information matrix.
MAS [13]	Regularisation	Memory Aware Synapses: estimates parameter importance from the sensitivity of outputs to perturbations; no stored data required.
GEM [16]	Replay	Gradient Episodic Memory: constrains the gradient direction so that old-task losses cannot increase.
TWP [7]	Regularisation	Topology-aware Weight Preservation: extends EWC with graph-topology-based importance weights.
LwF [18]	Distillation	Learning without Forgetting: uses knowledge distillation from a frozen snapshot of the model to prevent forgetting; stores no exemplars.
ER-GNN [26]	Replay	Experience Replay adapted for GNNs: randomly samples exemplars from previous tasks and jointly optimises on current and replayed data.
CaT [29]	Replay	Condense and Train: condenses past task data into a small synthetic dataset and replays it.
PUMA [30]	Replay + Distillation	State-of-the-art graph CL method combining prototype-based memory with knowledge distillation.
Joint	—	Train on all tasks simultaneously; *upper bound* with no forgetting by construction. Not directly comparable.

7.1.3 Implementation Details

Framework and hardware. All experiments are carried out using PyTorch [42] with PyTorch Geometric [43]. Hop-feature precomputation uses torch.sparse.mm (GPU) with tensors transferred to CPU to prevent VRAM overflow on large graphs. Experiments were run on a single NVIDIA GPU.

Reproducibility. Each configuration is run across five seeds $\{42, 43, 44, 45, 46\}$. All The mean $\pm$ standard deviation of these five runs is reported as results.

Dataset-specific hyperparameters. The complete set of hyperparameters for each is listed in Table 7.3. dataset. Common parameters for all datasets: number of hops $S = 3$, number of attention heads $H = 8$, dropout rate 0.1, learning rate 2×10^{-4}, weight decay 10^{-4}, epochs per task 100, and absorption frequency $K = 5$.

Table 7.3: Dataset-specific hyperparameters for the four CGLB benchmarks. Parameters not listed here are shared: S=3, H=8, dropout= 0.1, lr=2×10^{-4}, wd=10^{-4}, epochs/task= 100

Hyperparameter	CoraFull-CL	Arxiv-CL	Reddit-CL	Products-CL
d (model dimension)	128	384	256	384
σ_0 (initial cosine temp.)	15.0	10.0	15.0	10.0
f_{anc} (anchor fraction)	0.1	0.01	0.01	0.01
Train batch size	256	512	512	512
n_{steps} (consol.)	30	80	30	40
Consolidation lr	5×10^{-4}	3×10^{-4}	5×10^{-4}	3×10^{-4}
λ_{prox}	0.15	0.08	0.15	0.15
λ_{kv}	1.0	1.5	1.0	1.0
λ_{kd}	1.0	1.0	5.0	1.0
τ_{scale}	1.0	1.5	0.5	1.0

7.1.4 Evaluation Protocol

We test the four CGLB configurations (N-CGL) by combining two of the following:

- **Learning scenario:** Task-IL (model receives a task indicator at test time) vs. Class-IL (no task indicator; model must discriminate for all classes observed).
- **Topology:** Without inter-task edges (task subgraphs are disconnected) vs. with inter-task edges (the original graph connectivity is preserved).

The Class-IL setting without inter-task edges is the most challenging and it was used as the main assessment method.

Metrics. Following CGLB [8] we report:

- **Average Performance** (AP, %) : mean test accuracy over all Tasks after the entire task sequence has been processed. $\mathrm{AP}_T = \frac{1}{T}\sum_{j=1}^{T} M^p_{T,j}$.
- **Average Forgetting** (AF, %) = mean decrease in per-task accuracy relative to the peak within the task. $\mathrm{AF}_T = \frac{1}{T-1}\sum_{j=1}^{T-1}(M^p_{T,j} - M^p_{j,j})$. Negative AF means forgetfulness, close to zero means stability.

AP is the main indicator; AF is a secondary indicator of how much the The model fails to perform well on the previously learned tasks.

7.2 Main Results

7.2.1 Task-IL Without Inter-Task Edges

The results in the Task-IL setting without the use of edges are reported in Table 7.4. inter-task edges. The task index is fed into the model at test time, so it only needs to discriminate among the two classes of the queried The challenge is only one of *retaining* per-task decision boundaries, task; Throughout the task sequence.

Table 7.4: Performance under Task-IL without inter-task edges (↑ higher is better). Bold: best among all methods. Ours: CAM-Titans.

Method	CoraFull-CL		Arxiv-CL		Reddit-CL		Products-CL	
	AP/%↑	AF/%↑	AP/%↑	AF/%↑	AP/%↑	AF/%↑	AP/%↑	AF/%↑
Bare model	58.0 ± 1.7	-38.4 ± 1.8	61.7 ± 3.8	-28.2 ± 3.3	73.6 ± 3.5	-26.9 ± 3.5	67.6 ± 1.6	-25.4 ± 1.6
EWC [12]	78.9 ± 2.4	-15.5 ± 2.3	78.8 ± 2.7	-5.0 ± 3.1	91.5 ± 4.2	-8.1 ± 4.6	90.1 ± 0.3	-0.7 ± 0.3
MAS [13]	93.0 ± 0.5	-0.6 ± 0.2	88.4 ± 0.6	-0.0 ± 0.2	98.6 ± 0.5	-0.1 ± 0.1	91.2 ± 0.6	$\mathbf{-0.5 \pm 0.2}$
GEM [16]	91.6 ± 0.6	-1.8 ± 0.6	87.3 ± 0.6	2.8 ± 0.3	91.6 ± 0.5	-8.1 ± 5.8	87.5 ± 0.5	-2.9 ± 0.5
TWP [7]	92.2 ± 0.5	-0.9 ± 0.8	86.0 ± 0.8	-2.8 ± 0.8	87.4 ± 3.8	-12.6 ± 4.0	90.3 ± 0.1	-0.5 ± 0.1
LwF [18]	56.1 ± 2.0	-37.5 ± 1.8	84.2 ± 0.5	-3.7 ± 0.6	80.9 ± 4.3	-19.1 ± 4.4	66.5 ± 2.2	-26.3 ± 2.3
ER-GNN [26]	90.6 ± 0.1	-3.7 ± 0.1	86.7 ± 0.1	$\mathbf{11.4 \pm 0.9}$	98.9 ± 0.0	-0.1 ± 0.1	89.0 ± 0.4	-2.5 ± 0.3
Ours	$\mathbf{94.51 \pm 0.40}$	$\mathbf{-0.64 \pm 0.17}$	$\mathbf{92.68 \pm 0.31}$	-1.70 ± 0.29	$\mathbf{99.43 \pm 0.04}$	$\mathbf{-0.13 \pm 0.03}$	$\mathbf{92.48 \pm 0.34}$	-2.04 ± 0.17
Joint (upper bound)	95.2 ± 0.2	—	90.3 ± 0.2	—	99.4 ± 0.1	—	91.8 ± 0.2	—

Observations: Observations (Task-IL, no inter-task edges). In this setting, the highest AP is obtained by CAM-Titans on all four datasets. It gains over the strongest baseline (MAS) range from 1.5 pp on CoraFull-CL to 4.3 pp on Arxiv-CL, and forgetting is close to zero on CoraFull-CL and Reddit-CL ($|\mathrm{AF}| < 0.7\%$). One exception is Products-CL, where AF (-2.04%) is larger than MAS (-0.5%).

7.2.2 Task-IL With Inter-Task Edges

Table 7.5 shows the results when inter-task edges are preserved. This is a more realistic setting: the original graph's cross-task connectivity is intact: Each node in a task is influenced during the precomputation of the hop-feature. By nodes from all other tasks.

Table 7.5: Performance under Task-IL with inter-task edges (↑ higher is better). Bold: best among all methods.

Method	CoraFull-CL		Arxiv-CL		Reddit-CL		Products-CL	
	AP/%↑	AF/%↑	AP/%↑	AF/%↑	AP/%↑	AF/%↑	AP/%↑	AF/%↑
Bare model	60.9 ± 2.1	-35.7 ± 2.1	57.7 ± 1.2	-31.2 ± 1.0	67.4 ± 5.5	-22.9 ± 6.1	60.4 ± 1.5	-31.5 ± 1.5
EWC [12]	73.9 ± 2.6	-21.9 ± 2.6	67.6 ± 3.2	-17.8 ± 3.0	92.6 ± 2.2	-6.8 ± 2.4	73.6 ± 1.3	-25.2 ± 2.3
MAS [13]	93.6 ± 0.6	-0.9 ± 0.2	88.5 ± 0.4	0.1 ± 0.1	95.9 ± 0.5	-2.2 ± 0.4	78.3 ± 1.6	-0.2 ± 0.8
GEM [16]	91.8 ± 0.7	-3.1 ± 0.6	78.7 ± 1.2	-4.5 ± 0.6	77.1 ± 11.1	-23.2 ± 11.7	80.6 ± 0.9	-9.0 ± 1.0
TWP [7]	90.3 ± 1.1	-4.1 ± 1.0	87.1 ± 0.7	-1.2 ± 0.8	91.2 ± 1.8	-8.1 ± 1.9	88.2 ± 0.8	-1.8 ± 1.0
LwF [18]	61.3 ± 2.3	-34.9 ± 2.3	83.8 ± 1.1	-5.8 ± 1.1	81.8 ± 7.5	-18.1 ± 2.8	71.4 ± 2.0	-26.1 ± 1.9
ER-GNN [26]	90.9 ± 0.7	-4.8 ± 0.6	87.2 ± 0.2	$\mathbf{11.8 \pm 0.6}$	97.0 ± 0.3	-2.2 ± 0.3	88.9 ± 3.2	$\mathbf{0.6 \pm 0.8}$
Ours	$\mathbf{95.09 \pm 0.25}$	$\mathbf{-0.68 \pm 0.22}$	$\mathbf{92.95 \pm 0.36}$	-1.59 ± 0.38	$\mathbf{99.22 \pm 0.07}$	$\mathbf{-0.35 \pm 0.06}$	$\mathbf{92.57 \pm 0.26}$	-2.47 ± 0.27
Joint (upper bound)	95.4 ± 0.2	—	89.4 ± 0.3	—	98.7 ± 0.1	—	87.6 ± 0.5	—

Observations : The inclusion of inter-task edges has a very small impact on CAM-Titans: AP improves slightly on CoraFull-CL (95.09 vs. 94.51%) and Arxiv-CL (92.95 vs. 92.68%), indicating that cross-task neighbourhood information is absorbed beneficially during Hop-feature precomputation. The best AP for all four datasets is set by CAM-Titans in this settings.

7.2.3 Class-IL Without Inter-Task Edges

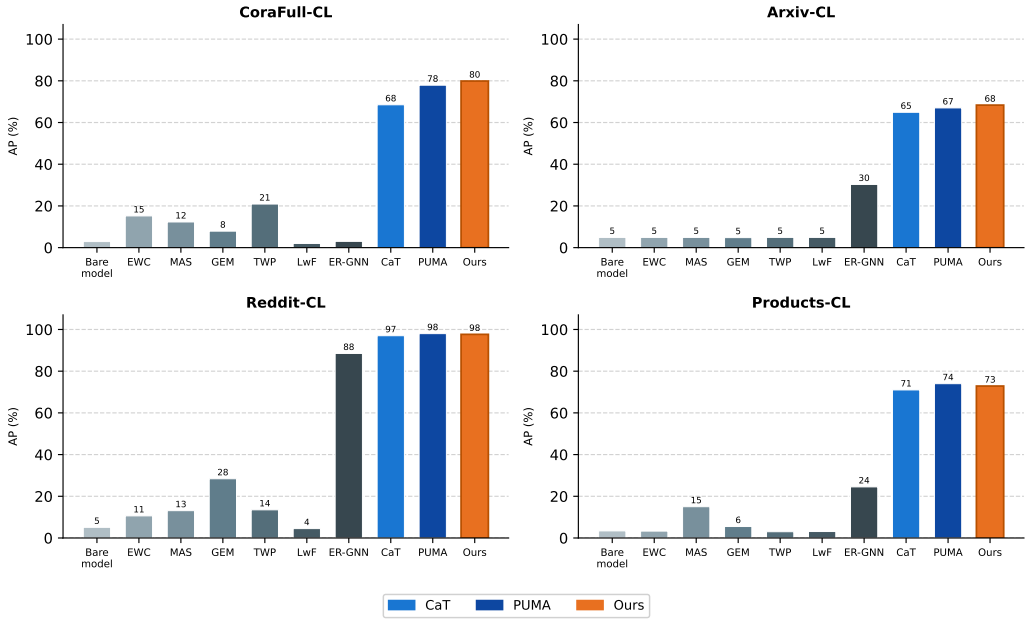
Table 7.6 reports results in the most severe Class-IL setting, the most challenging configuration evaluated. The model should not have a task indicator if it is not. learn to discriminate all classes seen so far together. This requires not only retaining per-task information but also *integrating* it into a A consistent multi-classifier head which expands as new tasks are added.

Table 7.6: Performance under Class-IL without inter-task edges (↑ higher is better). Bold: best among all method

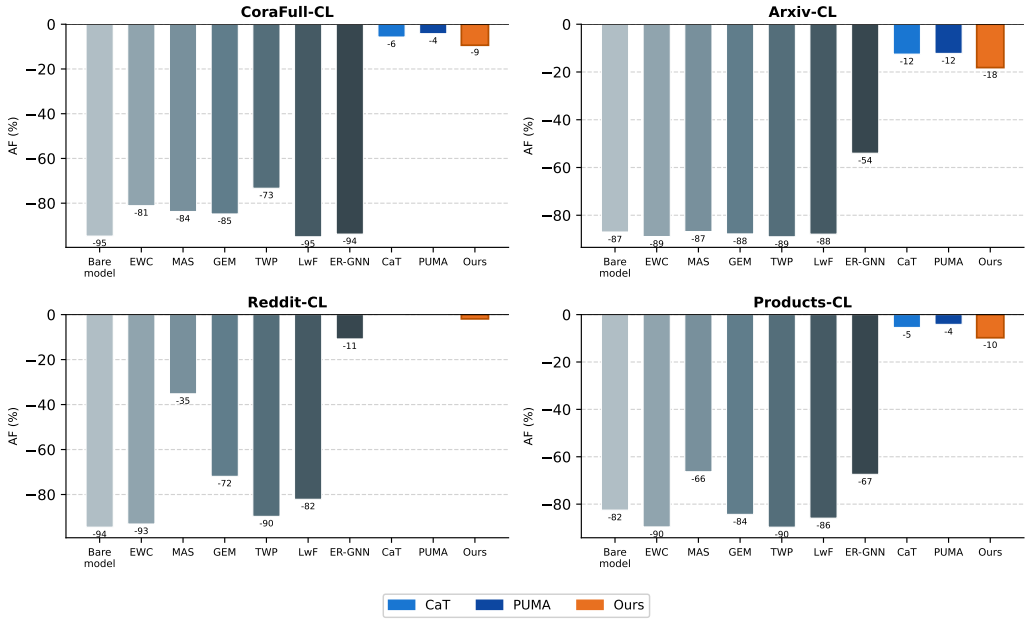
Method	CoraFull-CL		Arxiv-CL		Reddit-CL		Products-CL	
	AP/%↑	AF/%↑	AP/%↑	AF/%↑	AP/%↑	AF/%↑	AP/%↑	AF/%↑
Bare model	2.9 ± 0.0	-94.7 ± 0.1	4.9 ± 0.0	-87.0 ± 1.5	5.1 ± 0.3	-94.5 ± 2.5	3.4 ± 0.8	-82.5 ± 0.8
EWC [12]	15.2 ± 0.7	-81.1 ± 1.0	4.9 ± 0.0	-88.9 ± 0.3	10.6 ± 1.5	-93.1 ± 2.6	3.3 ± 1.2	-89.6 ± 2.0
MAS [13]	12.3 ± 3.8	-83.7 ± 4.1	4.9 ± 0.0	-86.8 ± 0.6	13.1 ± 2.6	-35.2 ± 3.5	15.0 ± 2.1	-66.3 ± 1.5
GEM [16]	7.9 ± 2.7	-84.8 ± 2.7	4.8 ± 0.5	-87.8 ± 0.2	28.4 ± 3.5	-71.9 ± 4.2	5.5 ± 0.7	-84.3 ± 0.9
TWP [7]	20.9 ± 3.8	-73.3 ± 4.1	4.9 ± 0.0	-89.0 ± 0.4	13.5 ± 2.6	-89.7 ± 2.7	3.0 ± 0.7	-89.7 ± 1.0
LwF [18]	2.0 ± 0.2	-95.0 ± 0.2	4.9 ± 0.0	-87.9 ± 1.0	4.5 ± 0.5	-82.1 ± 1.0	3.1 ± 0.8	-85.9 ± 1.4
ER-GNN [26]	3.0 ± 0.1	-93.8 ± 0.5	30.3 ± 1.5	-54.0 ± 1.3	88.5 ± 2.3	-10.8 ± 2.4	24.5 ± 1.9	-67.4 ± 1.9
CaT [29]	68.5 ± 0.9	-5.7 ± 1.3	64.9 ± 0.3	-12.5 ± 0.8	97.0 ± 0.1	-0.4 ± 0.1	71.0 ± 3.1	-5.4 ± 0.3
PUMA [30]	77.9 ± 0.2	$\mathbf{-4.2 \pm 0.9}$	67.0 ± 0.1	$\mathbf{-12.2 \pm 0.3}$	$\mathbf{98.0 \pm 0.1}$	$\mathbf{-0.3 \pm 0.1}$	$\mathbf{74.0 \pm 0.4}$	$\mathbf{-4.1 \pm 0.5}$
Ours	$\mathbf{79.89 \pm 0.16}$	-9.39 ± 0.3	$\mathbf{68.35 \pm 0.37}$	-18.15 ± 0.34	97.65 ± 0.02	-1.89 ± 0.03	72.87 ± 0.22	-9.76 ± 0.41
Joint (upper bound)	80.6 ± 0.3	—	46.4 ± 1.4	—	99.3 ± 0.2	—	71.5 ± 0.7	—

Observations : CAM-Titans leads on CoraFull-CL (79.89%) and Arxiv-CL (68.35%), but is 0.35% and 1.13% less than PUMA on Reddit-CL and Products-CL respectively. In addition, forgetting is greater than PUMA in all datasets (−9.4% vs. −4.2% on

CoraFull-CL), reflecting a limitation of the consolidation strategy when the classification head expands and inter-task interference increases with the size and diversity of the graphs.



(a) Average Performance (AP %) across all methods and datasets.

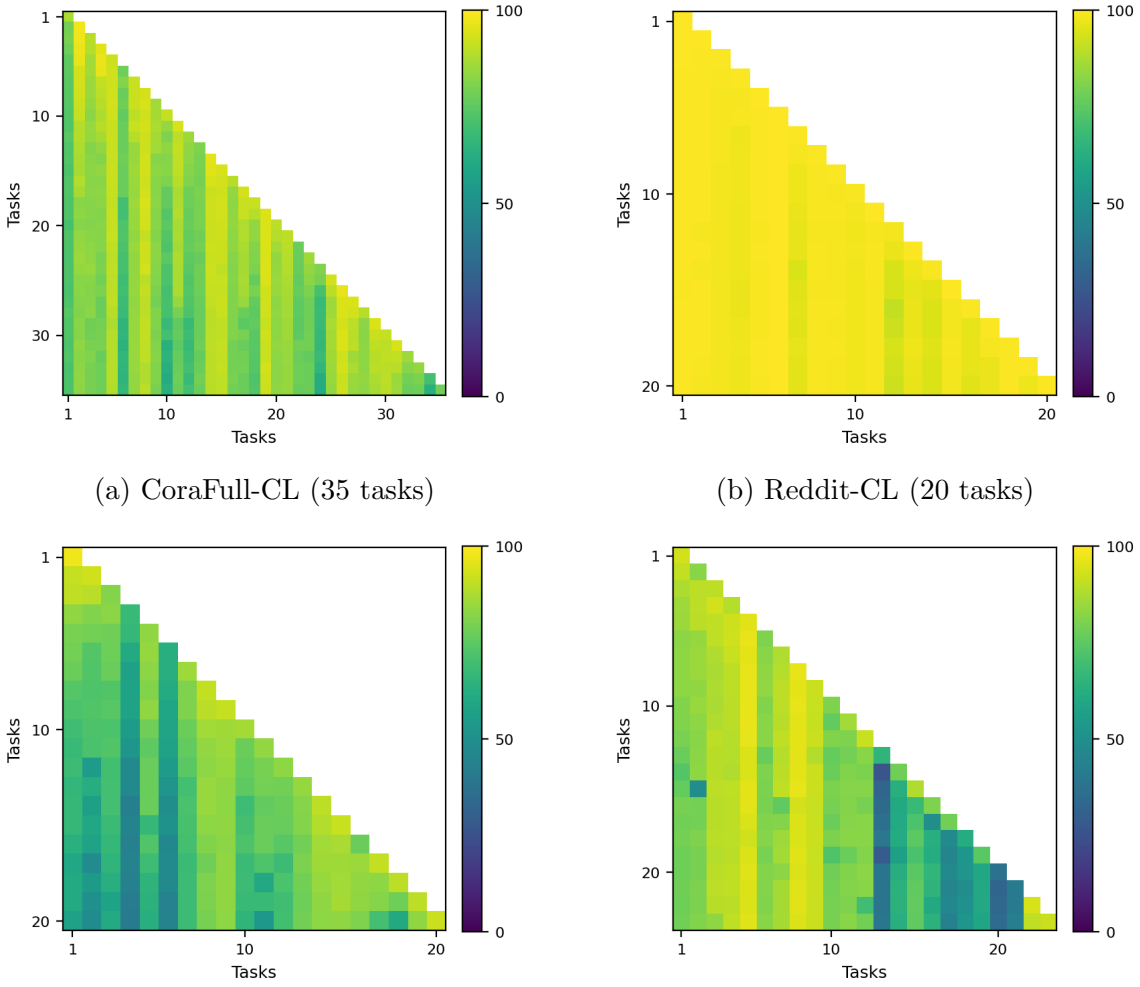


(b) Average Forgetting (AF %) across all methods and datasets.

Figure 7.1: Visual comparison of AP and AF under Class-IL without inter-task edges

7.2.4 Performance Evolution Across Tasks

To complement the final-accuracy numbers we examine two complementary views: the *per-task accuracy matrix* $M^p_{t,j}$ (performance on task j evaluated after task t has been learned) and the running AP / AF curves that trace how average metrics evolve as tasks arrive.



(a) CoraFull-CL (35 tasks)

(b) Reddit-CL (20 tasks)

(c) Arxiv-CL (20 tasks)

(d) Products-CL (23 tasks)

Figure 7.2: Per-task accuracy matrices $M^p_{t,j}$ under Class-IL without inter-task edges (mean over 5 seeds). Yellow $\approx 100\%$ (retained); blue $\approx 0\%$ (forgotten).

Observations : Reddit-CL is stable all the way through with AP above 97% and AF never below -2% as it has near zero final forgetting. The AP drop is relatively small for CoraFull-CL and Arxiv-CL after task 10, with the accuracy matrices showing that some structurally similar tasks are more difficult to remember. The greatest decline

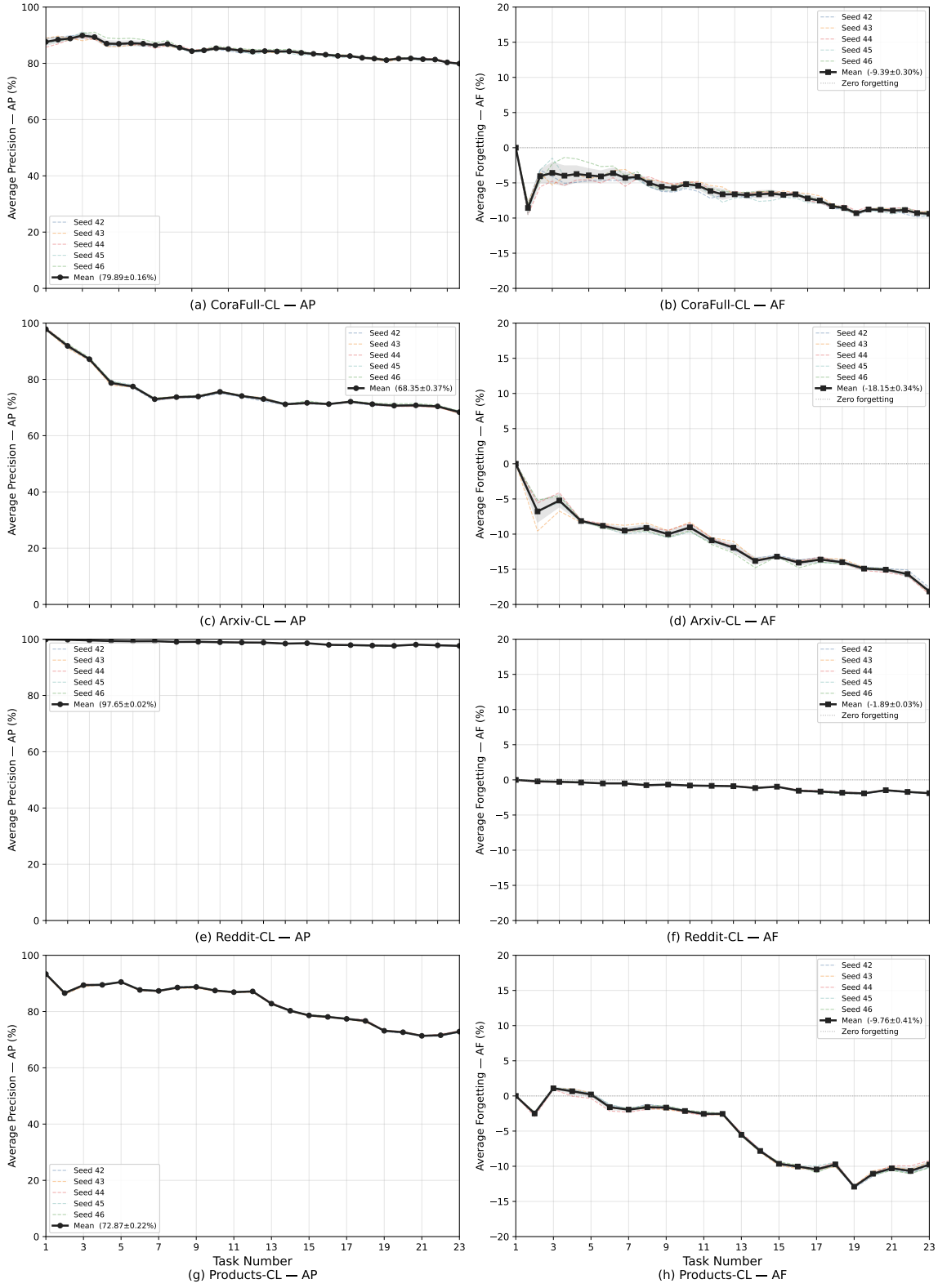


Figure 7.3: Running AP (left) and AF (right) curves under Class-IL without inter-task edges. Mean over 5 seeds; dashed lines show individual runs.

is seen in Products-CL around task 13, possibly because of semantically confusable product categories, though both metrics partially recover in subsequent tasks. In all data sets, the curves for the five seeds are almost identical, indicating stable training.

7.2.5 Ablation Study

We are ablating four components of CAM-Titans on Arxiv-CL (Class-IL, no inter-task). edges) over five seeds $\{42, 43, 44, 45, 46\}$. Every variant eliminates one of the components at a time; everything else is unchanged.

Table 7.7: Ablation results on Arxiv-CL (Class-IL, 5 seeds) Bold: best in column.

Variant	AP (%) ↑	AF (%) ↑
Full CAM-Titans	$\mathbf{68.35 \pm 0.37}$	$\mathbf{-18.15 \pm 0.34}$
w/o Cosine Classifier	65.72 ± 0.75	-22.23 ± 1.65
w/o Prototype Query	63.34 ± 0.47	-26.32 ± 0.54
w/o M_{task} (single memory)	63.40 ± 0.51	-27.49 ± 0.49
w/o Anchor Replay	4.94 ± 0.01	-68.50 ± 1.93

Table 7.7 shows that all of CAM-Titans' core mechanisms target unique axes of catastrophic forgetting in the strictest setting (Class-IL) on Arxiv-CL. As theoretically predicted, anchor replay is the first line of defense, and without it the network becomes subject to unlimited structural change, leading to an average performance (AP) of 4.94%, which is the random guessing level. Removing the M_{task} buffer collapses the nested optimization hierarchy, causing transient updates to destructively update the slow-weight base memory, and reducing AP to 63.40%. The penalty for disabling the prototype-guided query is almost the same (63.34% AP), as the address-drift issue is reintroduced, and the model misaddresses the associations at the task level as backbone representations change. Last but not least, the cosine classifier adds the most variance to forgetting ($-22.23 \pm 1.65\%$ AF); its absence leads to magnitude imbalance, where older classes take over gradient updates and overwhelmingly dominate the output logits. Finally, these results empirically confirm that a unified approach towards structural context, multi-timescale optimization, representation stability, and magnitude normalization is necessary to solve catastrophic forgetting on graphs.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This dissertation presented a continuous graph learning framework called CAM-Titans based on the Nested Learning principle of multi-frequency associative memory. The main argument was that catastrophic forgetting on graphs occurs from two failure modes: parametric overwriting and structural address-drift and both can be solved in one single unified architecture that partitions memory into parts that run at different speeds.

The architectural contribution is a two-buffer associative memory block embedded in a five-cell self-referential head. M_{base} slowly builds up cross-task structural knowledge through standard backpropagation and M_{task} encodes task-level residual correction via delta-rule consolidation after every task. The in-context delta-rule state M_{new} is third, transient level adaptation within each forward pass.

Two further contributions address known failure modes in Class-IL graph learning. The prototype-guided query overcomes address-drift misalignment by anchoring the task-memory read in a dynamically updated prototype coordinate system, ensuring that M_{task} associations written under past parameter states remain retrievable as the backbone evolves. The cosine classifier mitigates magnitude imbalance across the growing class head by normalise both weight and vectors and feature representations.

Empirically, CAM-Titans was tested with twelve configurations spanning four large-scale CGLB benchmarks, two learning scenarios (Task-IL and Class-IL), and two topology conditions (with and without inter-task edges). For the Task-IL setting, CAM-Titans performs best on all four datasets surpassing the strongest baseline (MAS) by up to 4.3 pp on Arxiv-CL while maintaining near-zero forgetting. In the more challenging Class-IL setting, CAM-Titans leads on CoraFull-CL (79.89%) and Arxiv-CL (68.35%). The ablation study confirms that all four components are non-redundant: anchor replay gives the foundational anti-forgetting signal, while the cosine classifier, prototype-guided query, Both and M_{task} buffer provide different and measurable improvements in both AP and AF.

8.2 Limitations

Although CAM-Titans is empirically successful, it has a number of structural and computational drawbacks. The main drawback of the framework is that it has higher forgetting rates in complex Class-IL settings on large graphs (e.g., Products-CL) because of the fixed consolidation budget and the static anchor bank. As the classification head grows and inter-task interference rises, the proximal regulariser is unable to completely stabilise M_{task}. This is compounded by the greedy centroid herding strategy which ignores feature drift over long sequences of tasks, so that anchors from early tasks are poor representatives and reduce the residual target $\mathbf{r}_i$ during Phase 3 consolidation. Moreover, the architecture has high overhead: when the model size and number of heads increase, 5 full associative memory cells are required. Lastly, the use of a precomputed multi-hop feature stack only allows CAM-Titans to be used in transductive settings, where no new nodes are seen at test time.

8.3 Future Work

There are a number of promising avenues to overcome these bottlenecks. Future versions could include an online anchor refresh, which periodically re-embeds stored features through the frozen hop stack and current encoder, and an adaptive consolidation schedule, which dynamically assigns n_{steps} based on the pre-consolidation residual norms ($\|\mathbf{r}_i\|$), to reduce anchor drift and rigid consolidation. Furthermore, the fixed hyperparameter λ_{prox} can be replaced by a learnable regularisation term using a simple bilevel optimisation, which would automatically tune the stability-plasticity balance without any manual tuning. A more promising approach to overcome the transductive constraint is to use on-the-fly neighbourhood sampling (such as GraphSAINT or ClusterGCN) to perform localised, inductive inference on unseen nodes. Finally, the extension of this multi-frequency memory design to other graph-incremental paradigms, where the prototype coordinate system needs to be generalizable across completely different graph domains, beyond class boundaries, is an open and relevant research direction.

References

[1] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[2] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[3] Sebastian Thrun. A lifelong learning perspective for mobile robot control. In *Intelligent robots and systems*, pages 201–214. Elsevier, 1995.

[4] Mark Bishop Ring. *Continual learning in reinforcement environments*. The University of Texas at Austin, 1994.

[5] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165, 1989.

[6] Roger Ratcliff. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, 97(2):285–308, 1990.

[7] Huihui Liu, Yiding Yang, and Xinchao Wang. Overcoming catastrophic forgetting in graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 8653–8661, 2021.

[8] Xikun Zhang, Dongjin Song, and Dacheng Tao. Cglb: Benchmark tasks for continual graph learning. *Advances in Neural Information Processing Systems*, 35:13006–13021, 2022.

[9] Falih Gozi Febrinanto, Feng Xia, Kristen Moore, Chandra Thapa, and Charu Aggarwal. Graph lifelong learning: A survey. *IEEE Computational Intelligence Magazine*, 18(1):32–51, 2023.

[10] Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. Nested learning: The illusion of deep learning architectures. *Advances in Neural Information Processing Systems*, 38:46968–47002, 2026.

[11] Gido M Van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[12] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[13] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European conference on computer vision (ECCV)*, pages 139–154, 2018.

[14] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International conference on machine learning*, pages 3987–3995. Pmlr, 2017.

[15] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

[16] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

[17] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *Advances in neural information processing systems*, 33:15920–15930, 2020.

[18] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[19] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[20] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[21] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International conference on machine learning*, pages 4548–4557. PMLR, 2018.

[22] Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 139–149, 2022.

[23] Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, et al. Dualprompt: Complementary prompting for rehearsal-free continual learning. In *European conference on computer vision*, pages 631–648. Springer, 2022.

[24] Yishi Xu, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, and Mark Coates. Graphsail: Graph structure aware incremental learning for recommender systems. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2861–2868, 2020.

[25] Junwei Su, Difan Zou, Zijun Zhang, and Chuan Wu. Towards robust graph incremental learning on evolving graphs. In *International Conference on Machine Learning*, pages 32728–32748. PMLR, 2023.

[26] Fan Zhou and Chengtai Cao. Overcoming catastrophic forgetting in graph neural networks with experience replay. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 4714–4722, 2021.

[27] Xikun Zhang, Dongjin Song, and Dacheng Tao. Sparsified subgraph memory for continual graph representation learning. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 1335–1340. IEEE, 2022.

[28] Junwei Su, Difan Zou, and Chuan Wu. On the limitation and experience replay for gnns in continual learning. *arXiv preprint arXiv:2302.03534*, 2023.

[29] Yilun Liu, Ruihong Qiu, and Zi Huang. Cat: Balanced continual graph learning with graph condensation. In *2023 IEEE International Conference on Data Mining (ICDM)*, pages 1157–1162. IEEE, 2023.

[30] Yilun Liu, Ruihong Qiu, Yanran Tang, Hongzhi Yin, and Zi Huang. Puma: Efficient continual graph learning for node classification with graph condensation. *IEEE Transactions on Knowledge and Data Engineering*, 37(1):449–461, 2024.

[31] Xikun Zhang, Dongjin Song, and Dacheng Tao. Hierarchical prototype networks for continual graph representation learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4622–4636, 2022.

[32] Chen Wang, Yuheng Qiu, Dasong Gao, and Sebastian Scherer. Lifelong graph learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13719–13728, 2022.

[33] Chaoxi Niu, Guansong Pang, Ling Chen, and Bing Liu. Replay-and-forget-free graph class-incremental learning: A task profiling and prompting approach. *Advances in Neural Information Processing Systems*, 37:87978–88002, 2024.

[34] Lecheng Kong, Theodore Vasiloudis, Seongjun Yun, Han Xie, and Xiang Song. Dynamic mixture-of-experts for incremental graph learning. *arXiv preprint arXiv:2508.09974*, 2025.

[35] Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. Nagphormer: A tokenized graph transformer for node classification in large graphs. *arXiv preprint arXiv:2206.04910*, 2022.

[36] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

[37] Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

[38] Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning. In *International conference on machine learning*, pages 3469–3489. PMLR, 2022.

[39] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *Advances in Neural Information Processing Systems*, 38:113506–113543, 2026.

[40] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023.

[41] Chaoxi Niu, Guansong Pang, and Ling Chen. Graph continual learning with debiased lossless memory replay. *arXiv preprint arXiv:2404.10984*, 2024.

[42] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[43] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Appendix A

Mathematical Proofs

Definition A.1 (Permutation Equivariance) *Let $f : \mathbb{R}^{d\times B} \to \mathbb{R}^{d\times B}$ be a neural network layer operating on a batch of B items. The function f is permutation equivariant if, for any $B \times B$ permutation matrix P, applying the permutation to the input results in the exact same permutation applied to the output:*

$$f(XP) = f(X)P \tag{A.1}$$

Proposition A.2 (Permutation Equivariance of Parallel Titans) *The Parallel Titans associative memory module, defined by the batch-level delta rule update and linear readout, is strictly permutation equivariant with respect to the node dimension B.*

Proof:

Let $P \in \{0,1\}^{B\times B}$ be an arbitary Permutation matrix. We know Permutation matrix is Orthogonal, i.e. $PP^\top = P^\top P = I_B$, where I_B is the $B \times B$ identity matrix.

For the Parallel Titans formulation, the inputs for a specific batch are the Key matrix, Value matrix, and Query matrix, denoted by $K, \hat{V}, Q \in \mathbb{R}^{d\times B}$ respectively. Multiplying on the right by P will permute the columns (the nodes). Let the permuted inputs be:

$$K_P = KP, \quad \hat{V}_P = \hat{V}P, \quad Q_P = QP. \tag{A.2}$$

Step 1 : Invariance of update rule The memory update delta rule is:

$$M_{\text{new}} = \alpha M_{\text{prev}} - \eta M_{\text{prev}} KK^\top - \eta(M_{\text{prev}}K - \hat{V})K^\top. \tag{A.3}$$

We replace the permuted inputs K_P and $\hat{V}_P$ in the update equation to obtain the memory state $M^{(P)}_{\text{new}}$:

$$M^{(P)}_{\text{new}} = \alpha M_{\text{prev}} - \eta M_{\text{prev}}(KP)(KP)^\top - \eta(M_{\text{prev}}KP - \hat{V}P)(KP)^\top. \tag{A.4}$$

Applying the transpose property $(KP)^\top = P^\top K^\top$ to the terms, we get:

$$M^{(P)}_{\text{new}} = \alpha M_{\text{prev}} - \eta M_{\text{prev}} K(PP^\top)K^\top - \eta(M_{\text{prev}}K - \hat{V})PP^\top K^\top. \tag{A.5}$$

Since $PP^\top = I_B$ the identity matrix can be factored out, leaving:

$$\begin{aligned} M_{\text{new}}^{(P)} &= \alpha M_{\text{prev}} - \eta M_{\text{prev}} K K^\top - \eta (M_{\text{prev}} K - \hat{V}) K^\top \\ &= M_{\text{new}}. \end{aligned} \tag{A.6}$$

This shows that the internal memory matrix M_{new} is strictly *permutation invariant*. The same knowledge is stored in the weight matrix, regardless of the order of presentation of the nodes in the batch.

Step 2 : Equivariance of the output The final result of the module is calculated by reading from the updated memory with the Query matrix:

$$Y = M_{\text{new}} Q. \tag{A.7}$$

If the query matrix $Q_P = QP$ is fed into the permutation-invariant memory state $M_{\text{new}}^{(P)} = M_{\text{new}}$, the output $Y^{(P)}$ is:

$$\begin{aligned} Y^{(P)} &= M_{\text{new}}^{(P)} Q_P \\ &= M_{\text{new}} (QP) \\ &= (M_{\text{new}} Q) P = YP \end{aligned} \tag{A.8}$$

If the input matrix is X and the output is Y then Y is right multiplied by the same permutation matrix P that is applied to the inputs. Hence, $f(XP) = f(X)P$, satisfying the definition of permutation equivariance. This ensures that the Parallel Titans module will not have any arbitrary ordering bias for graph-structured data. □

Proposition A.3 (Geometric Convergence of Phase 3 Consolidation) *The Phase 3 objective is a pure proximal ridge regression over M_{task}:*

$$\mathcal{L}_{\text{consol}}(M_{\text{task}}) = \lambda_{\text{kv}} \sum_{i=1}^{B_c} \left\| M_{\text{task}} \mathbf{k}_i - \mathbf{r}_i \right\|^2 + \lambda_{\text{prox}} \left\| M_{\text{task}} - M_{\text{prev}} \right\|_F^2, \tag{A.9}$$

where the residual target $\mathbf{r}_i = \mathbf{p}_{y_i} - M_{\text{base}} \mathbf{k}_i$ is the error made by M_{base} when it forgets anchor i, and computed once with M_{base} frozen before optimization starts. For all $\lambda_{\text{prox}} > 0$, $\mathcal{L}_{\text{consol}}$ is μ-strongly convex, where $\mu = 2\lambda_{\text{prox}}$ and possesses unique closed-form minimizer :

$$M_{\text{fast}}^* = \left(\lambda_{\text{kv}} R K^\top + \lambda_{\text{prox}} M_{\text{prev}} \right) \left(\lambda_{\text{kv}} K K^\top + \lambda_{\text{prox}} I \right)^{-1}, \tag{A.10}$$

where $K = [\mathbf{k}_1, \ldots, \mathbf{k}_{B_c}] \in \mathbb{R}^{d_i \times B_c}$ and $R = [\mathbf{r}_1, \ldots, \mathbf{r}_{B_c}] \in \mathbb{R}^{d_o \times B_c}$.

Moreover, for learning rate $\eta \leq \frac{1}{L}$, gradient descent converges geometrically :

$$\mathcal{L}_{\text{consol}}(M_t) - \mathcal{L}_{\text{consol}}(M^*) \;\leq\; \left(1 - \frac{2\lambda_{\text{prox}}}{L} \right)^t \left[\mathcal{L}_{\text{consol}}(M_0) - \mathcal{L}_{\text{consol}}(M^*) \right], \tag{A.11}$$

where $L = 2\lambda_{\text{kv}} B_c C^2 + 2\lambda_{\text{prox}}$ is a global upper bound on the gradient Lipschitz constant and $C = \max_i \|\mathbf{k}_i\|$.

Proof:

Step 1 : A reusable matrix gradient lemma

Lemma A.4 *Let $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$, $C \in \mathbb{R}^{m\times p}$,*

$$\nabla_A \|AB - C\|_F^2 \;=\; 2\,(AB - C)\,B^\top. \tag{A.12}$$

Proof:

Let $\mathcal{R} = AB - C$. Then $\|\mathcal{R}\|_F^2 = \operatorname{tr}(\mathcal{R}^\top \mathcal{R})$, so

$$d\|\mathcal{R}\|_F^2 = 2\operatorname{tr}(\mathcal{R}^\top\, d\mathcal{R}). \tag{A.13}$$

Since only A varies, $d\mathcal{R} = dA \cdot B$, giving

$$d\|\mathcal{R}\|_F^2 = 2\operatorname{tr}(\mathcal{R}^\top\, dA\, B) \overset{(*)}{=} 2\operatorname{tr}(B\mathcal{R}^\top\, dA) = 2\operatorname{tr}\big((\mathcal{R}B^\top)^\top dA\big), \tag{A.14}$$

Using the cyclic property of the trace in where $(*)$ The gradient is read off the reading. $\nabla_A\|AB - C\|_F^2 = 2(AB - C)B^\top$. □

Step 2 : Gradient of $\mathcal{L}_{\text{consol}}$ and the closed-form minimiser. Writing the objective in compact matrix form, substituting M for M_{task}

$$J(M) \;=\; \lambda_{\text{kv}}\|MK - R\|_F^2 + \lambda_{\text{prox}}\|M - M_{\text{prev}}\|_F^2. \tag{A.15}$$

Applying Lemma A.4 to each term:

KV term ($A \leftarrow M$, $B \leftarrow K$, $C \leftarrow R$):

$$\nabla_M\, \lambda_{\text{kv}}\|MK - R\|_F^2 \;=\; 2\lambda_{\text{kv}}\,(MK - R)\,K^\top. \tag{A.16}$$

Proximal term ($A \leftarrow M$, $B \leftarrow I$, $C \leftarrow M_{\text{prev}}$):

$$\nabla_M\, \lambda_{\text{prox}}\|M - M_{\text{prev}}\|_F^2 \;=\; 2\lambda_{\text{prox}}\,(M - M_{\text{prev}}). \tag{A.17}$$

Total gradient:

$$\nabla_M J(M) \;=\; 2\lambda_{\text{kv}}\,(MK - R)K^\top + 2\lambda_{\text{prox}}\,(M - M_{\text{prev}}). \tag{A.18}$$

Collecting M by setting $\nabla_M J = 0$:

$$M\big(\lambda_{\mathbf{kv}}\, KK^\top + \lambda_{\text{prox}}\, I\big) \;=\; \lambda_{\text{kv}}\, RK^\top + \lambda_{\text{prox}}\, M_{\text{prev}}. \tag{A.19}$$

Since $\lambda_{\text{kv}}\, KK^\top + \lambda_{\text{prox}}\, I$ is positive definite for $\lambda_{prox} > 0$ The unique global minimiser is

$$\boxed{M^*_{\text{fast}} \;=\; \big(\lambda_{\text{kv}}\, RK^\top + \lambda_{\text{prox}}\, M_{\text{prev}}\big)\big(\lambda_{\text{kv}}\, KK^\top + \lambda_{\text{prox}}\, I\big)^{-1}.} \tag{A.20}$$

Step 3 : Hessian and global strong convexity. From (A.18), the gradient is

$$\nabla_M J(M) \;=\; 2\lambda_{\text{kv}}\,(MK - R)K^\top + 2\lambda_{\text{prox}}\,(M - M_{\text{prev}}). \tag{A.21}$$

Dropping the constants R and M_{prev} and applying Lemma A.4 once more and now differentiating $\nabla_M J$ with respect to M gives the Hessian operator:

$$\begin{aligned}\nabla_M\big[\nabla_M J\big] \;&=\; 2\lambda_{\text{kv}}\,\nabla_M\big[MKK^\top\big] + 2\lambda_{\text{prox}}\,\nabla_M\big[M\big] \\ &=\; 2\lambda_{\text{kv}}\,KK^\top + 2\lambda_{\text{prox}}\,I,\end{aligned} \tag{A.22}$$

which is a constant (independent of M), confirming J is exactly quadratic. Since $KK^\top \succeq 0$ and $\lambda_{\text{prox}} > 0$:

$$\nabla^2 J \;=\; 2\lambda_{\text{kv}}\,KK^\top + 2\lambda_{\text{prox}}\,I \;\succeq\; 2\lambda_{\text{prox}}\,I \;=\; \mu\,I, \qquad \mu = 2\lambda_{\text{prox}} > 0. \tag{A.23}$$

Hence $J(M)$ is *globally* μ-strongly convex.

Step 4: Global upper bound on the gradient Lipschitz constant. From (A.18), the gradient of the KV term is $2\lambda_{\text{kv}}(MK - P)K^\top$, which is linear in M. For any M, M', set $C = \max_i \|\mathbf{k}_i\|$:

$$\begin{aligned}\big\|2\lambda_{\text{kv}}(MK - P)K^\top - 2\lambda_{\text{kv}}(M'K - P)K^\top\big\|_F \;&=\; 2\lambda_{\text{kv}}\big\|(M - M')KK^\top\big\|_F \\ &\le\; 2\lambda_{\text{kv}}\|KK^\top\|_2\,\|M - M'\|_F \\ &\le\; 2\lambda_{\text{kv}}B_cC^2\,\|M - M'\|_F,\end{aligned} \tag{A.24}$$

where the last step uses $\|KK^\top\|_2 = \lambda_{\max}(KK^\top) \le \sum_i \|\mathbf{k}_i\|^2 \le B_cC^2$.

Similarly, the gradient of the proximal term is $2\lambda_{\text{prox}}(M - M_{\text{prev}})$, giving

$$\big\|2\lambda_{\text{prox}}(M - M_{\text{prev}}) - 2\lambda_{\text{prox}}(M' - M_{\text{prev}})\big\|_F \;=\; 2\lambda_{\text{prox}}\|M - M'\|_F. \tag{A.25}$$

By the triangle inequality:

$$\|\nabla_M J(M) - \nabla_M J(M')\|_F \;\le\; L\,\|M - M'\|_F, \qquad L \;=\; 2\lambda_{\text{kv}}B_cC^2 + 2\lambda_{\text{prox}}. \tag{A.26}$$

The tighter exact Lipschitz constant $L_{\text{exact}} = \lambda_{\max}(\nabla^2 J) = 2\lambda_{\text{kv}}\lambda_{\max}(KK^\top) + 2\lambda_{\text{prox}}$ follows directly from the Hessian in (A.22), and satisfies $L_{\text{exact}} \le L$ by the same spectral bound above.

Step 5 : Geometric convergence. Let $e_t = J(M_t) - J(M^*)$. Since J is μ-strongly convex and L-smooth, gradient descent with step size $\eta = 1/L$ satisfies the descent lemma:

$$e_{t+1} \;\le\; e_t - \frac{1}{2L}\|\nabla_M J(M_t)\|_F^2. \tag{A.27}$$

The Polyak–Łojasiewicz inequality (implied by μ-strong convexity) gives $\|\nabla_M J(M_t)\|_F^2 \geq 2\mu\, e_t$, so:

$$e_{t+1} \;\leq\; \left(1 - \frac{\mu}{L}\right) e_t \;=\; \left(1 - \frac{2\lambda_{\text{prox}}}{L}\right) e_t. \tag{A.28}$$

Applying this recursively from $t = 0$:

$$J(M_t) - J(M^*) \;\leq\; \left(1 - \frac{2\lambda_{\text{prox}}}{L}\right)^t \big[J(M_0) - J(M^*)\big]. \tag{A.29}$$

□