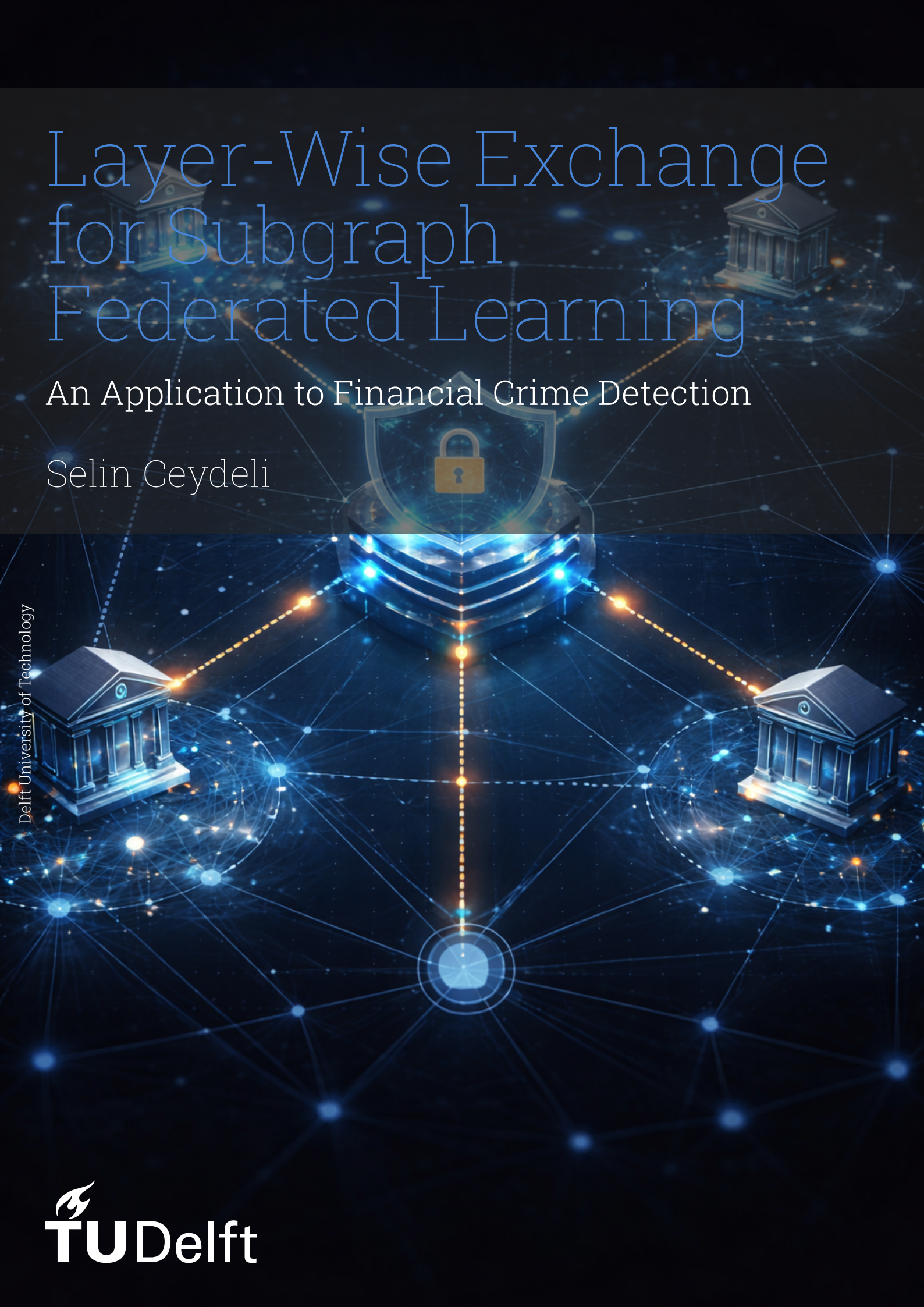
Layer-Wise Exchange for Subgraph Federated Learning

An Application to Financial Crime Detection

Selin Ceydeli



Delft University of Technology

TU Delft

Layer-Wise Exchange for Subgraph Federated Learning

An Application to Financial Crime Detection

by

Selin Ceydeli

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Wednesday June 24, 2026 at 13:45

Thesis committee:	Dr. Zeki Erkin — TU Delft, Chair
	Dr. Kubilay Atasu — TU Delft, Core Member & Supervisor
	Dr. Burcu Özkan — TU Delft, Core Member
	Dr. Rui Wang — TU Delft, Co-supervisor
Thesis Duration:	10, 2025 - 06, 2026
Faculty:	Faculty of Electrical Engineering, Mathematics & Computer Science

Cover:	AI-generated illustration representing subgraph federated learning, enabling collaboration among financial institutions for financial crime detection, the central theme of this thesis.
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Preface

This thesis is the result of my graduation research and serves as the final requirement for obtaining the degree of Master of Science (MSc) in Computer Science at Delft University of Technology. The work presented in this thesis was conducted within the Data-Intensive Systems group of the Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS).

First, I would like to express my sincere gratitude to my supervisors, Kubilay Atasu and Rui Wang, for their guidance, insightful discussions, and valuable feedback throughout this research and the writing of this thesis.

I greatly enjoyed conducting research at the intersection of graph machine learning, federated learning, and financial crime detection. I found it especially rewarding to work on a topic with potential real-world impact, as advances in collaborative financial crime detection may help uncover illicit activities that would otherwise remain hidden across institutional boundaries.

Finally, I would like to thank my family and friends for their encouragement and support throughout my studies and during the completion of this thesis.

Selin Ceydeli
Delft, June 2026

Abstract

Subgraph pattern detection aims to uncover complex interaction structures, for example those associated with money laundering in financial transaction networks. State-of-the-art graph neural network (GNN) solutions, however, assume centralized access to the entire graph. When the graph is instead distributed across multiple financial institutions, each client computes node representations using only its own subgraph, so client-local GNN computations diverge from those of a centralized model. We formalize this divergence as the *structural observability problem*, in which subgraph patterns crossing partition boundaries become locally unidentifiable. This divergence manifests as both a forward gap in the node representations and a backward gap in the training-time adjoint signal. To close both gaps, we propose a per-step, layer-wise exchange framework with two complementary components: a *forward exchange* that synchronizes node representations at every layer of the forward pass, and a *backward exchange* that synchronizes the corresponding gradient signals at every layer of the backward pass; neither component exposes raw features or labels. Under an extended subgraph assumption and shared model parameters across clients, we prove that the forward exchange recovers the representations a centralized GNN would compute over the full graph (*representation equivalence*) and the backward exchange makes the per-client parameter gradients sum to the exact centralized gradient (*gradient equivalence*). Together, forward and backward exchange make federated training equivalent to centralized training. Experiments on synthetic directed multigraphs with cycle, biclique, and scatter-gather patterns show that forward exchange and federated parameter aggregation are complementary rather than interchangeable, and that their combination recovers most of the gap to centralized performance. This recovery depends on per-step freshness, with stale per-epoch exchange leaving a measurable residual. Adding backward exchange yields further improvements, with the largest gains achieved when the cross-client connectivity is densest.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
AML	Anti-money laundering
FedAvg	Federated averaging
FL	Federated learning
Fwd-LE	Forward layer-wise exchange
Fwd/Bwd-LE	Forward/backward layer-wise exchange
GNN	Graph neural network
LE	Layer-wise exchange
ML	Machine learning
MPNN	Message passing neural network
PNA	Principal Neighborhood Aggregation
PR-AUC	Precision-Recall Area Under the Curve
RQ	Research question
SGD	Stochastic gradient descent
Sync-SGD	Synchronous SGD

1

Introduction

Money laundering remains a major challenge for financial institutions and regulators worldwide (Zhiyuan Chen et al., 2018; Nicholls, Kuppa, and Le-Khac, 2021). Detecting laundering activity requires analyzing complex patterns of financial transactions that span large networks of institutions and accounts (Kurshan, Shen, and Yu, 2020).

Financial transaction networks naturally form directed multigraphs (Egressy et al., 2024), where multiple transactions can occur between any two accounts (or nodes), and the directionality of the transactions (or edges) is important. In these networks, money laundering schemes often manifest as subgraph patterns (Altman et al., 2023). Figure 1.1 illustrates three established money laundering patterns. Directed cycles (a) route funds back to the originator through intermediate accounts, scatter-gather flows (b) fragment and later re-consolidate transfers, and directed bicliques (c) capture coordinated many-to-many transfers between groups of accounts. Detecting such patterns requires models that can reason over graph structure. Recently, graph neural networks (GNNs) (L. Wu et al., 2022) have become central in anti-money laundering (AML) solutions. By propagating information across local neighborhoods, GNNs can effectively capture the relational structures that define these patterns, making them a suitable choice for AML detection tasks.

State-of-the-art GNNs for subgraph pattern detection typically assume centralized data, relying on the entire graph to propagate information (Vignac, Loukas, and Frossard, 2020; You et al., 2021; Huang et al., 2023; Eliasof et al., 2023; Tahmasebi, Lim, and Jegelka, 2023; Egressy et al., 2024). In practice, transaction data is often distributed across multiple institutions, so each client holds only a subset of the global graph while cross-client patterns must still be discovered collaboratively. A natural approach is to treat this as a federated learning (FL) problem, with clients training local GNNs and aggregating parameters through a server (McMahan et al., 2016). However, there remains a key challenge: each client sees only a portion of any subgraph pattern whose full substructure spans multiple clients.

Prior work on subgraph FL frame this challenge as the missing neighbor problem. Zhang, Yang, et al., 2021; Zhang, Sun, et al., 2024 tackle it by generating synthetic neighbor features, which can partially compensate for unavailable boundary information without directly sharing local topology or features across clients. While these approaches improve structural observability, synthesized representations may deviate from the true neighborhood structure. An orthogonal approach by Liu et al., 2023; Aliakbari, Östman, and Amat, 2024 is to construct the global topology at the server, making remote structures available to clients. However, this approach leads to additional complexity at the server and may raise scalability or privacy concerns. A third line of work exchanges node embeddings between clients during training (Naman and Simmhan, 2026; A. Li et al., 2024), but exchanges are performed once per epoch and therefore, the exchanged representations are one-epoch stale. Although existing approaches mitigate the effects of missing cross-client information, they do not directly characterize or address the underlying information gap between federated and centralized training.

This thesis formalizes this information gap as the *structural observability problem* and proposes a per-step, layer-wise exchange framework that closes it. The framework has two complementary compo-

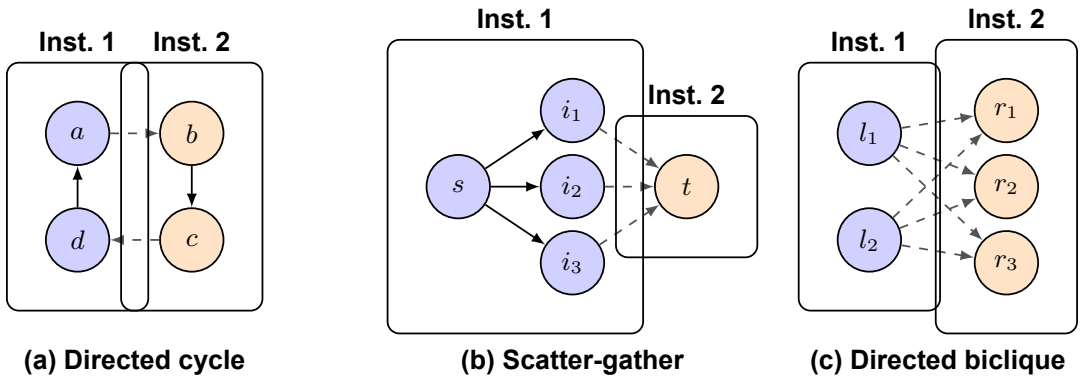


Figure 1.1: Three money laundering patterns spanning two institutions (Inst. 1 and Inst. 2). Solid black edges denote intra-institution transfers, while dashed gray edges cross institutional boundaries.

nents: a *forward exchange* that synchronizes node representations during the forward pass, and a *backward exchange* that synchronizes the corresponding gradient signals during the backward pass. We prove that, under the extended subgraph assumption (A1) and the shared-parameter assumption (A2) (all clients use identical GNN weights), the framework eliminates this gap exactly, making federated training equivalent to centralized training.

This work is guided by one main research question and five supporting subquestions:

- **Main RQ:** How much of the performance gap between federated and centralized training does layer-wise exchange close, and under what conditions?
  - **Supporting RQ1:** How much of this gap is closed by standard federated parameter aggregation alone, without layer-wise exchange?
  - **Supporting RQ2:** How much does federated parameter aggregation add on top of forward exchange?
  - **Supporting RQ3:** How much additional recovery does per-step forward exchange provide over per-epoch exchange?
  - **Supporting RQ4:** How does the frequency of parameter aggregation (per-epoch vs. per-step) affect the recovery achieved by forward exchange?
  - **Supporting RQ5:** How much additional recovery does backward exchange provide on top of forward exchange?

We address each of these questions in turn. Supporting RQ1 compares centralized training against fully local training and standard federated parameter aggregation, where the residual gap quantifies the performance loss due to structural unobservability. Supporting RQ2 measures the respective contributions of forward exchange and federated parameter aggregation by comparing their joint use against forward exchange in isolation under fully local training. Supporting RQs 3–4 then isolate the impact of exchange frequency (per-step vs. per-epoch) and parameter-aggregation frequency (per-epoch vs. per-step) on top of layer-wise exchange. Finally, Supporting RQ5 quantifies the additional recovery contributed by the backward exchange, by comparing each forward/backward configuration against its forward-only counterpart. Collectively, the answers to Supporting RQs 1–5 provide an answer to the main research question by identifying how much of the federated-centralized performance gap is recovered by each component of the proposed framework and under which conditions. We evaluate on synthetic directed multigraphs with cycle, biclique, and scatter-gather patterns, while varying the degree of cross-client partitioning. We additionally analyze the per-epoch communication cost of each method, characterizing the accuracy-communication trade-off across the evaluated methods.

1.1. Outline

The remainder of this thesis is structured as follows. Chapter 2 reviews the relevant background on graph neural networks and federated learning, and positions this work within the existing subgraph federated learning literature. Chapter 3 describes the synthetic directed multigraph benchmark used in the empirical evaluation. Chapter 4 formalizes the subgraph federated learning setting and defines the structural observability problem. Chapter 5 introduces the layer-wise exchange framework and proves its representation and gradient equivalence to centralized training. Chapter 6 specifies the implementation of each evaluated method, and Chapter 7 reports the experimental results. Chapter 8 analyzes the per-epoch communication cost of each method. Chapter 9 discusses the broader implications and practical considerations of the framework, and Chapter 10 concludes the thesis and outlines the directions for future work.

2

Background & Related Work

This chapter introduces the concepts on which the rest of this thesis builds and positions the proposed framework within the existing literature. We first review graph neural networks, the message-passing formulation that the layer-wise exchange framework synchronizes, and the Principal Neighborhood Aggregation (PNA) architecture used throughout the experiments (Section 2.1), followed by stochastic gradient descent and backpropagation (Section 2.2). We then cover federated learning and its parameter-aggregation strategies (Section 2.3), and finally survey subgraph federated learning, framing the missing neighbor problem and positioning the structural observability problem relative to prior work (Section 2.4).

2.1. Graph Neural Networks (GNNs)

GNNs are a class of deep learning models that operate on a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges, and each node $v \in V$ is associated with an input feature vector $\mathbf{x}_v \in \mathbb{R}^{d_0}$ (Scarselli et al., 2009; Kipf and Welling, 2017). A GNN learns a node representation $\mathbf{h}_v \in \mathbb{R}^d$ through iterative aggregation of neighborhood information, where after L layers, the representation $\mathbf{h}_v^{(L)}$ encodes structural information from the L-hop neighborhood of v.

GNNs have been successfully applied to node classification, link prediction, and graph classification tasks across domains including social networks, molecular property prediction, and financial transaction networks (Hamilton, Ying, and Leskovec, 2017; Gilmer et al., 2017; Egressy et al., 2024). In financial transaction networks, which is the motivating application of this thesis, graphs are represented as multigraphs. In such graphs, the directionality and multiplicity of edges carry important structural signals. Standard GNN designs that treat edges as undirected or aggregate over a single edge type are therefore insufficient for this setting, motivating the use of models that explicitly handle both directions and edge attributes (Egressy et al., 2024).

2.1.1. Message Passing Neural Networks (MPNN)

Most GNN architectures can be expressed with the MPNN framework (Gilmer et al., 2017). At each GNN layer l, each participating node v collects messages from its neighbors, aggregates them using an aggregation function AGG, and updates its own representation. Message passing can be formally defined as:

$$\mathbf{m}_v^{(l)} = \mathrm{AGG}\Big(\Big\{ f_\theta\Big(\mathbf{h}_u^{(l-1)}, \mathbf{h}_v^{(l-1)}, \mathbf{e}_{uv}\Big) : u \in \mathcal{N}(v) \Big\}\Big), \tag{2.1}$$

$$\mathbf{h}_v^{(l)} = U_\theta\Big(\mathbf{h}_v^{(l-1)}, \mathbf{m}_v^{(l)}\Big), \tag{2.2}$$

where $\mathbf{h}_v^{(l)} \in \mathbb{R}^d$ is the representation of v at layer l, $\mathbf{e}_{uv}$ is an optional edge attribute, $\mathcal{N}(v)$ is the set of

neighbors of v, and f_θ, U_θ are learnable functions. The recursion is initialized by an input embedding $\mathbf{h}_v^{(0)} = \phi_{\text{emb}}^{\theta_{\text{emb}}}(\mathbf{x}_v)$ that projects the raw features $\mathbf{x}_v \in \mathbb{R}^{d_0}$ into the d-dimensional representation space before the L rounds of message passing. The learnable parameters of an L-layer MPNN are thus $\Theta = \{\theta_{\text{emb}}\} \cup \{\theta^{(l)}\}_{l=0}^{L-1}$, comprising the input embedding and the per-layer message/update functions.

The expressiveness of the GNN model that uses the MPNN framework is determined by the chosen aggregation function AGG. Xu et al. (2019) demonstrate that sum aggregation is strictly more expressive than the mean or max functions under the Weisfeiler–Leman graph isomorphism test. Extending this expressivity discussion, Corso et al. (2020) empirically show that no single aggregator is universally optimal for all tasks.

2.1.2. Principal Neighborhood Aggregation (PNA)

Corso et al. (2020) address the expressivity limitation of single-aggregator GNNs by proposing PNA, which combines multiple aggregators with degree-aware scalers in each layer.

Multiple Aggregators

Rather than applying a single aggregation function, PNA computes several aggregators in parallel and concatenates their outputs:

$$\mathrm{AGG}_{\text{PNA}}(\mathcal{M}) = \Big\Vert_{\alpha \in \mathcal{A}} \alpha(\mathcal{M}),$$

where $\mathcal{M} = \{\mathbf{m}_u : u \in \mathcal{N}(v)\}$ is the multiset of neighbor messages and $\|$ denotes concatenation.

The aggregator set is $\mathcal{A} = \{\mu, \sigma, \max, \min\}$, where μ and σ denote element-wise mean and standard deviation over neighbors, and $\min$ and $\max$ denote the element-wise minimum and maximum over neighbor messages, each computed independently per feature dimension.

Degree-Aware Scalers

Because aggregators such as element-wise mean discard information about neighborhood size, Corso et al. (2020) introduce degree-aware scalers. These scalers modulate the aggregated message by the in-degree δ_v of the receiving node relative to the mean in-degree $\bar{\delta}$ of the training graph.

The scaler set is $\mathcal{S} = \{s_{\text{id}}, s_{\text{amp}}, s_{\text{att}}\}$, where s_{id} is the identity, s_{amp} is the amplifier, and s_{att} is the attenuator:

$$\begin{aligned} s_{\text{id}}(\mathbf{m}, \delta_v) &= \mathbf{m}, \\ s_{\text{amp}}(\mathbf{m}, \delta_v) &= \mathbf{m} \cdot \frac{\log(\delta_v + 1)}{\log(\bar{\delta} + 1)}, \\ s_{\text{att}}(\mathbf{m}, \delta_v) &= \mathbf{m} \cdot \frac{\log(\bar{\delta} + 1)}{\log(\delta_v + 1)}. \end{aligned}$$

s_{id} leaves the aggregated message unchanged, s_{amp} amplifies messages for nodes with higher-than-average degree, and s_{att} attenuates them for nodes with lower-than-average degree. Together they allow the model to encode neighborhood size as an explicit signal rather than discarding it through normalization.

Full PNA Aggregation

The full PNA aggregation applies every aggregator-scaler pair and concatenates the results:

$$\mathrm{AGG}_{\text{PNA}}(\mathcal{M}, \delta_v) = \Big\Vert_{\alpha \in \mathcal{A}} \Big\Vert_{s \in \mathcal{S}} s(\alpha(\mathcal{M}), \delta_v),$$

yielding $|\mathcal{A}| \times |\mathcal{S}| = 4 \times 3 = 12$ components per feature dimension before the update function U_θ.

2.1.3. Adaptations for Directed Multigraphs

Standard message-passing GNNs cannot detect some subgraph patterns that emerge on directed multigraphs (Zhengdao Chen et al., 2020). Egressy et al. (2024) show that three adaptations, which are (i) reverse message passing, (ii) directed multigraph port numbering, and (iii) ego IDs, together turn a sufficiently expressive MPNN into a model that can provably detect *any* directed subgraph pattern. Applied on top of PNA, these adaptations yield the Multi-PNA architecture used throughout this thesis.

Reverse message passing. A standard directed MPNN aggregates only over incoming neighbors $\mathcal{N}^{\text{in}}(v)$, so a node never receives messages along its outgoing edges and cannot, for instance, count its out-degree (Jaume et al., 2019). Reverse message passing adds a second, separate aggregation over the outgoing neighbors $\mathcal{N}^{\text{out}}(v)$, treating the two directions as distinct edge types:

$$\begin{aligned}
\mathbf{m}_v^{(l),\text{in}} &= \text{AGG}_{\text{in}}\Big(\Big\{ f_\theta\Big(\mathbf{h}_u^{(l-1)}, \mathbf{h}_v^{(l-1)}, \mathbf{e}_{uv}\Big) : u \in \mathcal{N}^{\text{in}}(v)\Big\}\Big), \\
\mathbf{m}_v^{(l),\text{out}} &= \text{AGG}_{\text{out}}\Big(\Big\{ f_\theta\Big(\mathbf{h}_u^{(l-1)}, \mathbf{h}_v^{(l-1)}, \mathbf{e}_{vu}\Big) : u \in \mathcal{N}^{\text{out}}(v)\Big\}\Big), \\
\mathbf{h}_v^{(l)} &= U_\theta\Big(\mathbf{h}_v^{(l-1)}, \mathbf{m}_v^{(l),\text{in}}, \mathbf{m}_v^{(l),\text{out}}\Big).
\end{aligned}$$

This is equivalent to a relational GNN with two edge types (a forward and a reverse relation), and lets the model distinguish incoming from outgoing structure rather than collapsing both into a single undirected aggregation.

Directed multigraph port numbering. When there are parallel edges in a graph (for example when there are multiple transactions between the same pair of accounts), a node cannot differentiate, from message passing alone, whether several messages come from the same neighbor or from distinct neighbors (Sato, Yamada, and Kashima, 2019). Port numbering assigns each directed edge an incoming and an outgoing port number, where all edges coming from (or going to) the same node share an incoming (or outgoing) port number. This way, a node can identify messages that originate from the same neighbor across rounds.

Ego IDs. Ego IDs mark the center, or *ego*, node of each sampled computation graph with a distinct binary feature (You et al., 2021). This lets a node recognize when a chain of messages returns to it, providing the signal needed to detect cycles that it participates in. On their own, ego IDs are not sufficient for cycle detection, but in combination with reverse message passing and port numbering, they complete the set of patterns the model can identify.

2.2. Stochastic Gradient Descent (SGD)

Neural networks are trained by iteratively minimizing a loss $\mathcal{L}(\Theta)$ over a parameter vector $\Theta \in \mathbb{R}^p$. Full-batch gradient descent updates parameters in the direction of the negative gradient computed over the entire training set $\mathcal{D}$, such that $\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \eta\, \nabla_\Theta\, \mathcal{L}(\Theta^{(t)}; \mathcal{D})$, where $\eta > 0$ denotes the learning rate. For large datasets, evaluating the gradient over all samples at every step is prohibitively expensive, so stochastic gradient descent (SGD) (Robbins and Monro, 1951; Bottou, 2010) replaces the full-batch gradient with an unbiased estimate computed from a uniformly sampled mini-batch $\mathcal{B}^{(t)} \subset \mathcal{D}$:

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \eta\, \nabla_\Theta\, \mathcal{L}(\Theta^{(t)}; \mathcal{B}^{(t)}). \tag{2.3}$$

Mini-batch sampling lowers the per-step computational cost from $O(|\mathcal{D}|)$ to $O(|\mathcal{B}|)$ and acts as an implicit regularizer, at the cost of introducing gradient noise. Adaptive variants such as Adam (Kingma and Ba, 2015) mitigate this noise by maintaining exponential moving averages of the first and second moments of past gradients to rescale per-parameter step sizes.

2.2.1. Backpropagation

The gradient $\nabla_\Theta \mathcal{L}$ in Eq. 2.3 is computed by backpropagation (Rumelhart, Hinton, and Williams, 1986), i.e. reverse-mode automatic differentiation. For a model that produces a node representation $\mathbf{h}^{(l)}$ at each layer l, backpropagation proceeds from the loss toward the input and, at every intermediate representation, forms the *adjoint*:

$$\boldsymbol{\delta}^{(l)} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(l)}},$$

which is the gradient of the loss with respect to that representation. By the chain rule, the layer-l adjoint is obtained from the layer-$(l+1)$ adjoint by multiplication with the transposed local derivative,

$$\boldsymbol{\delta}^{(l)} = \left(\frac{\partial \mathbf{h}^{(l+1)}}{\partial \mathbf{h}^{(l)}} \right)^{\top} \boldsymbol{\delta}^{(l+1)}, \tag{2.4}$$

so the adjoint is propagated backward along the same edges that carried activations forward. Because these local derivatives depend on the values at which they are evaluated, the backward pass reuses the activations computed in the forward pass. The parameter gradient $\partial \mathcal{L} / \partial \theta^{(l)}$ is then read off from the adjoint at layer l.

2.3. Federated Learning (FL)

FL is a distributed machine learning paradigm in which C clients collaboratively train a shared model without exchanging their local data (McMahan et al., 2016). Each client $c \in \{1, \dots, C\}$ holds a local dataset $\mathcal{D}_c$ of size $n_c = |\mathcal{D}_c|$, and the federation jointly minimizes the sample-weighted global objective:

$$\mathcal{L}(\Theta) \;=\; \sum_{c=1}^{C} \frac{n_c}{\sum_{c'=1}^{C} n_{c'}} \, \mathcal{L}_c(\Theta), \qquad \mathcal{L}_c(\Theta) \;=\; \mathbb{E}_{\xi \sim \mathcal{D}_c}[\ell(\Theta;\, \xi)]\,, \tag{2.5}$$

where ℓ is the per-sample loss. For a mini-batch $\mathcal{B} \subset \mathcal{D}_c$, we denote by $\mathcal{L}_c(\Theta;\, \mathcal{B})$ the corresponding empirical loss. The federation is coordinated by a central server that aggregates the model updates received from the clients and broadcasts the resulting global parameters back. FL algorithms differ primarily in *what* they aggregate (parameters vs. gradients) and *how often* they synchronize (per local epoch vs. per mini-batch).

2.3.1. Federated Averaging (FedAvg)

FedAvg (McMahan et al., 2016) is the most widely adopted FL algorithm. At each global round r, the server samples a subset of clients $\mathcal{S}^{(r)}$ and broadcasts the current global parameters $\Theta^{(r)}$. Each selected client $c \in \mathcal{S}^{(r)}$ initializes $\Theta_c^{(r,0)} \leftarrow \Theta^{(r)}$ and performs E local SGD epochs on its dataset $\mathcal{D}_c$. For each local step $t = 0, \dots, T_c - 1$, the client applies one SGD update (cf. Eq. 2.3) to the local objective:

$$\Theta_c^{(r,t+1)} \;\leftarrow\; \Theta_c^{(r,t)} \;-\; \eta \, \nabla_\Theta \, \mathcal{L}_c\Big(\Theta_c^{(r,t)};\, \mathcal{B}_c^{(r,t)}\Big)\,, \tag{2.6}$$

where $\mathcal{B}_c^{(r,t)} \subset \mathcal{D}_c$ is a local mini-batch. After completing T_c local steps, each client returns its updated parameters $\Theta_c^{(r)} := \Theta_c^{(r,T_c)}$, and the server aggregates them by sample-weighted averaging:

$$\Theta^{(r+1)} \;=\; \sum_{c \in \mathcal{S}^{(r)}} \frac{n_c}{\sum_{c' \in \mathcal{S}^{(r)}} n_{c'}} \, \Theta_c^{(r)}. \tag{2.7}$$

The updated global model is then broadcast to all clients for the next round. Because clients perform multiple local SGD steps between synchronizations, FedAvg reduces communication cost at the price

of additional local computation. Between synchronization rounds, however, client parameters drift apart as each client follows the gradient of its own local objective, and the server's averaging step only re-aligns them at the end of each round.

2.3.2. Synchronous SGD (Sync-SGD)

Sync-SGD is a federated variant of mini-batch SGD that synchronizes parameters after *every* mini-batch rather than after several local epochs as in FedAvg. At step t, all C clients hold the same model $\Theta^{(t)}$. Each client c samples a mini-batch $\mathcal{B}_c^{(t)}$ from its dataset $\mathcal{D}_c$ and computes a local gradient

$$g_c^{(t)} \;=\; \nabla_\Theta \, \mathcal{L}_c\Big(\Theta^{(t)};\, \mathcal{B}_c^{(t)}\Big)\,. \tag{2.8}$$

The server aggregates these gradients using sample-weighted averaging, matching the weighting of the global objective in Eq. 2.5:

$$g^{(t)} \;=\; \sum_{c=1}^{C} \frac{n_c}{\sum_{c'=1}^{C} n_{c'}}\, g_c^{(t)}, \tag{2.9}$$

and applies a single SGD step (cf. Eq. 2.3) to the shared parameters:

$$\Theta^{(t+1)} \;\leftarrow\; \Theta^{(t)} \;-\; \eta\, g^{(t)}. \tag{2.10}$$

Because all clients receive the same updated parameters after each mini-batch, Sync-SGD maintains parameter equivalence throughout training at a much finer granularity than FedAvg, at the cost of one communication round per gradient step rather than per local epoch.

2.4. Subgraph Federated Learning

The subgraph FL framework adapts standard FL to graph-structured data, where a global graph is partitioned among participating clients, each holding a disjoint subgraph (Zhang, Yang, et al., 2021; X. Li et al., 2025). Each client trains a local GNN model on its subgraph using standard neighborhood aggregation (Hamilton, Ying, and Leskovec, 2017). During federated training, clients periodically exchange model parameters with a central server, which aggregates them, typically by weighted averaging as in FedAvg (McMahan et al., 2016), and distributes the updated global model back to clients.

2.4.1. The Missing Neighbor Problem

A fundamental challenge in subgraph FL is the missing neighbor problem (Zhang, Yang, et al., 2021; X. Li et al., 2025; Yao et al., 2023; Zhang, Sun, et al., 2024), which arises from partitioning the global graph across clients. Since edges can span multiple clients, the relational dependencies between nodes are disrupted, and a node's full neighborhood may not be locally available during message passing. Boundary nodes, which are nodes with neighbors on other clients, can only aggregate over the subset of neighbors residing on the same client, resulting in a biased and incomplete estimate of their true neighborhood representation.

Prior work address the missing neighbor problem through five main design strategies, differing in how and when cross-client information is approximated or exchanged: 1) synthetic neighbor generation, 2) global graph reconstruction, 3) privacy-preserving graph expansion, 4) one-time feature and propagation exchange, and 5) continuous embedding exchange at every training epoch.

2.4.2. Synthetic Neighbor Generation

Zhang, Yang, et al., 2021 tackle the missing neighbor problem by training a neighbor generator on each client that synthesizes feature vectors for missing boundary neighbors, extending local subgraphs with approximate representations of remote nodes without accessing their true features or identities.

Zhang, Sun, et al., 2024 build on this idea by generating synthetic neighbor embeddings that encode multi-hop structural context rather than raw input features, producing richer approximations of missing neighborhood structure. Both methods operate without direct communication of real node data, relying instead on locally trained generative models to approximate the missing information.

2.4.3. Global Graph Reconstruction

Liu et al., 2023 take a different approach by aggregating structural information from all clients at the server to reconstruct an approximate global graph topology. This allows cross-client connections to be reasoned about globally rather than approximated locally. Aliakbari, Östman, and Amat, 2024 similarly leverage explicit global graph structure to capture inter-client dependencies, but decouples structural learning from feature sharing: clients contribute structural information without exchanging node features or learned representations. Although these methods better capture global topology, they introduce additional complexity at the server and may raise scalability or privacy concerns due to the direct sharing of graph topology between clients.

2.4.4. Privacy-Preserving Graph Expansion

C. Wu et al., 2022 propose a privacy-preserving personalization method for decentralized user-item graphs. Their method expands each client's local subgraph via a trusted server that matches encrypted item IDs across clients and returns anonymized neighbor embeddings. These embeddings, however, serve only as static neighbor information rather than as layer-wise messages that approximate centralized propagation.

2.4.5. One-Time Feature or Propagation Exchange

Yao et al., 2023 perform a one-time pre-training exchange of aggregated multi-hop neighbor features, which are reused as fixed cross-client signals during federated training. Similarly, Lei et al., 2023 propose a framework that decouples local graphs into internal and border parts, exchanges precomputed propagation results, and then trains on the resulting node representations. For efficiency, these embeddings are computed only once. Both approaches minimize communication, but cannot adapt to evolving representations.

2.4.6. Continuous Embedding Exchange at Every Training Epoch

A further line of work addresses the practical cost of exchanging node representations during FL. Naman and Simmhan, 2026 introduce an embedding server through which clients push and pull node representations at every training epoch, and propose systems-level optimizations to reduce the communication and memory overhead of this exchange. A. Li et al., 2024 further reduce this overhead by caching historical embeddings and selectively synchronizing cross-client representations based on training dynamics such as embedding staleness. Both approaches treat embedding exchange as a practical substitute for raw feature sharing under privacy constraints, and their primary contributions are in communication efficiency and scalability. However, neither work establishes the conditions under which the exchanged embeddings recover the centralized node representations, and both exchange representations only in the forward pass. Gradient signals are never exchanged in the backward pass.

2.4.7. Positioning of This Work

The approaches above differ in what information is exchanged and when the exchange occurs, but all rely on either approximations, partial structural reconstructions, or stale cross-client signals during training. Furthermore, none exchange the gradient signal that would let each client's update reflect the other clients' losses. Finally, none formally characterizes the information gap between federated and centralized GNN training, nor establishes conditions under which exchanged signals recover the representations or gradients that a centralized model would compute over the full graph. This thesis addresses this gap by formalizing it as the *structural observability problem* (Section 4.3) and closing it with the layer-wise exchange framework (Chapter 5), whose forward and backward components recover the centralized representations and gradients, respectively.

3

Dataset

This chapter first motivates the use of a synthetic pattern detection dataset and then describes its construction in detail.

3.1. Motivation

The motivation for using a synthetic dataset follows the rationale of Egressy et al. (2024): synthetic directed multigraphs allow the expressive power of graph neural networks to be studied under fully controlled conditions. Ground-truth labels are defined deterministically by the presence of structural patterns, so performance differences between models can be attributed directly to their ability to capture these structures. In contrast, real-world AML datasets introduce extreme class imbalance (Fan et al., 2026), highly variable neighborhood sizes (Altman et al., 2023), and large-scale computational cost (Weber et al., 2018), making controlled comparisons across federated configurations difficult.

While a synthetic dataset does not capture the full complexity of real-world AML systems, it enables controlled and reproducible evaluation under subgraph FL constraints, the primary focus of this thesis.

3.2. Dataset Construction

A synthetic dataset is generated to benchmark graph models on the pattern detection tasks, following the pseudocode and configurations of Egressy et al. (2024). The dataset is built around eleven money laundering *patterns*, which are structural objects such as directed cycles and bicliques. Rather than inserting these patterns post hoc, the graphs are produced by a *random circulant-like graph generator* (Egressy et al., 2024), whose parameters (the number of nodes, average degree, and locality radius) control how frequently each pattern arises. Each pattern defines a binary node-classification *task*: for every node, predict whether it participates in that pattern. These eleven tasks fall into two groups: four degree-based and seven higher-order structural (see Figure 3.1 for illustration).

The degree-based tasks are degree-in/out (the number of incoming and outgoing edges) and fan-in/out (the number of unique incoming and outgoing neighbors). For each of these four tasks, a node's label is set to *true* if the corresponding quantity is greater than three, and *false* otherwise.

The remaining seven tasks concern a node's participation in higher-order structural patterns: directed cycles of length two to six (C2–C6), scatter-gather (S-G), and directed bicliques (B-C). The labeling scheme depends on the pattern. For the cycle tasks, every node lying on a directed cycle of the specified length is labeled *true*. For scatter-gather, only the target node that gathers the flow is labeled *true*; the source and intermediate nodes are labeled *false*. For the biclique task, only the sink (right-hand) nodes are labeled *true*, while the source (left-hand) nodes are labeled *false*.

To prevent data leakage between the training, validation, and test datasets, a separate graph is generated for each split, each having the same distribution of the described patterns. The generated graphs

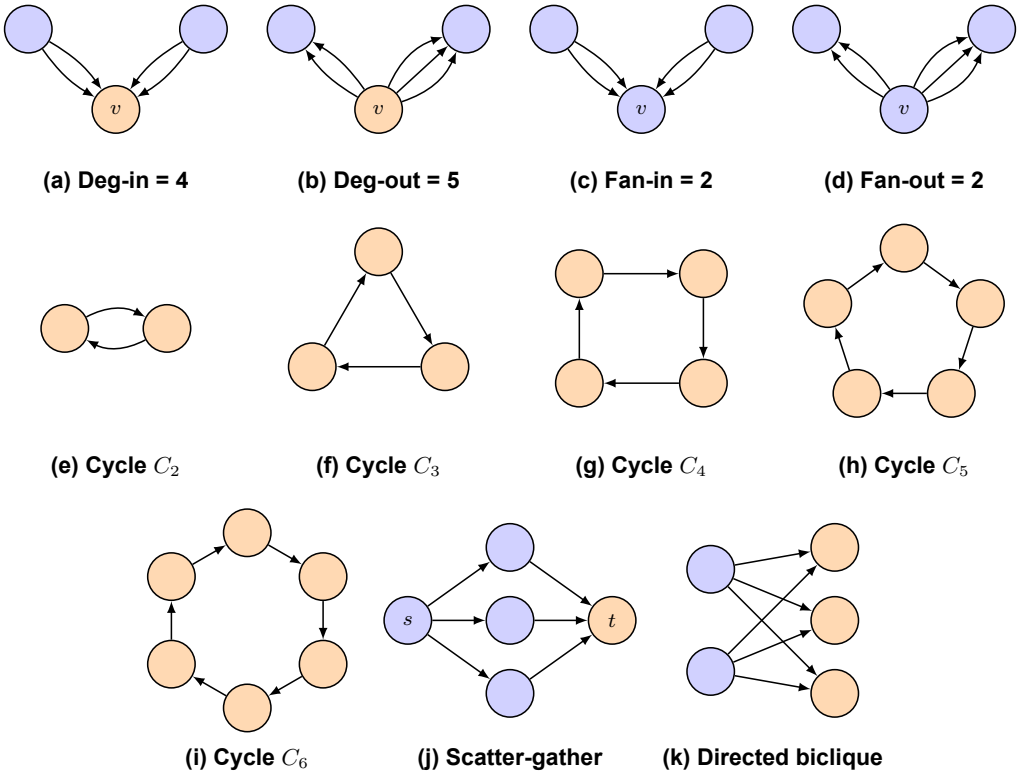


Figure 3.1: Illustrations of the eleven pattern detection tasks in the synthetic dataset; positively labeled nodes are highlighted in orange. (a)–(d) Degree-based patterns: the subject node v is positively labeled when the corresponding count exceeds three. Panels (a) and (c) show the same incoming configuration, and panels (b) and (d) the same outgoing configuration: $\deg$ counts edges with multiplicity (parallel edges contribute separately), whereas fan counts unique neighbors. The shared incoming configuration yields deg-in$(v) = 4$ (positive) but fan-in$(v) = 2$ (negative), and the shared outgoing configuration yields deg-out$(v) = 5$ (positive) but fan-out$(v) = 2$ (negative). (e)–(i) Directed cycles of length $k = 2, 3, \ldots, 6$; every node on the cycle is positively labeled. (j) Scatter-gather: a source s distributes flow through intermediate nodes to a target t; only the target t is positively labeled. (k) Directed biclique: every node on the left has a directed edge to every node on the right; only the sink (right-hand) nodes are positively labeled.

are directed multigraphs. Each graph contains 8192 nodes with an average node degree of 6. All nodes are initialized with a constant vector $\mathbf{x}_v = \mathbf{1}$, and no edge features are used.

The positive-label rates across the generated graphs are summarized in Table 3.1. For C4, C5, and C6, the positive class is the majority.

Table 3.1: Average share of positive labels per pattern in the synthetic train split. The distribution of the labels is consistent across the train, validation, and test splits.

Pattern	Deg-in	Deg-out	Fan-in	Fan-out	C2	C3	C4	C5	C6	S-G	B-C
Positive (%)	35.0	35.3	32.4	32.5	19.3	33.7	52.0	67.1	77.7	31.4	31.5

Scope of experimental evaluation. Although the dataset includes eleven task heads following the construction of Egressy et al. (2024), the experiments in this thesis focus on the seven higher-order structural tasks (C2–C6, S-G, and B-C). These tasks correspond directly to the money laundering patterns of Figure 1.1 and typically span multiple clients, making them the natural setting in which the structural observability problem manifests.

The four degree-based tasks (degree-in/out, fan-in/out) are local properties of a node's immediate neighborhood and are recovered exactly from a client's extended subgraph G_c whenever cross-client edges are retained under assumption (A1). Therefore, these four degree-based tasks do not isolate the federated-learning challenge this thesis addresses and are excluded from the experimental evaluation.

4

Problem Formulation

Having described the data, we now formalize the subgraph federated learning setting and the resulting structural observability problem, which decomposes into a *forward gap* in the node representations and a *backward gap* in the training-time adjoint signal.

4.1. Notation

Let $G = (V, E)$ be a global graph with node features $\mathbf{x} : V \to \mathbb{R}^{d_0}$ and edge features $\mathbf{e} : E \to \mathbb{R}^{d_e}$, distributed across C clients. Each client c owns a subset of nodes V_c^{own}, such that the collection $\{V_c^{\text{own}}\}_{c=1}^{C}$ forms a disjoint partition of V, i.e., $V_c^{\text{own}} \cap V_{c'}^{\text{own}} = \emptyset$ for $c \neq c'$ and $\bigcup_{c=1}^{C} V_c^{\text{own}} = V$.

Let $\mathcal{N}(v) = \mathcal{N}^{\text{in}}(v) \cup \mathcal{N}^{\text{out}}(v)$ denote the set of in- and out-neighbors of v in G. We distinguish two kinds of edges relative to client c: the *intra-client edges* $E_c = \{(u, v) \in E \mid u, v \in V_c^{\text{own}}\}$, with both endpoints owned by c, and the *cross-client edges* E_c^{rem}, with exactly one endpoint owned by c. Each client's owned subgraph is augmented to an extended subgraph $G_c = (V_c^{\text{own}} \cup V_c^{\text{rem}},\ E_c \cup E_c^{\text{rem}})$ that adds, for every cross-client edge, its non-owned endpoint. These added nodes, V_c^{rem}, are owned by other clients and are referred to as *remote nodes*. Client c knows its cross-client edges E_c^{rem} and the existence of its remote nodes V_c^{rem}, but has access to neither the features $\mathbf{x}_u$ nor the labels y_u of any remote node $u \in V_c^{\text{rem}}$. Formally, V_c^{rem} and E_c^{rem} are defined as follows:

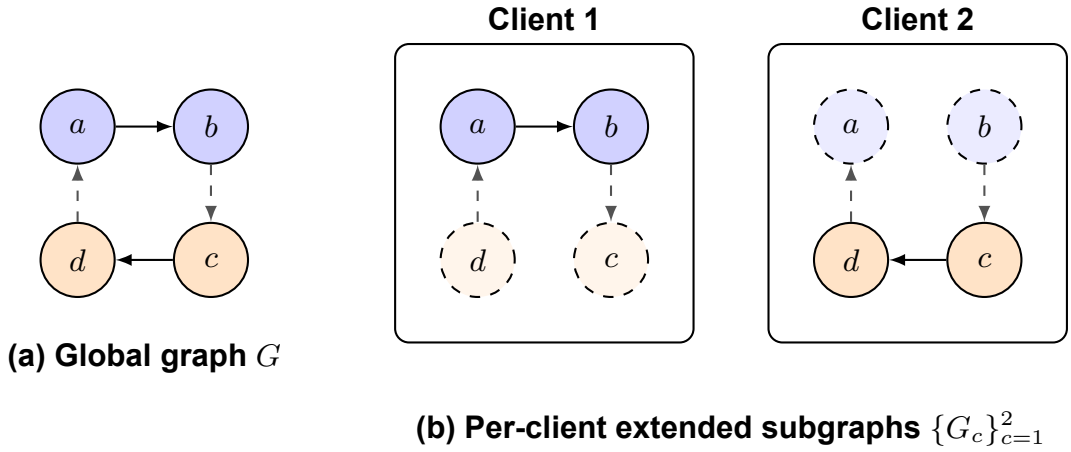
$$V_c^{\text{rem}} = \Big(\bigcup_{v \in V_c^{\text{own}}} \mathcal{N}(v) \Big) \setminus V_c^{\text{own}} \quad \text{and} \quad E_c^{\text{rem}} = \{(u, v) \in E \mid |\{u, v\} \cap V_c^{\text{own}}| = 1\}.$$

Figure 4.1 illustrates these definitions on a 4-cycle partitioned across two clients: each client's view consists of its owned nodes together with the remote endpoints it shares cross-client edges with, drawn at the same coordinates as in the global graph.

Message-passing GNN and representations. Consider a shared L-layer message-passing GNN with an input embedding $\phi_{\text{emb}}^{\theta_{\text{emb}}}$ and parameters $\Theta = \{\theta_{\text{emb}}\} \cup \{\theta^{(l)}\}_{l=0}^{L-1}$, with layer-$l$ update:

$$\mathbf{h}_v^{(l+1)} = \phi_{\theta^{(l)}}^{(l)} \left(\mathbf{h}_v^{(l)},\ \bigoplus_{u \in \mathcal{N}(v)} \psi_{\theta^{(l)}}^{(l)} \left(\mathbf{h}_v^{(l)},\ \mathbf{h}_u^{(l)},\ \mathbf{e}_{uv} \right) \right), \tag{4.1}$$

where $\mathbf{h}_v^{(0)} = \phi_{\text{emb}}^{\theta_{\text{emb}}}(\mathbf{x}_v)$, $\psi^{(l)}$ denotes the message function, $\bigoplus$ is a permutation-invariant aggregation operator, and $\phi^{(l)}$ denotes the node update function. We denote by $\mathbf{h}_v^{(l,\text{cent})}$ the layer-l representation produced by Equation 4.1 when run centrally on G with parameters Θ. For the distributed setting, we distinguish two client-side representations:



(a) Global graph G

(b) Per-client extended subgraphs $\{G_c\}_{c=1}^{2}$

Figure 4.1: Partition of a 4-cycle ($a \to b \to c \to d \to a$) across two clients. **(a)** The global graph G: node colors preview the partition (Client 1: blue $\{a, b\}$, Client 2: orange $\{c, d\}$); solid black edges are intra-client and dashed gray edges are cross-client. **(b)** The extended subgraph $G_c = (V_c^{\text{own}} \cup V_c^{\text{rem}}, E_c \cup E_c^{\text{rem}})$ on each client: *owned* nodes (V_c^{own}) have solid borders, while *remote* nodes (V_c^{rem}) have dashed borders and lightened fill to indicate that the client knows of their existence but holds neither their features nor their labels. Edges between owned nodes are intra-client (E_c, solid black); edges with exactly one owned endpoint are cross-client (E_c^{rem}, dashed gray) and are known to the owning client. The same node appears at the same coordinates across panels for visual continuity.

- $\hat{\mathbf{h}}_{v,c}^{(l)}$ denotes the *naive client-local* representation, in which client c aggregates over $\mathcal{N}(v)$ but substitutes a fixed placeholder $\mathbf{h}_{\text{rem}} = \mathbf{0}$ for every remote-node embedding $\mathbf{h}_u^{(l)}$ with $u \in V_c^{\text{rem}}$ (used in the structural observability problem of Section 4.3).
- $\mathbf{h}_{v,c}^{(l)}$ denotes the representation produced *under the layer-wise exchange protocol* of Chapter 5, in which remote embeddings are received from their owners at every layer.

Training objective and adjoints. Each node v carries a label y_v, and the model is trained with a per-node loss ℓ. The centralized objective and its adjoint (the gradient of the loss with respect to an intermediate representation) are

$$\mathcal{L}^{\text{cent}}(\Theta) = \sum_{v \in V} \ell\Big(\mathbf{h}_v^{(L,\text{cent})}, y_v\Big), \qquad \boldsymbol{\delta}_u^{(l),\text{cent}} = \frac{\partial \mathcal{L}^{\text{cent}}}{\partial \mathbf{h}_u^{(l)}},$$

where the *centralized adjoint* $\boldsymbol{\delta}_u^{(l),\text{cent}}$ is the gradient of the centralized loss with respect to the layer-l representation of node u.

Dual to the forward placeholder $\mathbf{h}_{\text{rem}} = \mathbf{0}$, the naive client-local backward pass substitutes a placeholder adjoint $\boldsymbol{\delta}_{\text{rem}} = \mathbf{0}$ for the gradient returned across each cross-client edge. It differentiates the naive client-local forward pass, so the objective that client c forms over its owned nodes and the adjoint that its owner c' then accumulates are

$$\hat{\mathcal{L}}_c(\Theta) = \sum_{v \in V_c^{\text{own}}} \ell\Big(\hat{\mathbf{h}}_{v,c}^{(L)}, y_v\Big), \qquad \hat{\boldsymbol{\delta}}_u^{(l)} = \frac{\partial \hat{\mathcal{L}}_{c'}}{\partial \hat{\mathbf{h}}_{u,c'}^{(l)}},$$

both evaluated at the naive client-local representations $\hat{\mathbf{h}}$, so the owner c' of node u accumulates only the contribution of its own loss.

4.2. Problem Setting

We consider the following assumptions for our analysis:

(A1) For every client c, the extended subgraph G_c is available, i.e. the remote nodes V_c^{rem} and remote edges E_c^{rem} are known to client c. However, client c has no access to the raw features $\mathbf{x}_u$ or labels y_u of any remote node $u \in V_c^{\text{rem}}$.

(A2) All clients share identical parameters Θ.

(A1) requires that the partitioning scheme retains cross-client edges in each client's extended subgraph, but the remote nodes' features and labels stay with their owners. This is a realistic requirement in distributed graph learning systems, where the existence of a cross-client link is usually observable even when the remote node's features and labels are not. In transaction networks, for instance, each institution records transfers to external parties together with their public identifier (e.g. IBAN), but has no visibility into those parties' internal attributes (Altman et al., 2023). Retaining these cross-client edges preserves the immediate neighborhood information, while multi-hop subgraph patterns spanning several clients remain unobservable to individual clients. The impact of extending the client subgraphs with cross-client edges is empirically shown in Appendix A.

(A2) requires that model parameters across clients are equivalent. This parameter equivalence may arise from federated parameter aggregation (e.g. FedAvg (McMahan et al., 2016)), from synchronous data-parallel training, from a centrally-trained model provided to all clients for inference, or from any other mechanism that yields the same Θ on every client. The representation equivalence of Theorem 5.1 is a statement about a single forward pass under (A2), independent of whether that pass is executed during training or inference. The gradient equivalence of Theorem 5.2 additionally requires the same parameters during the backward pass, which holds automatically within a training step since Θ is fixed between a step's forward and backward computation. In the FL case, the equivalence of the parameters is maintained by aggregating local updates. During inference, it is inherited from the shared model. In all cases, (A2) holds by construction of the parameter-sharing mechanism, independently of how it is realized.

For simplicity, the remainder of our theoretical analysis considers only the full-batch setting, where every node performs aggregation over its entire neighborhood.

4.3. Structural Observability Problem

A client's extended subgraph G_c includes both its cross-client edges and the remote nodes they reach under assumption (A1). Hence, each client knows its boundary structure. What a client lacks is the data on those remote nodes: their raw features $\mathbf{x}_u$ and labels y_u are held by other clients, and so are the representations and adjoints that the centralized computation would derive from them. A client can therefore see a boundary edge and the remote node it connects to, yet cannot reproduce the information that would flow across that edge in the centralized computation. This flow is bidirectional: in the centralized computation graph, an edge (u, w) carries the forward activation $\mathbf{h}_u^{(l)}$, which contributes to the update of $\mathbf{h}_w^{(l+1)}$, and the backward adjoint $\partial\mathcal{L}/\partial\mathbf{h}_u^{(l)}$, which receives a contribution from w. Lacking the remote node's data, the naive client-local computation can supply neither direction and substitutes a zero placeholder for each: $\mathbf{h}_{\text{rem}} = \mathbf{0}$ in the forward pass and $\boldsymbol{\delta}_{\text{rem}} = \mathbf{0}$ in the backward pass. We quantify these two gaps separately. The forward gap (Section 4.3.1) is defined independently of any loss function and therefore applies to both inference and training, whereas the backward gap (Section 4.3.2) arises only during training.

4.3.1. Forward Gap: Missing Representations

Under standard distributed GNN training or inference, client c can compute representations using only the features available locally. Since the features $\mathbf{x}_u$ of remote nodes $u \in V_c^{\text{rem}}$ are unavailable, client c cannot initialize their representations $\mathbf{h}_u^{(0)}$, producing local representations that diverge from their centralized counterparts.

Beyond degrading representation quality, missing neighbors can make certain pattern structures undetectable. When a pattern's defining substructure spans multiple clients, local learning becomes insufficient to recover the missing information. For example, a directed cycle split across two clients appears as an open path on each. A shared model, whether obtained by parameter aggregation or supplied

directly for inference, synchronizes *what* the model has learned, but cannot restore *what* each client can see.

We formalize the forward gap as a *representation equivalence gap*: the layer-L deviation between what client c computes at node v and what a centralized model would produce. Recall from the notation that $\hat{\mathbf{h}}_{v,c}^{(l)}$ denotes the naive client-local representation, in which client c aggregates over $\mathcal{N}(v)$ as granted by (A1), but substitutes the placeholder $\mathbf{h}_{\text{rem}} = \mathbf{0}$ for every remote embedding. This isolates the effect of missing remote features and embeddings from the effect of missing cross-client edges: the client sees the connectivity, but not what is on the other end. Since the update is recursive, the substitution compounds across layers. After L layers, $\hat{\mathbf{h}}_{v,c}^{(L)}$ depends on (i) the features of owned nodes within v's L-hop neighborhood in the owned subgraph (V_c^{own}, E_c), and (ii) the placeholder $\mathbf{h}_{\text{rem}}$ inserted at every cross-client edge encountered within L hops. The representation equivalence gap at node v on client c after L layers is

$$\Delta_{v,c}^{(L)} = \mathbf{h}_v^{(L,\text{cent})} - \hat{\mathbf{h}}_{v,c}^{(L)}.$$

For nodes at partition boundaries, $\Delta_{v,c}^{(L)} \neq \mathbf{0}$ in general, and its magnitude grows with the fraction of remote neighbors and the depth of the network.

4.3.2. Backward Gap: Missing Adjoint Signal

The backward gap concerns the adjoint signal that the backward pass propagates in reverse along every edge of the computation graph and, unlike the forward gap, exists only during training.

In the centralized computation, each edge (u, w) that carried the forward activation $\mathbf{h}_u^{(l)}$ into the update of $\mathbf{h}_w^{(l+1)}$ also carries, in reverse, the adjoint contribution that w returns to u. Concretely, the layer-l centralized adjoint at node u accumulates one contribution from each node w whose layer-$(l+1)$ representation depends on $\mathbf{h}_u^{(l)}$, namely u itself and its neighbors $\mathcal{N}(u)$:

$$\boldsymbol{\delta}_u^{(l),\text{cent}} = \left(\frac{\partial \mathbf{h}_u^{(l+1)}}{\partial \mathbf{h}_u^{(l)}}\right)^\top \boldsymbol{\delta}_u^{(l+1),\text{cent}} + \sum_{w \in \mathcal{N}(u)} \left(\frac{\partial \mathbf{h}_w^{(l+1)}}{\partial \mathbf{h}_u^{(l)}}\right)^\top \boldsymbol{\delta}_w^{(l+1),\text{cent}}. \tag{4.2}$$

When an owned node $u \in V_{c'}^{\text{own}}$ has a remote neighbor $w \in V_{c'}^{\text{rem}}$, the term routed back from w in Equation 4.2 is unavailable to client c': both the representation $\mathbf{h}_w^{(l+1)}$ and its adjoint $\boldsymbol{\delta}_w^{(l+1)}$ live on another client. Dual to the forward placeholder $\mathbf{h}_{\text{rem}} = \mathbf{0}$, the naive client-local backward pass fills this gap with the placeholder adjoint $\boldsymbol{\delta}_{\text{rem}} = \mathbf{0}$, so the owner c' accumulates only the contributions routed back from its owned nodes and from its own loss $\hat{\mathcal{L}}_{c'}$. The information that the centralized computation would send back across the cross-client edge is discarded.

We formalize the backward gap as an *adjoint equivalence gap*: the layer-l deviation between the centralized adjoint at node u and the adjoint its owner c' accumulates is

$$\boldsymbol{\Gamma}_{u,c'}^{(l)} = \boldsymbol{\delta}_u^{(l),\text{cent}} - \hat{\boldsymbol{\delta}}_u^{(l)}.$$

Two effects contribute to $\boldsymbol{\Gamma}_{u,c'}^{(l)}$. First, the entire adjoint contributions from remote neighbors are dropped and replaced by $\boldsymbol{\delta}_{\text{rem}} = \mathbf{0}$. This is the direct dual of the forward placeholder $\mathbf{h}_{\text{rem}}$. Second, because the recursion is evaluated at the naive client-local representations $\hat{\mathbf{h}}_{u,c'}^{(l)}$ rather than their centralized counterparts, the layer-l forward gap $\Delta_{u,c'}^{(l)} = \mathbf{h}_u^{(l,\text{cent})} - \hat{\mathbf{h}}_{u,c'}^{(l)}$ propagates into the backward pass: the derivative factors $\partial \mathbf{h}^{(l+1)} / \partial \mathbf{h}^{(l)}$ in Equation 4.2 depend on the point at which they are evaluated, so computing them at the inexact forward representations yields incorrect adjoint contributions even at nodes with no remote neighbors of their own, wherever their forward representations are already inexact.

Beyond these per-layer effects, the missing adjoint at one layer propagates into the next, mirroring the forward substitution, so $\boldsymbol{\Gamma}_{u,c'}^{(l)}$ compounds as the backward pass descends through the layers.

For nodes at partition boundaries, $\mathbf{\Gamma}_{u,c'}^{(l)} \neq \mathbf{0}$ in general, and its magnitude grows with the fraction of remote neighbors and the depth of the network. Because layer-l adjoints feed the parameter gradients, the gap propagates from representations to parameters: the gradient assembled from the local objectives, $\sum_c \partial \hat{\mathcal{L}}_c / \partial \Theta$, deviates from the centralized gradient $\partial \mathcal{L}^{\text{cent}} / \partial \Theta$, even when (A2) guarantees that every client differentiates the same parameters Θ.

5

Layer-Wise Exchange Framework

We address the structural observability problem of Section 4.3 with a layer-wise exchange protocol that synchronizes information across clients at every layer of the GNN, without a central server. The protocol has two components, addressing the two gaps. The *forward exchange* (Section 5.1) synchronizes node representations during the forward pass and closes the forward gap, recovering centralized representations exactly (Theorem 5.1). The *backward exchange* (Section 5.2) synchronizes adjoint signals during the backward pass and closes the backward gap, recovering centralized gradients exactly (Theorem 5.2). Together, under the assumptions of Section 4.2 extended with per-layer exchange, they make federated training equivalent to centralized training.

5.1. Forward Exchange

We first synchronize representations in the forward pass, then prove this recovers the centralized forward computation exactly.

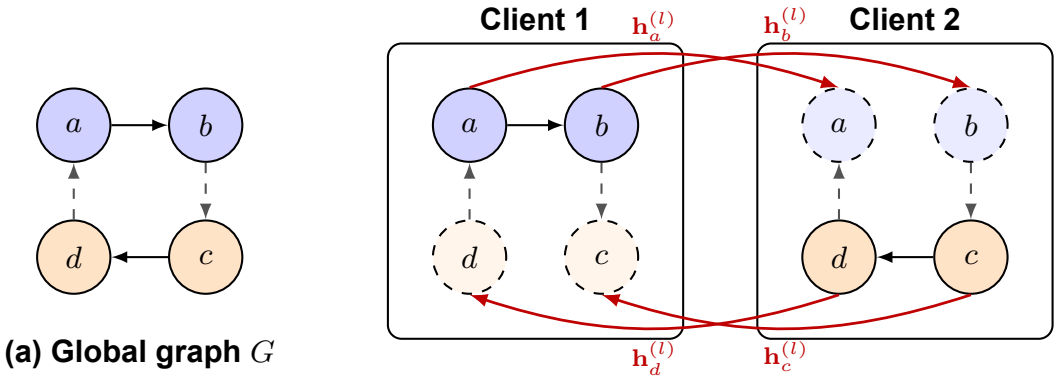
5.1.1. Protocol

We call an owned node $u \in V_c^{\text{own}}$ a *boundary node* of client c if $u \in V_{c'}^{\text{rem}}$ for some $c' \neq c$, i.e. if u lies on the other side of a cross-client edge owned by another client. After each layer l, every client sends the layer-l representations of its boundary nodes to the clients that hold them as remote and in return receives the layer-l representations of its own remote nodes from their owners (Figure 5.1). No central server mediates the exchange. We formalize this synchronization as a third condition extending the assumptions (A1)–(A2) of Section 4.2:

(A3) After each layer l, every client c receives $\mathbf{h}_u^{(l)}$ for all remote nodes $u \in V_c^{\text{rem}}$ from their owning clients.

All clients advance in lockstep. Layer $l+1$ does not begin on any client until the layer-l exchange has completed for all clients. Intuitively, the protocol restores at every layer the representations that a centralized GNN would produce over the full graph. Concretely, under this protocol each client computes representations that match the centralized model exactly, $\mathbf{h}_{v,c}^{(L)} = \mathbf{h}_v^{(L,\text{cent})}$, eliminating the divergence captured by $\Delta_{v,c}^{(L)}$ in Section 4.3.

At layer l, the content of the exchange is the set of representations $\{\mathbf{h}_{u,c'}^{(l)} : u \in V_{c'}^{\text{own}}\}$ computed locally by each owning client c', indexed by $u \in V$. Client c receives only those entries for which $u \in V_c^{\text{rem}}$ and sets $\mathbf{h}_{u,c}^{(l)} \leftarrow \mathbf{h}_{u,c'}^{(l)}$. In particular, the raw features $\mathbf{x}_u$ and labels of remote nodes never leave their owner: at $l = 0$, the owner c' applies the input embedding locally to obtain $\mathbf{h}_{u,c'}^{(0)} = \phi_{\text{emb}}^{\theta_{\text{emb}}}(\mathbf{x}_u)$ and transmits only this learned representation. This preserves the information access stated in Section 4.2, where client c has no access to the features or labels of V_c^{rem}, while allowing (A3) to hold at every layer $l \geq 0$.



(a) Global graph G

(b) Distributed view with layer-wise exchange

Figure 5.1: Layer-wise exchange on a 4-cycle ($a \to b \to c \to d \to a$) split across two clients. **(a)** The global graph: node colors preview the partition (Client 1: blue, Client 2: orange); cross-client edges are dashed gray and intra-client edges are solid black. **(b)** The distributed view: each client materializes solid copies of its *owned* nodes and dashed copies of the *remote* endpoints; the same node appears at the same coordinates across panels for visual continuity. After every layer l, each owner broadcasts $\mathbf{h}_{u,c'}^{(l)}$ (red) to every client that holds u as a remote node, and all clients synchronize at this barrier before advancing to layer $l{+}1$. Raw features never leave the owning client.

The protocol maintains the following invariant: entering layer $l+1$ on any client c, every node $u \in V_c^{\text{own}} \cup V_c^{\text{rem}}$ has its layer-l representation materialized on c and equal to the representation computed by its owner under the shared parameters. This is precisely the inductive hypothesis that drives Theorem 5.1. We denote by ExchangeLayer(l) the one-shot collective in which every owner transmits its layer-l representations to each client holding the corresponding node as remote. Algorithm 1 gives the full protocol, and Section 5.1.2 shows that it recovers centralized representations exactly.

Algorithm 1 Forward Exchange (one forward pass, all clients in parallel)

Require: Extended subgraphs $\{G_c\}_{c=1}^{C}$, shared parameters $\Theta = \{\theta_{\text{emb}}\} \cup \{\theta^{(l)}\}_{l=0}^{L-1}$, depth L.

for each client c **in parallel do**
  $\mathbf{h}_{v,c}^{(0)} \leftarrow \phi_{\text{emb}}^{\theta_{\text{emb}}}(\mathbf{x}_v) \quad \forall v \in V_c^{\text{own}}$
end for
ExchangeLayer(0) ▷ each owner sends $\mathbf{h}_{u,c'}^{(0)}$ to every client c with $u \in V_c^{\text{rem}}$
for $l = 0, 1, \ldots, L-1$ **do**
  for each client c **in parallel do**
    for $v \in V_c^{\text{own}}$ **do**
      $\mathbf{h}_{v,c}^{(l+1)} \leftarrow \phi_{\theta^{(l)}}^{(l)}\left(\mathbf{h}_{v,c}^{(l)}, \bigoplus_{u \in \mathcal{N}(v)} \psi_{\theta^{(l)}}^{(l)}\left(\mathbf{h}_{v,c}^{(l)}, \mathbf{h}_{u,c}^{(l)}, \mathbf{e}_{uv}\right)\right)$
    end for
  end for
  ExchangeLayer$(l+1)$ ▷ barrier: all owners deliver $\mathbf{h}_{u,c'}^{(l+1)}$ to clients holding u as remote, before layer $l+2$
end for
return $\{\mathbf{h}_{v,c}^{(L)} : v \in V_c^{\text{own}}\}$ on each client c

Communication cost. Per-layer communication complexity is $O(\sum_c |V_c^{\text{rem}}| \cdot d)$, where d is the hidden width. The set transmitted by each owner c' is $\{\mathbf{h}_{u,c'}^{(l)} : u \in V_{c'}^{\text{own}} \cap \bigcup_{c \neq c'} V_c^{\text{rem}}\}$, i.e. only the owned nodes that some other client holds as remote.

5.1.2. Representation Equivalence

We now show that the forward exchange recovers centralized representations exactly. Under assumptions (A1)–(A3), every client's representation matches the one a centralized model would produce over the full graph.

Theorem 5.1 (Representation Equivalence)**.** *Under* (A1)–(A3) *and full-batch forward pass, for every client* c*, every owned node* $v \in V_c^{\text{own}}$*, and every layer* $l \in \{0, \dots, L\}$*:*

$$\mathbf{h}_{v,c}^{(l)} = \mathbf{h}_v^{(l,\text{cent})}.$$

Proof. By induction on the layer index l.

Base case ($l = 0$). For each client c and each owned node $v \in V_c^{\text{own}}$, the local initialization sets

$$\mathbf{h}_{v,c}^{(0)} = \phi_{\text{emb}}^{\theta_{\text{emb}}}(\mathbf{x}_v).$$

For each remote node $u \in V_c^{\text{rem}}$, u is owned by some client c', which computes $\mathbf{h}_{u,c'}^{(0)} = \phi_{\text{emb}}^{\theta_{\text{emb}}}(\mathbf{x}_u)$ locally. ExchangeLayer(0) then delivers this value to c under (A3), so $\mathbf{h}_{u,c}^{(0)} = \phi_{\text{emb}}^{\theta_{\text{emb}}}(\mathbf{x}_u)$. The centralized model applies the same input embedding under (A2), hence $\mathbf{h}_{u,c}^{(0)} = \mathbf{h}_u^{(0,\text{cent})}$ for every $u \in V_c^{\text{own}} \cup V_c^{\text{rem}}$.

Inductive hypothesis. Assume that for some $l \geq 0$, for all clients c and all nodes $u \in V_c^{\text{own}} \cup V_c^{\text{rem}}$ (both owned and remote),

$$\mathbf{h}_{u,c}^{(l)} = \mathbf{h}_u^{(l,\text{cent})}.$$

Inductive step ($l \to l+1$). Let $v \in V_c^{\text{own}}$ be any owned node. We need to show that client c computes $\mathbf{h}_{v,c}^{(l+1)} = \mathbf{h}_v^{(l+1,\text{cent})}$.

Step 1: Neighborhood availability. By the definition of V_c^{rem} and (A1), $\mathcal{N}(v) \subseteq V_c^{\text{own}} \cup V_c^{\text{rem}}$, so every neighbor of v in the global graph is present on client c either as an owned or a remote node.

Step 2: Correctness of neighbor embeddings. For each neighbor $u \in \mathcal{N}(v)$:

- If $u \in V_c^{\text{own}}$, then $\mathbf{h}_{u,c}^{(l)}$ is computed locally by client c and is exact by the inductive hypothesis.
- If $u \in V_c^{\text{rem}}$, then u is owned by some other client c', which computes $\mathbf{h}_{u,c'}^{(l)} = \mathbf{h}_u^{(l,\text{cent})}$ by the inductive hypothesis applied at client c'. By (A3), client c receives this exact embedding, so $\mathbf{h}_{u,c}^{(l)} = \mathbf{h}_u^{(l,\text{cent})}$.

In both cases, $\mathbf{h}_{u,c}^{(l)} = \mathbf{h}_u^{(l,\text{cent})}$ for all $u \in \mathcal{N}(v)$.

Step 3: Layer update equivalence. By (A2), client c applies the same layer functions $\psi_{\theta^{(l)}}^{(l)}$ and $\phi_{\theta^{(l)}}^{(l)}$ as the centralized computation. Since the self-embedding $\mathbf{h}_{v,c}^{(l)}$, all neighbor embeddings $\{\mathbf{h}_{u,c}^{(l)}\}_{u \in \mathcal{N}(v)}$, and the edge features $\{\mathbf{e}_{uv}\}_{u \in \mathcal{N}(v)}$ are identical to the centralized setting, substituting into Equation 4.1 gives:

$$\begin{aligned}
\mathbf{h}_{v,c}^{(l+1)} &= \phi_{\theta^{(l)}}^{(l)}\left(\mathbf{h}_{v,c}^{(l)}, \bigoplus_{u \in \mathcal{N}(v)} \psi_{\theta^{(l)}}^{(l)}\left(\mathbf{h}_{v,c}^{(l)}, \mathbf{h}_{u,c}^{(l)}, \mathbf{e}_{uv}\right)\right) \\
&= \phi_{\theta^{(l)}}^{(l)}\left(\mathbf{h}_v^{(l,\text{cent})}, \bigoplus_{u \in \mathcal{N}(v)} \psi_{\theta^{(l)}}^{(l)}\left(\mathbf{h}_v^{(l,\text{cent})}, \mathbf{h}_u^{(l,\text{cent})}, \mathbf{e}_{uv}\right)\right) \\
&= \mathbf{h}_v^{(l+1,\text{cent})}.
\end{aligned}$$

This establishes the claim for owned nodes at layer $l+1$. To re-establish the inductive hypothesis for remote nodes, note that each $u \in V_c^{\text{rem}}$ is owned by some client c', which computes $\mathbf{h}_{u,c'}^{(l+1)} = \mathbf{h}_u^{(l+1,\text{cent})}$ as an owned node by the argument just given. ExchangeLayer$(l+1)$ then delivers this value to c under (A3), so $\mathbf{h}_{u,c}^{(l+1)} = \mathbf{h}_u^{(l+1,\text{cent})}$. Hence the hypothesis holds for all $u \in V_c^{\text{own}} \cup V_c^{\text{rem}}$ at layer $l+1$.

Since the partition is exhaustive ($\bigcup_c V_c^{\text{own}} = V$), every node in G is owned by exactly one client. By induction, for all $l \in \{0, \dots, L\}$ and all owned nodes $v \in V_c^{\text{own}}$, $\mathbf{h}_{v,c}^{(l)} = \mathbf{h}_v^{(l,\text{cent})}$. □

Loss equivalence. Under the forward exchange, each client c forms its loss on the exchanged representations,

$$\mathcal{L}_c = \sum_{v \in V_c^{\text{own}}} \ell\Big(\mathbf{h}_{v,c}^{(L)},\, y_v\Big)\,,$$

in contrast to the naive client-local loss $\hat{\mathcal{L}}_c$ of Section 4.3.2, which is evaluated at the inexact representations $\hat{\mathbf{h}}_{v,c}^{(L)}$. Because Theorem 5.1 gives $\mathbf{h}_{v,c}^{(L)} = \mathbf{h}_v^{(L,\text{cent})}$ for every owned node and the partition $\{V_c^{\text{own}}\}_c$ is exhaustive,

$$\sum_{c=1}^{C} \mathcal{L}_c = \sum_{v \in V} \ell\Big(\mathbf{h}_v^{(L,\text{cent})},\, y_v\Big) = \mathcal{L}^{\text{cent}}.$$

The sum of the per-client losses is therefore exactly the centralized objective, which is the loss the backward exchange of Section 5.2 differentiates.

5.2. Backward Exchange

We now synchronize adjoints in the backward pass, then prove this recovers the centralized gradients exactly.

5.2.1. Protocol

The forward collective ExchangeLayer(l) is a linear operation: it copies each owner's layer-l representation to every client that holds the node as remote. By the standard reverse-mode rule, the adjoint of such a copy is a sum: a value fanned out to several clients in the forward pass has its gradient summed back from all of them in the backward pass. The backward exchange is therefore the adjoint of the forward exchange: after backpropagating through layer l, every client that holds a node u as remote returns to u's owner the partial adjoint it computed at u, and the owner sums these contributions (Figure 5.2). No central server mediates the exchange. We formalize this synchronization as a fourth condition extending the assumptions (A1)–(A3):

(A4) After backpropagating through layer l, every owner c' of a node u receives the partial adjoint $\boldsymbol{\delta}_{u,c}^{(l)}$ from every client c with $u \in V_c^{\text{rem}}$, and accumulates

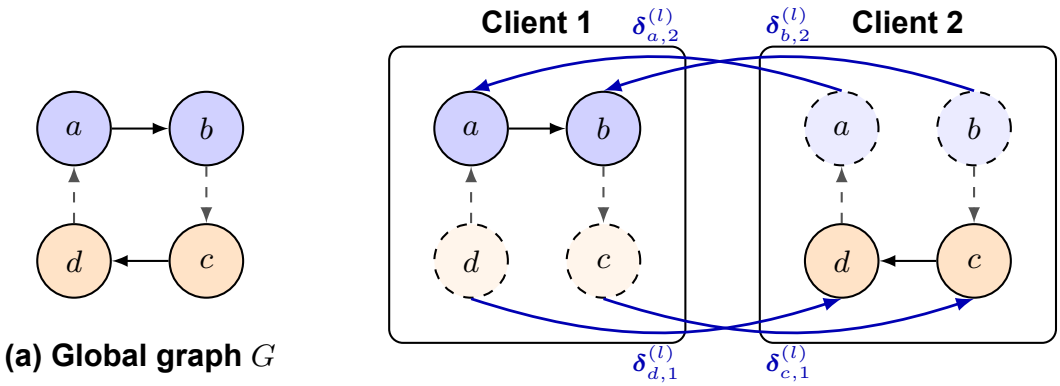
$$\boldsymbol{\delta}_u^{(l)} = \boldsymbol{\delta}_{u,c'}^{(l)} + \sum_{c:\, u \in V_c^{\text{rem}}} \boldsymbol{\delta}_{u,c}^{(l)}.$$

All clients advance in lockstep in decreasing layer order. The backward pass of layer $l-1$ does not begin on any client until the layer-l backward exchange has completed for all clients. Intuitively, the protocol restores at every layer the adjoint that the centralized backward pass would accumulate at each node. Concretely, under this protocol, each owner accumulates the adjoint that matches the centralized computation exactly, $\boldsymbol{\delta}_u^{(l)} = \boldsymbol{\delta}_u^{(l),\text{cent}}$, eliminating the deviation captured by $\boldsymbol{\Gamma}_{u,c'}^{(l)}$ in Section 4.3.2.

At layer l, each client c first receives the complete layer-$(l+1)$ adjoints $\boldsymbol{\delta}_v^{(l+1)}$ for its owned outputs $v \in V_c^{\text{own}}$, backpropagates one layer, and forms for each node u it holds the partial adjoint

$$\boldsymbol{\delta}_{u,c}^{(l)} = \sum_{\substack{v \in V_c^{\text{own}} \\ u \in \{v\} \cup \mathcal{N}(v)}} \Big(\partial \mathbf{h}_{v,c}^{(l+1)} / \partial \mathbf{h}_{u,c}^{(l)}\Big)^{\top} \boldsymbol{\delta}_v^{(l+1)},$$

the contribution to the adjoint at u routed through the receiving nodes that c owns. Each owner c' then accumulates these as in (A4): its own contribution $\boldsymbol{\delta}_{u,c'}^{(l)}$ collects the owner-side terms that the naive client-local adjoint $\hat{\boldsymbol{\delta}}_u^{(l)}$ of Section 4.3.2 retains, while the exchange supplies the cross-client terms $\sum_{c:\, u \in V_c^{\text{rem}}} \boldsymbol{\delta}_{u,c}^{(l)}$ that the naive computation discards. Because the forward exchange restores the exact representations and each client backpropagates the complete adjoints from above, every factor is evaluated at its centralized value. Together, the forward and backward exchange close the adjoint equivalence gap of Section 4.3.2.



(a) Global graph G

(b) Distributed view with backward exchange

Figure 5.2: Backward exchange on the same 4-cycle ($a \to b \to c \to d \to a$) split across two clients, dual to Figure 5.1. **(a)** The global graph, with the same partition (Client 1: blue, Client 2: orange); cross-client edges are dashed gray and intra-client edges are solid black. **(b)** The distributed view: after backpropagating through layer l, each client that holds u as a remote node returns its partial adjoint $\boldsymbol{\delta}^{(l)}_{u,c}$ (blue) to u's owner, which sums the returned contributions with its own to form the complete adjoint $\boldsymbol{\delta}^{(l)}_{u}$; all clients synchronize at this barrier before backpropagating to layer $l-1$. The exchange is the transpose of the forward broadcast in Figure 5.1: the arrows run from remote copies back to owners, and the forward copy becomes a backward sum. Only adjoint vectors cross client boundaries; raw features and labels never leave the owning client.

As in the forward exchange, only learned vectors cross client boundaries. The protocol transmits adjoint vectors, never the raw features or labels of any client's owned nodes, which determine $\mathcal{L}_c$ but remain local. This preserves the information access stated in Section 4.2 during the backward pass, mirroring the forward guarantee.

Entering the backward pass of layer $l-1$ on any client, every node u has its complete layer-l adjoint accumulated at its owner, equal to the adjoint the centralized backward pass would compute, $\boldsymbol{\delta}^{(l),\mathrm{cent}}_{u}$. This is precisely the reverse-induction hypothesis that drives Theorem 5.2. During the same pass, each owner c also accumulates its parameter-gradient contribution $g^{(l)}_{c} = \sum_{v \in V^{\mathrm{own}}_{c}} (\partial \mathbf{h}^{(l+1)}_{v,c} / \partial \theta^{(l)})^{\top} \boldsymbol{\delta}^{(l+1)}_{v}$ from its owned nodes. We denote by BackwardExchangeLayer(l) the one-shot collective in which every client holding a node as remote returns its layer-l partial adjoint to the owner, which sums the contributions. Algorithm 2 gives the full protocol, and Section 5.2.2 shows that it recovers centralized gradients exactly.

Algorithm 2 Backward Exchange (one backward pass, all clients in parallel)

Require: Forward activations $\mathbf{h}_{u,c}^{(l)}$ from Algorithm 1, shared parameters Θ, depth L, per-owner losses $\mathcal{L}_c = \sum_{v \in V_c^{\text{own}}} \ell\Big(\mathbf{h}_{v,c}^{(L)}, y_v\Big)$.

for each client c **in parallel do**
  $\boldsymbol{\delta}_{v,c}^{(L)} \leftarrow \partial \ell(\mathbf{h}_{v,c}^{(L)}, y_v)/\partial \mathbf{h}_{v,c}^{(L)}, \quad \forall v \in V_c^{\text{own}}$
end for
for $l = L-1, \dots, 0$ **do**
  for each client c **in parallel do**
    Backpropagate through layer l using the completed adjoints $\boldsymbol{\delta}_v^{(l+1)}$ at owned outputs $v \in V_c^{\text{own}}$, delivered by BackwardExchangeLayer$(l+1)$: accumulate $g_c^{(l)}$ and form partial adjoints $\boldsymbol{\delta}_{v,c}^{(l)}$ for $v \in V_c^{\text{own}}$ and $\boldsymbol{\delta}_{u,c}^{(l)}$ for the remote $u \in V_c^{\text{rem}}$ used as neighbors.
  end for
  BackwardExchangeLayer(l) ▷ adjoint of ExchangeLayer(l): every client returns $\boldsymbol{\delta}_{u,c}^{(l)}$ to owner c', which sets $\boldsymbol{\delta}_u^{(l)} \leftarrow \boldsymbol{\delta}_{u,c'}^{(l)} + \sum_{c:\, u \in V_c^{\text{rem}}} \boldsymbol{\delta}_{u,c}^{(l)}$
  barrier: all owners hold complete $\boldsymbol{\delta}^{(l)}$ before layer $l-1$
end for
Each owner c forms $g_c^{\text{emb}} = \sum_{v \in V_c^{\text{own}}} (\partial \mathbf{h}_{v,c}^{(0)}/\partial \theta_{\text{emb}})^\top \boldsymbol{\delta}_v^{(0)}$ from the completed $\boldsymbol{\delta}^{(0)}$.
return $g_c^{\text{emb}} + \sum_l g_c^{(l)}$ on each client (aggregate via the chosen server step)

Communication cost. Per-layer communication complexity is $O(\sum_c |V_c^{\text{rem}}| \cdot d)$, identical to the forward exchange, since each remote node returns one adjoint vector of width d to its owner. The set delivered to each owner c' is $\{\boldsymbol{\delta}_{u,c}^{(l)} : u \in V_{c'}^{\text{own}} \cap \bigcup_{c \neq c'} V_c^{\text{rem}}\}$, i.e. only the owned nodes that some other client holds as remote, which are exactly the boundary nodes of c'.

5.2.2. Gradient Equivalence

We now show that the backward exchange recovers centralized gradients exactly. Under assumptions (A1)–(A4), every node's accumulated adjoint matches the centralized adjoint, and the summed per-client parameter gradient matches the centralized gradient.

Theorem 5.2 (Gradient Equivalence)**.** *Under* (A1)–(A4) *and full-batch training, for every node $u \in V$ and every layer $l \in \{0, \dots, L\}$, the adjoint accumulated at u's owner equals the centralized adjoint,*

$$\boldsymbol{\delta}_u^{(l)} = \boldsymbol{\delta}_u^{(l),\text{cent}},$$

and consequently the parameter gradients accumulated across clients sum to the centralized gradient for every parameter in Θ. For each layer $l \in \{0, \dots, L-1\}$,

$$\sum_c g_c^{(l)} = \frac{\partial \mathcal{L}^{\text{cent}}}{\partial \theta^{(l)}}, \qquad g_c^{(l)} := \sum_{v \in V_c^{\text{own}}} \left(\frac{\partial \mathbf{h}_{v,c}^{(l+1)}}{\partial \theta^{(l)}}\right)^\top \boldsymbol{\delta}_v^{(l+1)},$$

and likewise

$$\sum_c g_c^{\text{emb}} = \frac{\partial \mathcal{L}^{\text{cent}}}{\partial \theta_{\text{emb}}},$$

for the input embedding, where $g_c^{(l)}$ (resp. g_c^{emb}) is the gradient client c accumulates for $\theta^{(l)}$ (resp. θ_{emb}) from its owned nodes.

Proof. Because the backward exchange is the exact reverse-mode adjoint of a forward pass that already reproduces the centralized computation (Theorem 5.1), gradient equivalence is the correctness

of reverse-mode differentiation applied to the distributed decomposition. The reverse induction below makes this precise: it first shows the accumulated adjoint matches the centralized adjoint at every node and layer, from which the parameter-gradient identity follows directly. By reverse induction on l, using Theorem 5.1, which gives exact forward activations $\mathbf{h}_{u,c}^{(l)} = \mathbf{h}_u^{(l,\mathrm{cent})}$ on every client.

Base case ($l = L$). For each client c and each owned node $v \in V_c^{\mathrm{own}}$, client c computes its output adjoint from its own loss term, $\boldsymbol{\delta}_{v,c}^{(L)} = \partial \ell(\mathbf{h}_{v,c}^{(L)}, y_v)/\partial \mathbf{h}_{v,c}^{(L)}$. By the loss equivalence $\sum_c \mathcal{L}_c = \mathcal{L}^{\mathrm{cent}}$ established after Theorem 5.1, node v enters $\mathcal{L}^{\mathrm{cent}}$ only through this same term $\ell(\cdot, y_v)$, so $\boldsymbol{\delta}_v^{(L),\mathrm{cent}} = \partial \ell(\mathbf{h}_v^{(L,\mathrm{cent})}, y_v)/\partial \mathbf{h}_v^{(L,\mathrm{cent})}$. By Theorem 5.1, $\mathbf{h}_{v,c}^{(L)} = \mathbf{h}_v^{(L,\mathrm{cent})}$, hence $\boldsymbol{\delta}_{v,c}^{(L)} = \boldsymbol{\delta}_v^{(L),\mathrm{cent}}$. The output adjoint is local to each owner, so no exchange is required at layer L and the complete adjoint coincides with it: $\boldsymbol{\delta}_v^{(L)} = \boldsymbol{\delta}_v^{(L),\mathrm{cent}}$.

Inductive hypothesis. Assume that for some $l+1 \leq L$, every node u has $\boldsymbol{\delta}_u^{(l+1)} = \boldsymbol{\delta}_u^{(l+1),\mathrm{cent}}$ accumulated at its owner.

Inductive step ($l+1 \to l$). Each client c backpropagates through layer l using the complete adjoints of its owned outputs. For each $v \in V_c^{\mathrm{own}}$, the complete adjoint $\boldsymbol{\delta}_v^{(l+1)}$ is held by c (its owner) after BackwardExchangeLayer$(l+1)$ and equals $\boldsymbol{\delta}_v^{(l+1),\mathrm{cent}}$ by the inductive hypothesis. Applying reverse-mode through the layer-l update yields, for each input node $u \in V_c^{\mathrm{own}} \cup V_c^{\mathrm{rem}}$,

$$\boldsymbol{\delta}_{u,c}^{(l)} = \sum_{\substack{v \in V_c^{\mathrm{own}} \\ u \in \{v\} \cup \mathcal{N}(v)}} \left(\frac{\partial \mathbf{h}_{v,c}^{(l+1)}}{\partial \mathbf{h}_{u,c}^{(l)}} \right)^{\top} \boldsymbol{\delta}_v^{(l+1)}.$$

By the inductive hypothesis and Theorem 5.1, every Jacobian and adjoint above equals its centralized value, so $\boldsymbol{\delta}_{u,c}^{(l)}$ is exactly the centralized recursion restricted to the terms whose receiving node v is owned by c.

Summing over all clients via (A4), the owner c' of u obtains

$$\boldsymbol{\delta}_u^{(l)} = \sum_{\text{clients } c} \sum_{\substack{v \in V_c^{\mathrm{own}} \\ u \in \{v\} \cup \mathcal{N}(v)}} \left(\frac{\partial \mathbf{h}_v^{(l+1)}}{\partial \mathbf{h}_u^{(l)}} \right)^{\top} \boldsymbol{\delta}_v^{(l+1),\mathrm{cent}}.$$

Because the partition is exhaustive ($\bigcup_c V_c^{\mathrm{own}} = V$), the union over clients of $\{v \in V_c^{\mathrm{own}} : v = u \text{ or } u \in \mathcal{N}(v)\}$ is $\{u\} \cup \{w : u \in \mathcal{N}(w)\} = \{u\} \cup \mathcal{N}(u)$, where the last equality uses the symmetry of $\mathcal{N} = \mathcal{N}^{\mathrm{in}} \cup \mathcal{N}^{\mathrm{out}}$. Hence

$$\boldsymbol{\delta}_u^{(l)} = \left(\frac{\partial \mathbf{h}_u^{(l+1)}}{\partial \mathbf{h}_u^{(l)}} \right)^{\top} \boldsymbol{\delta}_u^{(l+1),\mathrm{cent}} + \sum_{w \in \mathcal{N}(u)} \left(\frac{\partial \mathbf{h}_w^{(l+1)}}{\partial \mathbf{h}_u^{(l)}} \right)^{\top} \boldsymbol{\delta}_w^{(l+1),\mathrm{cent}} = \boldsymbol{\delta}_u^{(l),\mathrm{cent}},$$

matching Equation 4.2. This completes the induction.

Parameter gradient. The parameter $\theta^{(l)}$ produces the layer-$(l+1)$ representations, so the centralized gradient is

$$\frac{\partial \mathcal{L}^{\mathrm{cent}}}{\partial \theta^{(l)}} = \sum_{v \in V} \left(\frac{\partial \mathbf{h}_v^{(l+1)}}{\partial \theta^{(l)}} \right)^{\top} \boldsymbol{\delta}_v^{(l+1),\mathrm{cent}}.$$

Each node v is produced by its owner c, which holds the complete adjoint $\boldsymbol{\delta}_v^{(l+1)} = \boldsymbol{\delta}_v^{(l+1),\mathrm{cent}}$ after BackwardExchangeLayer$(l+1)$. Hence

$$g_c^{(l)} = \sum_{v \in V_c^{\text{own}}} \left(\frac{\partial \mathbf{h}_{v,c}^{(l+1)}}{\partial \theta^{(l)}} \right)^\top \boldsymbol{\delta}_v^{(l+1)}$$

equals the centralized terms restricted to $v \in V_c^{\text{own}}$, and since the partition is exhaustive,

$$\sum_c g_c^{(l)} = \frac{\partial \mathcal{L}^{\text{cent}}}{\partial \theta^{(l)}}.$$

The same argument applied with the recovered layer-0 adjoint $\boldsymbol{\delta}_v^{(0)}$ gives $\sum_c g_c^{\text{emb}} = \partial \mathcal{L}^{\text{cent}} / \partial \theta_{\text{emb}}$, with $g_c^{\text{emb}} = \sum_{v \in V_c^{\text{own}}} (\partial \mathbf{h}_{v,c}^{(0)} / \partial \theta_{\text{emb}})^\top \boldsymbol{\delta}_v^{(0)}$, so the recovered gradient covers all of Θ. □

6

Methodology

This chapter specifies the implementation choices across the methods evaluated in Chapter 7. Section 6.1 lists settings shared across all methods together with a per-method summary of federated participation and aggregation frequency. The remaining sections describe per-method specifics. The general formal definitions of FedAvg and Sync-SGD are given in Sections 2.3.1 and 2.3.2, and the layer-wise exchange protocol is defined in Chapter 5.

6.1. Training Configuration

All methods use the same 6-layer Multi-PNA backbone described in Section 7.1, trained with Adam, the same neighbor-sampling fanout schedule, and the same per-task positive class weighting. Table 6.1 lists the shared values. All experiments are repeated over two seeds (base_seed, base_seed$+1$) and results are reported as mean$\pm$std.

Table 6.1: Shared training hyperparameters across all methods.

Parameter	**Value**
Multi-PNA layers L	6
Hidden dimension d	64
Ego ID embedding dim	32
Port embedding dim d_p	8
Dropout	0.1
Optimizer	Adam
Learning rate	0.001
Weight decay	0.0001
Batch size B	32
Fanout (per layer)	$[10, 10, 10, 10, 5, 5]$
Per-task positive class weight	$w_t^+ = N_t^- / N_t^+$
Local epochs E	1
Global rounds R	100
Base seed	0

For FedAvg-style methods (FedAvg, FedAvg + Fwd-LE, FedAvg + Fwd/Bwd-LE, and OptimES), $R \cdot E = 100$ corresponds to 100 aggregation events. For Sync-SGD, Sync-SGD + Fwd-LE, and Sync-SGD + Fwd/Bwd-LE, the same budget is enforced by training for $R \cdot E$ passes through each client's local data, with one synchronous gradient step per mini-batch. For Fully Local, Fully Local + Fwd-LE, and Fully Local + Fwd/Bwd-LE, where no aggregation occurs, each client trains independently for $R \cdot E$ local-epoch equivalents.

The compared methods also differ along two axes that Table 6.1 does not capture: the client fraction q (share of clients participating in each communication event) and the frequency at which parameters or embeddings are aggregated. Table 6.2 summarizes both. The per-method specifics are documented in the respective sections of this chapter (Sections 6.2 to 6.6).

Table 6.2: Per-method federated participation and aggregation frequency.

Method group	Client fraction q	Aggregation Frequency
Fully Local, + Fwd-LE, + Fwd/Bwd-LE	–	none
FedAvg, + Fwd-LE, + Fwd/Bwd-LE	1.0	per epoch
OptimES	1.0	per epoch
Sync-SGD, + Fwd-LE, + Fwd/Bwd-LE	1.0	per step

The training loss is the binary cross-entropy averaged over the seven pattern-based tasks (see Chapter 3) and the seed nodes of the current mini-batch $\mathcal{B}$:

$$\mathcal{L} = \frac{1}{|\mathcal{B}|} \sum_{v \in \mathcal{B}} \ell(\hat{\mathbf{y}}_v, \mathbf{y}_v) \,,$$

where $\hat{\mathbf{y}}_v \in \mathbb{R}^7$ is the predicted logit vector and $\mathbf{y}_v \in \{0,1\}^7$ is the label vector.

6.2. Fully Local Training

Each client trains an independent Multi-PNA on its local subgraph G_c, with no parameter or representation sharing.

Local degree histograms. PNA's degree-aware scalers are calibrated per client. Client c computes its own mean in-degree

$$\bar{\delta}_c = \frac{1}{|V_c|} \sum_{v \in V_c} \delta_v^{(c)},$$

where $\delta_v^{(c)}$ is the in-degree of v in G_c. Inclusion of cross-client edges introduces remote nodes with different connectivity patterns, so $\bar{\delta}_c$ generally differs across clients and from the global $\bar{\delta}$.

Local port embedding tables. Port indices are assigned locally per client based on the edge multiplicity within G_c. The in-port table on client c has vocabulary size $P_c^{\text{in}} = \max_{v \in V_c} \delta_v^{\text{in},(c)}$ and the out-port table $P_c^{\text{out}} = \max_{v \in V_c} \delta_v^{\text{out},(c)}$. These sizes vary across clients with the local maximum degree.

Training scope. Mini-batch sampling draws seed nodes from V_c^{own} only. The local loss on client c is

$$\mathcal{L}_c = \frac{1}{|V_c^{\text{own}}|} \sum_{v \in V_c^{\text{own}}} \ell(\hat{\mathbf{y}}_v, \mathbf{y}_v).$$

Remote nodes V_c^{rem} participate in neighborhood aggregation during the forward pass, as granted by the extended subgraph assumption (A1), but are excluded from the loss.

Model selection and evaluation. Each client selects its own best checkpoint based on macro PR-AUC on its local validation subgraph, and is evaluated on its own test subgraph at test time. The reported metric is the unweighted average of per-client test scores:

$$\text{Score} = \frac{1}{C} \sum_{c=1}^{C} \text{PR-AUC}_c^{\text{test}}.$$

6.3. FedAvg

Local training on each client follows Section 6.2. The following additions handle cross-client aggregation.

Global round structure. Training proceeds over R global rounds. At the start of round r, the server samples $\mathcal{S}^{(r)} \subseteq \{1, \ldots, C\}$ with $|\mathcal{S}^{(r)}| = \lceil q \cdot C \rceil$ and client fraction $q = 1.0$, broadcasts the current $\Theta^{(r)}$, and each selected client performs E local epochs before returning $\Theta_c^{(r)}$.

Server aggregation. Aggregation uses the sample-weighted form of Equation (2.7), with weights $|V_c^{\text{own}}|$. Port-embedding parameters are excluded from aggregation because their vocabulary sizes differ across clients (Section 6.2). Each client retains its own port-embedding tables across rounds. Degree histograms similarly remain per-client and are not aggregated.

Model selection and evaluation. The best global round r^* is selected based on macro PR-AUC averaged across all clients' validation subgraphs. The selected global model is evaluated on each client's test subgraph and the reported metric is

$$\text{Score} = \frac{1}{C} \sum_{c=1}^{C} \text{PR-AUC}_c^{\text{test}}\left(\Theta^{(r^*)}\right).$$

6.4. Sync-SGD

Sync-SGD replaces the per-epoch parameter averaging of FedAvg with per-step gradient aggregation, following Section 2.3.2.

Single shared optimizer state. All clients hold the same parameters at every step. In our implementation, C task instances are created (one per client, for its own train loader and loss), but the model and optimizer of clients $c > 1$ are overwritten with those of client 1 so that there is exactly one Adam state across the federation. The choice is required because Adam's first- and second-moment estimates are not shared across separate per-client optimizers, and mixing them would break the single-global-optimizer semantics.

Per-step aggregation. Each step zeros the shared gradient buffer once, every client performs a forward pass on its own mini-batch and a sample-count-weighted backward pass (so the accumulated gradient equals the sample-weighted average of per-client gradients, consistent with Equation (2.9)), and then one optimizer step is applied to the shared parameters. All clients participate every step ($q = 1.0$).

Port embeddings, degree histograms, model selection. Port embeddings and degree histograms remain per-client, as in Section 6.3. The best global step is selected based on macro PR-AUC averaged across clients' validation subgraphs.

6.5. Layer-Wise Exchange Variants

The layer-wise exchange protocol of Chapter 5 is combined with each parameter-aggregation regime above, in two variants. The *forward-only* variants (+ Fwd-LE) implement the forward exchange of Section 5.1 alone; the *joint forward/backward* variants (+ Fwd/Bwd-LE) additionally implement the backward exchange of Section 5.2.

6.5.1. Forward-only variants (+ Fwd-LE)

The forward-only variants share a single `EmbeddingTable` of shape $[L, |V|, d]$, reset before every training step, so only embeddings produced under the current parameters are written and there is no cross-step staleness. Remote embeddings are read from the table with the autograd graph cut at the table boundary (writes are detached), so gradients flow only through each client's locally owned nodes.

Fully Local + Fwd-LE. Layer-wise exchange is applied on top of the fully-local procedure of Section 6.2, where no parameter aggregation is performed. Each client selects and evaluates its own model independently. This setting deliberately violates (A2): client parameters diverge over training, so the exchanged embeddings are computed under client-specific weights.

FedAvg + Fwd-LE. Layer-wise exchange is applied on top of the FedAvg procedure of Section 6.3 using the same client fraction $q = 1.0$. Synchronous per-step exchange requires every client to participate at every step, so partial participation is incompatible with the protocol. The remaining settings (sample-weighted aggregation, port embeddings excluded, per-client degree histograms, global-model selection) similarly follow Section 6.3.

Sync-SGD + Fwd-LE. Layer-wise exchange is applied on top of the Sync-SGD procedure of Section 6.4. Sync-SGD enforces parameter equivalence for the shared Multi-PNA backbone at each gradient step and operates with $q = 1.0$, so no further change is required.

6.5.2. Joint forward/backward variants (+ Fwd/Bwd-LE)

The backward-coupled variants replace the detached `EmbeddingTable` with a live, autograd-tracked on-device cache (`LiveEmbeddingCache`) and replace the per-client `loss.backward()` with a single combined backward,

$$\left(\sum_c \mathcal{L}_c\right) \texttt{.backward()},$$

per step. Because remote embeddings are now spliced in without detaching, autograd's chain rule sums every consumer's $\partial \mathcal{L}_c / \partial \mathbf{h}_u^{(l)}$ at the shared owned-embedding tensor and continues backward through the owning client's parameters, implementing the backward exchange of Section 5.2 so that each owner's gradient reflects all consumers' losses. The forward-exchange semantics (per-step reset, write-then-inject ordering, local fallback for unwritten remote nodes) are unchanged, and the live cache is reset every step, since autograd graphs cannot survive an optimizer step. Peak memory scales with the number of participating clients, because every client's forward activation graph is held until the single combined backward.

Fully Local + Fwd/Bwd-LE. The combined backward is applied without any parameter aggregation. Each client is initialized from its own seed and selects and evaluates its own model. As with Fully Local + Fwd-LE, this deliberately violates (A2): the coupled gradients are computed across client-specific weights.

FedAvg + Fwd/Bwd-LE. The combined backward is applied on top of the FedAvg procedure of Section 6.3 ($q = 1.0$, sample-weighted aggregation, port embeddings excluded, per-client degree histograms, global-model selection).

Sync-SGD + Fwd/Bwd-LE. The combined backward is applied on top of the Sync-SGD procedure of Section 6.4. Each client's loss is weighted by its owned-node count before summing, so the combined backward equals the sample-weighted gradient averaging of Equation (2.9), folded into one autograd call; the single shared Adam state is updated once per step.

6.6. OptimES

OptimES (Naman and Simmhan, 2026) is included as a per-epoch alternative to FedAvg + Fwd-LE (Section 6.5) that isolates the effect of exchange frequency. Parameter aggregation follows the FedAvg procedure of Section 6.3 (per-epoch sample-weighted averaging, with port-embedding tables and degree histograms kept per-client), and the client fraction is kept at $q = 1.0$ so that every client pushes its embeddings at every epoch boundary. OptimES differs only in adding per-epoch embedding exchange via the two-cache scheme described below.

Per-epoch exchange. Clients exchange layer-l representations $\mathbf{h}_{u,c}^{(l)}$ for $l \in \{1, \ldots, L-1\}$ only at epoch boundaries, rather than after every layer of every step as in Algorithm 1.

Two-cache scheme. Each client maintains two embedding tables. `cache_in` is read during the epoch to inject remote representations into the forward pass. Reads are gradient-detached, so the cache acts as frozen context within the epoch. `cache_out` is populated by an end-of-epoch push pass. At the start of the next epoch, the two caches are swapped, so reads see the most recently pushed values.

Pre-training seed pass. Before epoch 1, a pre-training forward pass populates `cache_in` once per session. Without this seed pass, the first epoch would read uninitialized cache entries.

Diverged-parameter push. The end-of-epoch push pass uses each client's *pre*-aggregation (still diverged) parameters, consistent with the original OptimES specification (Naman and Simmhan, 2026). Parameter aggregation occurs after the push.

Model selection and evaluation. Selection and evaluation follow Section 6.3. The best global round r^* is chosen by macro PR-AUC averaged across clients' validation subgraphs, and the resulting global model is evaluated on each client's test subgraph.

7

Experiments

This chapter empirically evaluates the methods introduced in Chapter 6, addressing the research questions posed in Chapter 1. Section 7.1 defines the experimental setup, Section 7.2 reports per-task results, and Section 7.3 examines scaling with the number of clients.

7.1. Experiment Setup

Dataset. We use directed multigraphs with seven injected structural patterns as labeled subgraphs: directed cycles of length 2–6 (**C2**–**C6**), scatter-gather (**S-G**), and biclique (**B-C**). Each of the train, validation, and test splits uses an independently generated graph drawn from the same distribution, preventing leakage across splits (see Chapter 3 for details).

GNN model and training. We adopt the Multi-PNA architecture of Egressy et al. (2024), which extends the Principal Neighborhood Aggregation (PNA) model (Corso et al., 2020) with reverse message passing, ego ID embeddings, and port numbering, which is a combination shown to be more expressive on directed multigraphs (see Section 2.1.3 for details). All experiments use a 6-layer instance of this model. Because the synthetic graphs carry no node or edge features, structural signal is supplied entirely by the ego and port embeddings; node features are initialized as the constant vector $\mathbf{x}_v = \mathbf{1}$ for all v. Each of the seven patterns corresponds to a binary node-classification head, trained jointly in a multi-label setting.

Metrics. All experiments are evaluated using minority-class PR-AUC (area under the precision–recall curve). For each task, PR-AUC is computed for the minority class. Minority class is set to be the positive class when positives are the minority and the negative class when positives are the majority. This makes the metric suitable for the imbalanced multi-label setting of this thesis, where most pattern labels are rare, while ensuring that tasks with a majority-positive class (Cycle-4, Cycle-5, and Cycle-6; see Table 3.1) are evaluated on their harder-to-detect class.

Compared to the minority-class F1 score, which is computed at a single decision threshold and can exhibit high variance under class imbalance, PR-AUC provides a threshold-independent and more stable measure of model performance. For the centralized Multi-PNA results reported in Appendix B, minority-class F1 scores are additionally reported alongside minority-class PR-AUC to enable direct comparison with prior work on this benchmark by Egressy et al., 2024.

Performance reference points. Table 7.1 reports two reference points used throughout the chapter. The centralized Multi-PNA, trained on the global graph without any partitioning, defines the per-task performance ceiling. The *always-positive* classifier, which predicts the positive class for every node, defines a trivial lower bound at 31.04% macro-averaged PR-AUC. Per-task its PR-AUC equals the minority-class prevalence $\min(P_t, N_t - P_t)/N_t$. The hyperparameter tuning results for the centralized Multi-PNA are documented in Appendix B.

Table 7.1: Reference points used throughout the experiments: per-task PR-AUC (%) of the centralized Multi-PNA trained on the global graph without partitioning (upper bound) and a trivial always-positive classifier (lower bound). The **Macro** column reports the mean across the seven structural tasks.

Method	C2	C3	C4	C5	C6	S-G	B-C	Macro
Reference points (no partitioning)								
Centralized Multi-PNA	99.63	99.60	99.68	93.51	88.57	99.34	99.30	97.09
Always-Positive	19.31	33.15	48.08	32.25	21.68	30.77	32.06	31.04

Federated splits. We partition the global graph using two community-aware strategies that induce qualitatively different cross-client connectivity profiles. *Louvain* (Blondel et al., 2008) maximizes modularity and therefore places partition boundaries along the graph's natural low-density seams, producing few cross-client edges per client. *METIS* (Karypis and Kumar, 1998) minimizes edge-cut subject to a hard balance constraint on partition size. The balance requirement of METIS forces cuts through denser regions when natural communities are unevenly sized, yielding higher cross-client edge density. In both cases, we retain cross-client edges in the client subgraphs (see section 4.2).

Labels. All experiments use *global labels*: pattern labels are computed once on the global graph and applied to each owned node unchanged, regardless of which client owns it.

Methods compared. We evaluate ten methods, grouped as three baselines, six layer-wise exchange variants built on top of them, and OptimES. The three baselines differ in how they share parameters. *Fully Local* trains an independent Multi-PNA on each client with no parameter or representation sharing. *FedAvg* (McMahan et al., 2016) aggregates parameters once per epoch (Section 2.3.1), while *Sync-SGD* aggregates gradients at every optimization step into a shared optimizer state (Section 2.3.2).

The proposed layer-wise exchange framework of Chapter 5 is then applied on top of each of these three regimes (Fully Local, FedAvg, Sync-SGD) in two variants: *forward-only exchange* (denoted + *Fwd-LE*), which synchronizes representations during the forward pass (Section 5.1), and *joint forward/backward exchange* (denoted + *Fwd/Bwd-LE*), which additionally synchronizes adjoints during the backward pass (Section 5.2). Combining the three base regimes with the two exchange variants yields the six exchange-enhanced methods.

Finally, *OptimES* (Naman and Simmhan, 2026) is the tenth method, included as a per-epoch alternative to *FedAvg* + *Fwd-LE*. By exchanging embeddings only at epoch boundaries, it isolates the impact of embedding freshness relative to the proposed per-step exchange mechanism. Implementation details of all evaluated methods are provided in Chapter 6.

Deviations from Theoretical Assumptions. The representation equivalence of Theorem 5.1 and the gradient equivalence of Theorem 5.2 hold under full-batch inference and exactly shared parameters and layer functions across clients (A2). The experimental implementation departs from these idealized conditions in three ways, so the theorems' assumptions are only partially satisfied.

First, full-batch inference is impractical on large graphs, so we train with mini-batch neighborhood sampling (Hamilton, Ying, and Leskovec, 2017). Each forward pass then operates on a sampled subgraph rather than the complete L-hop neighborhood, so the recovered representations match the centralized ones only within the sampled receptive field.

Second, the Multi-PNA degree scalers depend on a degree histogram computed locally on each client's subgraph. Because cross-client cuts alter local degree distributions, these histograms differ across clients, so the layer functions $\psi^{(l)}$ and $\phi^{(l)}$ are not exactly identical between clients even when their learnable weights are aggregated.

Third, the port embedding tables are client-specific and are not aggregated under either FedAvg or Sync-SGD. The shared parameters, which are the input projection, the message-passing layers, and the output head, are aggregated (per epoch under FedAvg, per step under Sync-SGD). However, the port embeddings remain local. Hence, the shared-parameter assumption (A2) holds for the aggregated weights and not for the full parameter set.

Together, these deviations mean that the experimental model does not fully satisfy the conditions of Theorems 5.1 and 5.2. The residual gap to the centralized reference reported in Section 7.2 reflects their combined effect, and we revisit its empirical magnitude there.

7.2. Main Results

We organize the empirical evaluation around the research questions introduced in Chapter 1. (i) For RQ1, we measure the federated-centralized gap under Fully Local and FedAvg training to characterize its origin. (ii) For RQ2, we compare Fully Local + Fwd-LE against FedAvg + Fwd-LE to test whether layer-wise exchange depends on parameter synchronization. (iii) For RQ3, we compare FedAvg + Fwd-LE against the per-epoch alternative OptimES (Naman and Simmhan, 2026) to understand the impact of per-step embedding freshness. (iv) For RQ4, we compare FedAvg + Fwd-LE and Sync-SGD + Fwd-LE to examine how the frequency of parameter aggregation (per-epoch versus per-step) affects the extent of recovery. (v) For RQ5, we compare each Fwd/Bwd-LE configuration against its Fwd-LE counterpart to quantify the additional recovery from backward exchange.

Together, these analyses address the main RQ by quantifying both the recovery achieved by layer-wise exchange and the residual gap that remains to the centralized reference. We additionally discuss two secondary findings: that strong parameter coupling alone (Sync-SGD without LE) partially substitutes for forward exchange, and that mini-batch sampling caps the achievable model performance.

Each claim is evaluated under *two* distinct partitioning regimes that induce qualitatively different cross-client structure: Louvain (modularity-based communities) and METIS (balanced k-way edge-cut). The per-task PR-AUC scores at $k = 15$ clients are reported in Table 7.2 and Table 7.3, respectively.

Table 7.2: PR-AUC (%) at 15 clients on **Louvain** split. LE = Layer-wise Exchange. Methods are grouped by the cross-client information exchanged and its freshness: no exchange; forward exchange, either per-epoch/stale (OptimES) or per-step/fresh (Fwd-LE); and joint forward/backward per-step exchange (Fwd/Bwd-LE). Δ columns are absolute differences from the indicated reference: Δ Local = vs. fully local; Δ FedAvg = vs. FedAvg.

Method	C2	C3	C4	C5	C6	S-G	B-C	Macro	Δ Local	Δ FedAvg
No Exchange										
Fully Local	43.50±0.17	64.15±0.25	83.57±0.15	80.94±0.18	81.17±0.16	69.74±0.39	71.21±0.20	70.61	–	–
FedAvg	56.48±4.29	70.52±1.86	85.15±0.45	78.42±0.58	74.49±1.81	75.09±0.75	76.57±0.16	73.82	+3.21	–
Sync-SGD	99.33±0.00	95.51±0.07	93.94±1.00	85.85±0.00	89.10±0.03	92.72±2.13	89.20±3.84	92.24	+21.63	+18.42
Forward Exchange — Per-Epoch (stale)										
OptimES	95.42±0.33	84.06±2.17	86.83±0.37	84.59±0.13	87.29±0.09	72.28±0.14	71.46±0.15	83.13	+12.52	+9.31
Forward Exchange — Per-Step (fresh)										
Fully Local + Fwd-LE	88.99±0.36	59.07±0.03	79.16±0.07	76.56±0.26	77.76±0.42	66.48±0.61	68.83±0.13	73.84	+3.23	+0.02
FedAvg + Fwd-LE	95.06±0.80	89.66±0.95	87.12±1.00	84.42±0.14	87.70±0.21	73.64±0.39	71.91±0.42	84.22	+13.61	+10.40
Sync-SGD + Fwd-LE	99.10±0.16	95.35±0.69	95.06±0.21	86.60±0.24	89.72±0.01	96.22±0.35	96.08±0.19	94.02	+23.41	+20.20
Forward + Backward Exchange — Per-Step (fresh)										
Fully Local + Fwd/Bwd-LE	43.76±0.27	64.30±0.20	83.54±0.03	81.91±0.07	82.95±0.34	69.60±0.06	70.78±0.34	70.98	+0.37	−2.84
FedAvg + Fwd/Bwd-LE	97.02±0.08	90.77±0.58	86.58±0.74	84.53±0.29	87.72±0.37	74.57±0.26	72.08±0.54	84.75	+14.14	+10.93
Sync-SGD + Fwd/Bwd-LE	99.38±0.10	96.33±0.19	95.26±0.03	86.61±0.43	89.61±0.27	96.67±0.28	95.91±0.22	94.25	+23.64	+20.43

The structural observability gap is empirically pronounced. The Fully Local and FedAvg rows of Tables 7.2 and 7.3 show that partitioning the global graph induces a representation equivalence gap between centralized and distributed models: macro PR-AUC drops from the centralized ceiling of 97.09% to 70.61% (Louvain) and 69.42% (METIS) under fully local training, and is only marginally recovered by FedAvg (73.82% and 71.93%, respectively). Parameter aggregation alone cannot close the gap.

The drop is most severe for C2 and C3 patterns, whose centralized detection achieves near-perfect performance. C2 falls from 99.63% to 43.50% (Louvain) / 48.96% (METIS) (≥ 50pp) and C3 from 99.60% to 64.15% (Louvain) / 63.76% (METIS) (≥ 35pp). For the remaining patterns, the drop is consistently smaller, ranging from ~ 30pp (S-G, B-C) down to ~ 7–12pp for C6. Notably, for the C6 pattern, the centralized ceiling itself (88.57%) bounds how large the gap could be. This asymmetry is due to graph partitioning: short cycles contain only a handful of edges, so a single cross-client cut suffices to destroy the entire pattern. Longer cycles retain a contiguous arc within each client, which a local model can still exploit as partial evidence.

Table 7.3: PR-AUC (%) at 15 clients on **METIS** split. LE = Layer-wise Exchange. Methods are grouped by the cross-client information exchanged and its freshness: no exchange; forward exchange, either per-epoch/stale (OptimES) or per-step/fresh (Fwd-LE); and joint forward/backward per-step exchange (Fwd/Bwd-LE). Δ columns are absolute differences from the indicated reference: Δ Local = vs. fully local; Δ FedAvg = vs. FedAvg.

Method	C2	C3	C4	C5	C6	S-G	B-C	Macro	Δ Local	Δ FedAvg
No Exchange										
Fully Local	48.96±3.95	63.76±0.45	80.00±0.37	78.06±0.27	76.73±0.71	68.68±0.51	69.72±0.42	69.42	–	–
FedAvg	61.05±3.76	68.51±1.50	80.48±0.67	76.71±1.01	71.32±2.81	71.41±0.07	74.06±0.59	71.93	+2.51	–
Sync-SGD	99.50±0.06	92.91±0.88	89.44±0.51	82.89±0.10	85.44±0.19	85.35±1.21	82.04±0.32	88.22	+18.80	+16.29
Forward Exchange — Per-Epoch (stale)										
OptimES	76.69±4.19	69.44±1.29	81.91±0.17	80.55±0.13	83.62±0.23	71.44±0.38	72.23±0.13	76.55	+7.13	+4.62
Forward Exchange — Per-Step (fresh)										
Fully Local + Fwd-LE	88.89±0.18	59.44±0.07	79.23±0.25	76.69±0.12	77.98±0.15	66.77±0.42	68.33±0.04	73.90	+4.48	+1.97
FedAvg + Fwd-LE	93.71±0.08	83.70±3.62	83.58±0.02	82.70±0.14	85.51±0.06	72.47±0.31	72.07±0.12	81.96	+12.54	+10.03
Sync-SGD + Fwd-LE	97.26±0.54	90.18±0.94	88.21±0.34	84.98±0.13	88.05±0.02	84.09±1.44	83.57±0.37	88.05	+18.63	+16.12
Forward + Backward Exchange — Per-Step (fresh)										
Fully Local + Fwd/Bwd-LE	41.91±0.17	62.66±0.06	80.96±0.06	80.12±0.49	80.11±0.66	68.94±0.09	71.06±0.03	69.39	−0.03	−2.54
FedAvg + Fwd/Bwd-LE	94.84±0.16	87.79±0.13	84.55±0.24	83.05±0.31	86.13±0.15	73.87±0.24	72.01±0.14	83.18	+13.76	+11.25
Sync-SGD + Fwd/Bwd-LE	99.04±0.18	96.05±0.57	93.94±0.56	85.76±0.09	89.03±0.24	94.48±1.03	93.74±0.15	93.15	+23.73	+21.22

Although the magnitude varies, every pattern incurs a measurable loss under partitioning, confirming that local unidentifiability is a consequence of the distributed setting rather than a pattern-specific artifact.

Forward exchange recovers the gap only when paired with parameter aggregation. Adding forward exchange to fully local training (Fully Local + Fwd-LE) lifts macro PR-AUC to 73.84% (Louvain) and 73.90% (METIS), on par with plain FedAvg ($\Delta_{\text{FedAvg}} = +0.02, +1.97$pp) but falling 8–10pp short of FedAvg + Fwd-LE. Without synchronized parameters, the exchanged embeddings are no longer those a centralized model would compute, so the shared-parameter assumption (A2) fails and the equivalence of Theorem 5.1 no longer applies.

The per-pattern view shows that forward exchange alone is not uniformly insufficient. For C2, Fully Local + Fwd-LE reaches 88.99% (Louvain) and 88.89% (METIS), up from 43.50% and 48.96% under plain Fully Local. This could be explained by the Cycle-2 pattern being recoverable from a single remote endpoint embedding, even one computed under diverged weights. C3 shows the opposite behavior: Fully Local + Fwd-LE actually underperforms plain Fully Local (59.07% vs. 64.15% on Louvain), because deeper structural reasoning amplifies parameter divergence and the exchanged embeddings inject contradictory signal rather than missing context. Adding parameter aggregation on top (FedAvg + Fwd-LE) lifts C3 back to 89.66% (Louvain) and 83.70% (METIS), confirming that parameter consistency is the missing ingredient.

Forward exchange and parameter aggregation are therefore complementary, not substitutable. Shallow patterns tolerate the violation of the shared-parameter assumption (A2) while the deeper ones require parameter consistency for the exchanged embeddings to be informative.

Forward exchange narrows the gap, and per-layer freshness is what does the work. Adding forward exchange to FedAvg (FedAvg + Fwd-LE) lifts macro PR-AUC to 84.22% (Louvain) and 81.96% (METIS), gains of $+10.40$ and $+10.03$pp over plain FedAvg.

The recovery is concentrated on short cycles: C2 jumps by $+38.58$pp on Louvain and $+32.66$pp on METIS, and C3 by $+19.14$pp and $+15.19$pp. These small cycles are the patterns most damaged by a single cross-client cut and the ones whose detection most directly recovers when remote endpoint embeddings become available. Longer cycles and the non-cycle patterns (S-G, B-C) gain less because their local arcs already provide partial evidence.

FedAvg + Fwd-LE also outperforms the stale per-epoch alternative, OptimES, by 1.09pp on Louvain and 5.41pp on METIS. The Louvain margin is narrow because partition boundaries lie along low-density seams, so few remote-node embeddings are ever read per step. On METIS, where edge-cut is denser, every step references many stale embeddings and their drift from the current parameters compounds. The result is consistent with the forward exchange assumption (A3): as cross-client connectivity densifies, freshness becomes a stricter requirement for the representation equivalence of Theorem 5.1.

Synchronizing parameters per-step closes the residual forward gap. Sync-SGD + Fwd-LE lifts macro PR-AUC to 94.02% (Louvain) and 88.05% (METIS), an additional $+9.80$ and $+6.09$pp over FedAvg + Fwd-LE. The residual gap left by FedAvg + Fwd-LE comes from within-epoch parameter drift: FedAvg synchronizes model parameters only at epoch boundaries, so embeddings written at step t are computed under client-specific parameters and the shared-parameter assumption (A2) fails between syncs. Sync-SGD synchronizes the aggregated weights at each step, which restores the shared-parameter condition of Theorem 5.1 for those weights within each sampled computation graph.

The per-pattern decomposition shows that the additional gain achieved by per-step parameter synchronization is concentrated on S-G and B-C patterns rather than on the cycles. On Louvain, S-G improves from 73.64% to 96.22% ($+22.58$pp) and B-C from 71.91% to 96.08% ($+24.17$pp) while only $+2$–8pp is observed on the cycle tasks. METIS shows the same shape with smaller absolute numbers: S-G and B-C gain $+11.62$ and $+11.50$pp, again the largest of any task.

The asymmetry mirrors the structural argument for the gap itself: cycles are destroyed by a single cut and recovered primarily through neighborhood completion (which FedAvg + Fwd-LE already supplies), whereas S-G and B-C remain partially visible in each local subgraph and are limited instead by embedding quality. Per-step parameter equivalence produces embeddings that better separate these higher-arity patterns, which is what closes the residual forward gap left by per-epoch aggregation.

Constant parameter synchronization trains a model with reduced reliance on forward exchange. Plain Sync-SGD without cross-client exchange outperforms FedAvg + Fwd-LE on both splits. It is within 1.78pp of Sync-SGD + Fwd-LE on Louvain, and matches the Sync-SGD + Fwd-LE performance on METIS. We attribute this result to the ability of Sync-SGD to acquire more transferable information from the clients and to better capture the global topology. As model capacity improves, reliance on forward exchange decreases, explaining plain Sync-SGD's strong performance. However, this reduced reliance is specific to the forward exchange; the backward exchange still carries cross-client signal that parameter synchronization alone does not capture.

Backward exchange completes the framework and helps most where cross-client coupling is densest. Adding backward exchange on top of forward exchange closes the adjoint equivalence gap, and thereby recovers the centralized gradient (Theorem 5.2). Across both parameter-aggregation regimes, it delivers consistent improvements over forward-only exchange. FedAvg + Fwd/Bwd-LE improves upon FedAvg + Fwd-LE by $+0.53$pp on Louvain ($84.22 \rightarrow 84.75$) and by $+1.22$pp on METIS ($81.96 \rightarrow 83.18$). Similarly, Sync-SGD + Fwd/Bwd-LE improves upon Sync-SGD + Fwd-LE by $+0.23$pp on Louvain ($94.02 \rightarrow 94.25$) and by $+5.10$pp on METIS ($88.05 \rightarrow 93.15$).

The largest improvement is observed for Sync-SGD on METIS ($+5.10$pp). This setting simultaneously maximizes the two conditions under which backward exchange is most effective. First, METIS's dense edge cut severs many cross-client edges, so the naive backward pass discards a substantial amount of gradient information that can be recovered through backward exchange. Second, Sync-SGD's per-step synchronization keeps the aggregated weights matched across clients, so the returned adjoints are accumulated under near-identical parameters. Under Sync-SGD, layer-wise exchange therefore reconstructs the centralized gradient closely in line with Theorem 5.2. Notably, Sync-SGD + Fwd/Bwd-LE is the only configuration that surpasses plain Sync-SGD on both partitioning strategies, achieving gains of $+2.01$pp on Louvain and $+4.93$pp on METIS.

These results indicate that the complete forward/backward exchange framework recovers learning signal that per-step parameter synchronization alone cannot capture. These gains, however, come at the communication cost associated with per-step exchange, which is analyzed in Chapter 8.

Backward exchange, like forward exchange, requires parameter synchronization. The effectiveness of backward exchange depends on parameter synchronization, even more so than forward exchange. In the fully local regime, backward exchange not only fails to provide additional benefit but reverses the gains achieved by forward exchange alone. Fully Local + Fwd/Bwd-LE drops to 70.98% on Louvain and 69.39% on METIS, performing worse than Fully Local + Fwd-LE (73.84% and 73.90%, respectively) and effectively reverting to the level of plain fully local training. The effect is most significant for C2 detection. While forward exchange alone raised C2 performance to 88.99% on Louvain

and 88.89% on METIS, adding backward exchange reduces performance to 43.76% and 41.91% respectively, matching the plain fully local baseline on Louvain (43.50%) and falling even below it on METIS (48.96%).

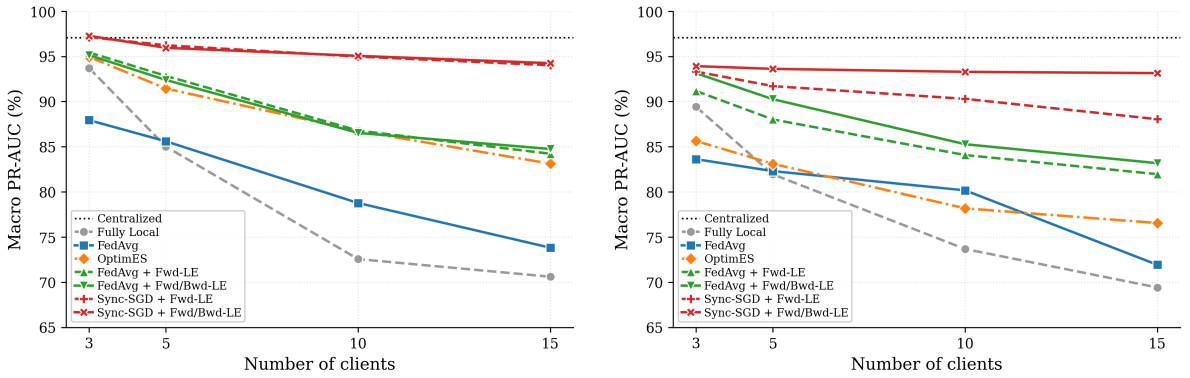
The underlying cause is the same violation of the assumption (A2) that limited forward exchange, now manifested during the backward pass. When clients maintain different parameter states, the owner of a remote representation accumulates adjoint contributions that were computed under inconsistent model weights. As a result, the gradient each owner accumulates no longer approximates the centralized gradient. Instead, it combines incompatible update directions and injects contradictory optimization signals. This is the backward-pass analog of the contradictory-embedding effect previously observed for Fully Local + Fwd-LE on C3. These results therefore reinforce the conclusion of this chapter: forward and backward exchange recover the centralized computation only when they are built upon a foundation of parameter consistency.

Mini-batch sampling builds robustness but limits the attainable ceiling. Mini-batch training introduces two opposing effects. On one hand, each forward pass operates on a stochastically sampled subgraph rather than the complete L-hop neighborhood, encouraging the model to learn robust representations under partial information. On the other hand, this same sampling inherently limits the maximum achievable accuracy. Because the receptive field observed during any forward pass is incomplete, layer-wise exchange can recover centralized representations only within the sampled receptive field, rather than across the full neighborhood assumed by Theorem 5.1. This sampling-induced incompleteness is one reason even the strongest configuration, Sync-SGD + Fwd/Bwd-LE, remains below the centralized benchmark (2.84pp on Louvain and 3.94pp on METIS).

The role of backward exchange further clarifies the nature of this remaining gap. On METIS, backward exchange closes most of the gap that forward-only exchange leaves, reducing the residual to the centralized setting from 9.04pp to 3.94pp. This shows that a large part of what forward-only exchange left unrecovered on the denser partition was due to the absence of gradient equivalence, which the backward exchange supplies, rather than to the deviations from the theorems' assumptions. The smaller gap that remains after both forward and backward exchange is consistent with those residual deviations identified in Section 7.1, incomplete mini-batch receptive fields together with the client-local degree statistics and port embeddings, though our experiments do not isolate their individual contributions.

7.3. Robustness to Federation Scale

Figure 7.1 sweeps the number of clients $k \in \{3, 5, 10, 15\}$ under both partitioning strategies. Across both Louvain and METIS, all four per-step exchange variants (FedAvg + Fwd-LE, FedAvg + Fwd/Bwd-LE, Sync-SGD + Fwd-LE, and Sync-SGD + Fwd/Bwd-LE) dominate the Fully Local and FedAvg baselines at every client count. The ordering established at $k = 15$ holds throughout the sweep: the Sync-SGD variants outperform the FedAvg variants, which in turn outperform plain FedAvg. The advantage of exchange widens as the federation grows: the Fully Local and FedAvg baselines degrade sharply with more clients, while the exchange-enhanced curves degrade far more gradually.



(a) Macro PR-AUC as a function of the number of clients under the **Louvain** partitioning scheme.

(b) Macro PR-AUC as a function of the number of clients under the **METIS** partitioning scheme.

Figure 7.1: Macro PR-AUC as the federation scales from 3 to 15 clients under two partitioning strategies: (a) Louvain and (b) METIS, both with cross-client edges. The dotted horizontal line indicates the centralized performance (97.09%), serving as a centralized reference.

The contrast between fresh per-step exchange and the stale per-epoch alternative tracks the partition-density argument made earlier. On Louvain, FedAvg + Fwd-LE and OptimES are nearly indistinguishable at every k, whereas on METIS, FedAvg + Fwd-LE consistently leads OptimES by a clear margin, with no narrowing as the federation grows. Per-layer freshness therefore becomes more important as cross-client connectivity densifies, at every federation scale.

Adding backward exchange preserves these trends while sharpening the advantage on the denser split. On Louvain, each Fwd/Bwd-LE curve tracks its Fwd-LE counterpart almost exactly at every k, consistent with the negligible backward gain observed on sparsely-cut partitions. On METIS, the Fwd/Bwd-LE curves sit consistently above their Fwd-LE counterparts, and for Sync-SGD the separation widens with k. Sync-SGD + Fwd/Bwd-LE stays nearly flat at $\sim 93\%$ across the entire range, while Sync-SGD + Fwd-LE declines to 88.05% at $k = 15$. Backward exchange thus makes Sync-SGD + Fwd/Bwd-LE the configuration least sensitive to federation scale.

8

Communication Cost Analysis

Chapter 7 quantified how much of the centralized performance each method recovers. This chapter quantifies what that recovery costs to communicate. We study the communication cost of every evaluated method *analytically*.

The chapter is organized as follows. Section 8.1 defines the notation, Section 8.2 derives the per-epoch communication volume of each method, and finally Section 8.3 orders the methods by it. The resulting ordering is what the accuracy-communication trade-off of Chapter 9 builds on.

8.1. Notation

Following the notation introduced in Chapter 4, let V_c^{own} be client c's owned node set and E_c^{rem} its set of cross-client edges, with cardinalities $N_c = |V_c^{\text{own}}|$ and $|E_c^{\text{rem}}|$. We measure embedding communication in units of cross-client edges: each edge in E_c^{rem} carries one d-dimensional embedding whenever its owned endpoint aggregates across it, so communication is counted as one embedding exchange per cross-client edge.

Let S be the number of mini-batch steps per local epoch, $B_c = N_c/S$ the batch size (equivalently, the number of seed nodes), L the number of message-passing layers, d the hidden dimension, and P the trainable parameter count. For the per-step exchange, let $R_c^{(s)}$ be the set of cross-client edges client c samples at step s (per layer). We write

$$|R_c| = \frac{1}{S}\sum_{s=1}^{S}|R_c^{(s)}|$$

for the average number of cross-client edges sampled per layer, per step. The per-epoch total of sampled edges is then $S\,|R_c|$.

Here L, d, P, and S are shared across clients, whereas N_c, B_c, $|E_c^{\text{rem}}|$, and $|R_c|$ are client-specific.

8.2. Per-method volumes

Table 8.1 summarizes the per-epoch communication volume of each evaluated method. The volumes follow directly from each method's communication pattern. FedAvg transmits the parameter vector once at each epoch boundary and therefore incurs only the parameter communication cost P. OptimES performs a single embedding exchange per epoch across all L layers, activating each cross-client edge once and contributing an embedding communication term of $L\,d\,|E_c^{\text{rem}}|$. FedAvg + Fwd-LE replaces this one-time exchange with a per-step exchange over the cross-client edges activated during mini-batch training. Consequently, the embedding term changes from $L\,d\,|E_c^{\text{rem}}|$ to $S\,L\,d\,|R_c|$.

Table 8.1: Per-epoch communication volume of each method, for client c.

Method	Per-epoch volume (client c)
FedAvg	P
OptimES	$P + L\,d\,\lvert E_c^{\mathrm{rem}}\rvert$
FedAvg + Fwd-LE	$P + S\,L\,d\,\lvert R_c\rvert$
FedAvg + Fwd/Bwd-LE	$P + 2\,S\,L\,d\,\lvert R_c\rvert$
Sync-SGD	$S\,P$
Sync-SGD + Fwd-LE	$S\,P + S\,L\,d\,\lvert R_c\rvert$
Sync-SGD + Fwd/Bwd-LE	$S\,P + 2\,S\,L\,d\,\lvert R_c\rvert$

In contrast, Sync-SGD synchronizes the full parameter vector at every optimization step, multiplying the parameter communication term by S and yielding a cost of $S\,P$. The corresponding forward/backward exchange variants additionally transmit adjoints, adding a backward exchange of equal size and thereby doubling the exchange volume to $2\,S\,L\,d\,\lvert R_c\rvert$ in the layer-wise exchange variants.

Fully Local, Fully Local + Fwd-LE, and Fully Local + Fwd/Bwd-LE are omitted from the analysis because they aggregate no parameters across clients. Their communication volumes are 0, $S\,L\,d\,\lvert R_c\rvert$, and $2\,S\,L\,d\,\lvert R_c\rvert$, respectively.

8.3. Cost orderings

8.3.1. FedAvg + Fwd-LE versus OptimES

$$\frac{V_{\text{FedAvg+Fwd-LE}}}{V_{\text{OptimES}}} = \frac{P + S\,L\,d\,\lvert R_c\rvert}{P + L\,d\,\lvert E_c^{\mathrm{rem}}\rvert}.$$

Both methods share the parameter term P and exchange embeddings across all L layers, differing only in the number of cross-client edge activations: $S\,\lvert R_c\rvert$ for Fwd-LE against $\lvert E_c^{\mathrm{rem}}\rvert$ for OptimES, which activates each cross-client edge exactly once per epoch. The two extreme cases are as follows.

Lower extreme: one-hop aggregation with neighbor sampling. The S mini-batches of a local epoch partition the owned nodes V_c^{own} (each client uses batch size $B_c = N_c/S$), so every owned node is a seed in exactly one step. Under one-hop aggregation only seeds aggregate, and a cross-client edge is activated only in the step in which its owned endpoint is seeded. Therefore, each cross-client edge is activated at most once per epoch. With neighbor sampling, each seed aggregates over only a sampled fraction of its neighbors, and hence over only a fraction of its incident cross-client edges. The number of activations is thus strictly reduced, giving

$$S\,\lvert R_c\rvert \;\le\; \lvert E_c^{\mathrm{rem}}\rvert.$$

Forward exchange therefore communicates no more than OptimES, and strictly less whenever sampling drops any cross-client edge.

Worst case: unbounded hops without sampling. If the receptive field spans the entire subgraph at every step, every owned node aggregates in every step and each cross-client edge is activated in all S steps, so $\lvert R_c\rvert = \lvert E_c^{\mathrm{rem}}\rvert$ and $S\,\lvert R_c\rvert = S\,\lvert E_c^{\mathrm{rem}}\rvert$. Forward exchange then communicates S times the OptimES embedding volume.

8.3.2. Fwd/Bwd-LE versus Fwd-LE

For either parameter-aggregation regime, the difference

$$V_{\text{Fwd/Bwd-LE}} - V_{\text{Fwd-LE}} = S\,L\,d\,\lvert R_c\rvert$$

is the forward-exchange term again: the backward pass returns one adjoint per exchanged embedding, of the same width, at every layer and every step. The joint variant therefore communicates strictly

more than its forward-only counterpart, doubling the embedding term while leaving the parameter term unchanged.

8.3.3. FedAvg + Fwd-LE versus Sync-SGD

$$\frac{V_{\text{FedAvg+Fwd-LE}}}{V_{\text{Sync-SGD}}} = \frac{1}{S} + \frac{L\,d\,|R_c|}{P}.$$

The first term reflects that FedAvg + Fwd-LE synchronizes parameters once per epoch rather than once per step, and is therefore small whenever there are many steps per epoch. The second compares the per-step volume of exchanged embeddings against the full model size, and its magnitude depends on how large the exchange is relative to the model.

In our setting, the training graph has 8192 nodes. When the graph is split across 15 clients, client c owns approximately $N_c \approx 8192/15 \approx 546$ nodes that serve as the pool from which training seeds are drawn. With $S \approx 17$ steps per local epoch (shared across clients), client c uses batch size $B_c = N_c/S \approx 546/17 \approx 32$, so $1/S \approx 0.06$.

The second term, however, is not negligible here. The Multi-PNA backbone is small, with $P \approx 4.4 \times 10^5$ parameters (≈ 1.7 MiB), while six-hop neighbor sampling (fan-out $[10, 10, 10, 10, 5, 5]$) activates most of client c's cross-client edges at every step. Assuming each cross-client edge connects to a distinct remote node, the number of activated cross-client edges equals the number of remote embeddings exchanged, and the exchange-coverage logs record $|R_c| \approx 480$ per layer per step. With $L = 6$ and $d = 64$ the embedding term is

$$\frac{L\,d\,|R_c|}{P} \approx 0.42,$$

comparable to the entire parameter vector. The ratio is therefore

$$\frac{V_{\text{FedAvg+Fwd-LE}}}{V_{\text{Sync-SGD}}} \approx 0.06 + 0.42 \approx 0.48,$$

so forward exchange under per-epoch aggregation communicates roughly half as much as per-step parameter broadcast. This is the opposite of the large-model regime in which the embedding term would be negligible: here the backbone is small and the sampled receptive field is large, so the two channels are of similar size. For larger models the ratio would fall toward $1/S$.

The bidirectional variant (FedAvg + Fwd/Bwd-LE) doubles the embedding term, giving

$$\frac{V_{\text{FedAvg+Fwd/Bwd-LE}}}{V_{\text{Sync-SGD}}} = \frac{1}{S} + \frac{2\,L\,d\,|R_c|}{P} \approx 0.06 + 0.84 \approx 0.90,$$

so even the most expensive FedAvg-family method stays below Sync-SGD, but only marginally. Sync-SGD therefore provides per-step synchronization at a higher communication cost: roughly twice that of FedAvg + Fwd-LE, though only about 10% more than FedAvg + Fwd/Bwd-LE.

8.3.4. Summary

In our setting the communication costs satisfy

$$V_{\text{FedAvg}} < V_{\text{FedAvg+Fwd-LE}} < V_{\text{FedAvg+Fwd/Bwd-LE}},$$

and

$$V_{\text{Sync-SGD}} < V_{\text{Sync-SGD+Fwd-LE}} < V_{\text{Sync-SGD+Fwd/Bwd-LE}}.$$

OptimES is left out of these orderings because its position relative to FedAvg + Fwd-LE is regime-dependent. Since $V_{\text{OptimES}} = P + L\,d\,|E_c^{\text{rem}}|$ and $V_{\text{FedAvg+Fwd-LE}} = P + S\,L\,d\,|R_c|$, the two order by whether $S\,|R_c|$ exceeds $|E_c^{\text{rem}}|$:

$$V_{\text{OptimES}} < V_{\text{FedAvg+Fwd-LE}} \iff |E_c^{\text{rem}}| < S\,|R_c|.$$

This holds whenever the per-step exchange re-activates cross-client edges across steps, so that their epoch total $S\,|R_c|$ exceeds the full cross-client edge set $|E_c^{\text{rem}}|$ that OptimES sends once. In our six-hop setting $|R_c| \approx |E_c^{\text{rem}}|$, so $S\,|R_c| \approx S\,|E_c^{\text{rem}}|$ far exceeds $|E_c^{\text{rem}}|$, the condition is met, and Fwd-LE communicates about S times the OptimES embedding volume.

Moreover, provided the total bidirectional exchange volume stays below the additional parameter traffic that per-step synchronization incurs,

$$2\,S\,L\,d\,|R_c| < (S-1)\,P \qquad \iff \qquad \frac{1}{S} + \frac{2\,L\,d\,|R_c|}{P} < 1,$$

the two families join into a single total ordering through

$$V_{\text{FedAvg+Fwd/Bwd-LE}} < V_{\text{Sync-SGD}}.$$

This condition holds in our setting, where the ratio evaluates to ≈ 0.90. It can fail only when the per-step exchange is itself nearly as large as a full parameter broadcast.

The FedAvg family is consistently cheaper because parameter synchronization occurs once per epoch rather than once per step. Within either family, communication cost increases monotonically from no exchange, to forward-only exchange, to joint forward/backward exchange. The Sync-SGD family therefore orders analogously to the FedAvg family, since each exchange variant contributes only the corresponding embedding term on top of the shared $S\,P$ parameter cost. This ordering is used in the accuracy-communication trade-off discussion of Chapter 9.

9

Discussion

This chapter steps back from the per-experiment analysis of Chapter 7 and the communication-cost analysis of Chapter 8, and discusses the broader implications of the findings, the conditions under which they apply, and the practical and ethical considerations involved in deploying layer-wise exchange in real-world subgraph pattern detection systems.

Synthesis of empirical findings. The experiments validate the core theoretical claims of this thesis: that the forward exchange recovers centralized representations (Theorem 5.1) and the backward exchange recovers centralized gradients (Theorem 5.2), together eliminating the structural observability gap under the extended subgraph (A1) and shared-parameter (A2) assumptions. However, they also reveal two findings that the theorems do not anticipate.

The first finding is a pattern-dependent asymmetry in what closes the gap. For short cycles (C2, C3), the recovery comes almost entirely from forward exchange added on top of FedAvg, and tightening the parameter coupling from per-epoch to per-step synchronization contributes little beyond that. Non-cycle patterns (S-G, B-C) show the opposite pattern: forward exchange on top of FedAvg yields almost no improvement, and the recovery arrives only once parameters are synchronized at every step via Sync-SGD. The two pattern families are limited by different bottlenecks. Cycles are destroyed by a single cross-client cut but become locally observable as soon as the missing embeddings for the remote nodes are exchanged, so per-epoch parameter aggregation already suffices. The higher-arity non-cycle patterns remain partially visible within each client and are limited not by missing structure but by embedding quality, which only continuous per-step synchronization improves. This asymmetry indicates that subgraph FL methods cannot be evaluated on a single representative pattern: methods that recover one pattern family well may leave others untouched.

The second finding concerns the substitutability of communication channels, which differs between the forward and backward exchange and depends on partition density. Plain Sync-SGD, without any forward exchange, closely tracks Sync-SGD + Fwd-LE on both splits, even though no embeddings cross client boundaries. Once parameters are continuously synchronized, the shared model acquires more transferable information from the clients and better captures the global topology, so the explicit *forward* representation exchange becomes largely redundant under per-step aggregation. The *backward* exchange behaves differently, but only where the cross-client edges are dense. Isolating the contribution of backward exchange on top of per-step forward exchange yields a clear gain on the dense METIS partition, yet barely moves performance on the sparse Louvain partition. On the dense cut, the cross-client *gradient* signal that our framework exchanges between clients cannot be reconstructed from parameter aggregation alone. On the sparse cut, where few edges are severed, the improvements provided by the backward exchange are negligible. The design choice in subgraph FL is therefore not only how much to communicate but which channel to invest in and under what connectivity: forward representation exchange is largely substitutable by continuous parameter coupling, whereas backward adjoint exchange remains complementary to it precisely when cross-client coupling is dense.

Relation to distributed training: graph connectivity unifies data and tensor parallelism. Distributed deep learning offers two classical ways to split a training workload. *Data parallelism* (Dean et al., 2012; Goyal et al., 2017) replicates the full model on every worker, feeds each a disjoint shard of data, and synchronizes only gradients or parameters across workers. It assumes the samples are independent, so each replica's forward and backward pass is self-contained. *Tensor (model) parallelism* (Shoeybi et al., 2019) instead splits a single layer's computation across workers, which must then exchange activations in the forward pass and the corresponding gradients in the backward pass at every layer.

The two paradigms are usually orthogonal: independent data calls for data parallelism, while a model too large to replicate calls for model parallelism. Viewed through this lens, the federated baselines of this thesis, FedAvg and Sync-SGD, are pure data parallelism. They replicate the model and aggregate parameters or gradients, the latter being exactly textbook synchronous data parallelism. What breaks the data-parallel assumption is that our data is a connected graph rather than a set of independent samples: a node's receptive field spans several clients, so the per-node computation graph is itself distributed even though the model is fully replicated. Recovering the centralized computation therefore demands the communication pattern of tensor parallelism, which our layer-wise exchange framework supplies: the forward exchange transmits boundary activations and the backward exchange returns the corresponding adjoints at every layer, layered on top of a data-parallel parameter regime. Our framework is therefore a hybrid of the two distributed training paradigms: it never partitions the model, yet graph connectivity forces the activation and gradient exchange that, in dense-model training, only model partitioning would require.

Accuracy-communication trade-off. The accuracy gains of Chapter 7 come at a communication cost, which Chapter 8 quantifies. Weighing the accuracy of a method against its communication cost is what determines which method to deploy in practice. The cost analysis orders the methods by per-epoch communication volume, and the cost ordering runs opposite to the accuracy ordering. Per-epoch parameter aggregation (the FedAvg family) is consistently cheaper than per-step synchronization (the Sync-SGD family). In our setting, every Sync-SGD variant costs more than even FedAvg + Fwd/Bwd-LE, because it broadcasts the full parameter vector at every step. The only comparison without a fixed order is OptimES against FedAvg + Fwd-LE, whose relative cost depends on how often each cross-client edge is re-sampled across steps (Section 8.3.1). The strongest configuration evaluated in this thesis, Sync-SGD + Fwd/Bwd-LE, is also the most expensive, whereas a forward-only variant under per-epoch aggregation (FedAvg + Fwd-LE) recovers much of the gap at a fraction of the cost.

Whether to pay for the additional accuracy is application-dependent. In high-stakes detection problems such as AML, the gains documented in Section 7.2 can justify the cost. False negatives may allow illicit activity to remain undetected, false positives can trigger costly manual investigations, and undetected laundering can carry severe regulatory penalties. In such settings, the communication budget is justified by the cost of the errors it prevents.

Privacy considerations. Although layer-wise exchange avoids sharing raw features, labels, or local topology, the embeddings themselves may still encode information about local features, neighborhoods, and model states. A participant who follows the protocol but is curious about other clients' data could use them to infer whether a particular node was in the training set, what its features encode, or which edges connect it to its neighbors. The backward exchange widens this surface further. The adjoint vectors it transmits are gradient signals, which are themselves vulnerable to inversion attacks that reconstruct inputs from gradients. Several techniques could reduce these privacy risks at the cost of reduced accuracy: adding local or global differential privacy noise to the exchanged embeddings and adjoints (Abadi et al., 2016; Dwork, 2025); using secure aggregation (Bonawitz et al., 2017) or cryptographic protocols that restrict access to authorized participants. None of these techniques is tested in this thesis, and a full evaluation of the privacy-accuracy trade-off is left for future work.

Limitations of the framework and evaluation. The equivalence guarantees hold only under their stated assumptions: representation equivalence (Theorem 5.1) requires the extended subgraph assumption (A1), shared parameters (A2), and forward exchange (A3), while gradient equivalence (Theorem 5.2) additionally requires backward exchange (A4). The layer-wise exchange framework enforces

assumptions (A1), (A3), and (A4) by construction, and the shared-parameter assumption (A2) for the aggregated parameters at the granularity of the chosen aggregation regime (FedAvg vs. Sync-SGD), but requires full client participation ($q = 1.0$) at every step. This requirement is incompatible with realistic FL deployments where clients drop in and out.

Beyond client participation, the experimental realization itself departs from the idealized assumptions. The guarantees assume full-batch inference and exactly shared parameters and layer functions, while the implementation uses mini-batch neighbor sampling, client-local degree statistics, and unaggregated port embeddings (Section 7.1). Their combined effect is the residual gap to the centralized reference that persists even for the strongest configuration (Section 7.2).

In the scope of this thesis, the empirical evaluation is restricted to synthetic directed multigraphs of 8192 nodes, two partitioning strategies, and up to fifteen clients, with a fixed six-layer Multi-PNA model used throughout. Production federations often span more clients and more heterogeneous data than this benchmark captures. Scaling behavior beyond fifteen clients is not measured, and the interaction between pattern type and model depth is not explored. These restrictions do not undermine the structural claims but bound where they can be expected to transfer unchanged.

Further reductions in communication and privacy leakage. A natural extension to the presented layer-wise framework is to exchange embeddings *selectively* rather than universally. Instead of injecting every remote-node embedding at every layer, clients could restrict exchange to boundary regions likely to participate in cross-client patterns, guided by structural heuristics or learned attention weights. A complementary direction is *clustered* exchange, where clients are grouped by transaction volume, geography, business relationship, or observed cross-client connectivity, and frequent exchange is restricted to clients within the same group. Both strategies reduce communication cost and lower the risk of inference attacks at the same time.

Implications for real-world cross-client pattern detection. Anti-money laundering motivates this thesis: transaction graphs are distributed across institutions, jurisdictions, and regulatory regimes, and the patterns of interest cross those boundaries. The framework defined in this thesis restores the missing message-passing and gradient signals at partition boundaries, but a production AML deployment would face constraints beyond the algorithmic mechanism. Real financial graphs evolve continuously, with new transactions arriving in streams rather than fixed snapshots. Laundering typologies drift over time as criminals adapt to known detection patterns. Heterogeneity across institutions is also likely to be stronger than in the synthetic benchmark, since institutions can differ in customer populations, transaction volumes, jurisdictional risk standards, and reporting conventions. Extending the framework to dynamic and heterogeneous AML environments is therefore an important direction for future work.

Societal impact. Improving cross-client financial crime detection has clear societal value. Money laundering is structurally linked to a broader set of harms, including fraud, corruption, human trafficking, drug and weapons trafficking, and terrorist financing (Safdari et al., 2015). The patterns this thesis targets are precisely those that any single institution cannot see on its own. At the same time, the methods proposed here operate on highly sensitive data, and false accusations can cause real harm to individuals and organizations. The goal is therefore not maximizing detection accuracy in isolation but designing systems that balance effectiveness with privacy, fairness, auditability, and regulatory compliance. By framing the central problem as one of structural observability and providing a mechanism that restores it without centralizing the data, this thesis contributes a tool that improves detection while keeping privacy, fairness, and auditability as design goals rather than casualties.

10

Conclusion

This thesis investigated how much of the performance gap between federated and centralized training can be closed by synchronizing information across clients at every layer of the GNN. We formalized the gap as the *structural observability problem*. When a global graph is partitioned across clients, the substructures that define cross-client patterns become locally unobservable, and standard federated parameter aggregation alone cannot restore them. To address this, we proposed a per-step, layer-wise exchange framework with two complementary components: a *forward exchange* that synchronizes remote node representations during the forward pass, and a *backward exchange* that synchronizes the corresponding adjoint signals during the backward pass, neither of which exchanges raw features, labels, or local topology. We proved that, under the extended subgraph assumption (A1) and the shared-parameter assumption (A2), the forward exchange lets every client compute representations identical to those a centralized model would produce over the full graph (Theorem 5.1), and that the backward exchange makes the per-client parameter gradients sum to the exact centralized gradient (Theorem 5.2). Together they make federated training equivalent to centralized training in principle.

Experiments on synthetic directed multigraphs confirm that federated parameter aggregation alone is insufficient to recover the lost structural information, and that combining it with layer-wise exchange substantially closes the gap to the centralized training. FedAvg with forward exchange consistently outperforms plain FedAvg across both partitioning strategies and every federation scale considered. It also outperforms the stale, per-epoch OptimES (Naman and Simmhan, 2026), most clearly under denser cross-client connectivity, confirming that per-step freshness drives the recovery. The remaining gap then closes in two stages: tightening parameter aggregation from per-epoch to per-step reduces it, and adding the backward exchange reduces what remains, the latter yielding its largest gains where cross-client coupling is densest. The recovery is also asymmetric across pattern families: short cycles benefit most from neighborhood completion through forward exchange, whereas non-cycle patterns such as scatter-gather and biclique benefit most from per-step parameter equivalence, indicating that subgraph FL methods cannot be evaluated on a single representative pattern.

Taken together, these results show that the gap between federated and centralized training on partitioned graphs is a consequence of *what* clients can observe, *when* they synchronize, and *which* signal they exchange. Future work includes relaxing the full-participation requirement of the per-step protocol so that the framework remains applicable when clients drop in and out, and introducing stronger privacy protections for the exchanged embeddings and adjoints to quantify and bound the residual information leakage. Because the recovery is driven by per-step exchange, which carries a communication cost that grows with cross-client connectivity, a complementary future direction is reducing communication overhead through selective or clustered exchange, restricting the protocol to boundary regions or to clients with dense mutual connectivity. Finally, extending the framework to dynamic and heterogeneous real-world graph learning systems, such as production AML deployments, would test the structural observability argument under the conditions this thesis was motivated by.

References

Abadi, Martin et al. (2016). “Deep learning with differential privacy”. In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 308–318.

Aliakbari, Javad, Johan Östman, and Alexandre Graell i Amat (2024). “Decoupled Subgraph Federated Learning”. In: *International Conference on Learning Representations*.

Altman, Erik et al. (2023). “Realistic synthetic financial transactions for anti-money laundering models”. In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*. NIPS ’23. New Orleans, LA, USA: Curran Associates Inc.

Blondel, Vincent D. et al. (2008). “Fast unfolding of communities in large networks”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10, P10008. doi: 10.1088/1742-5468/2008/10/P10008.

Bonawitz, Keith et al. (2017). “Practical secure aggregation for privacy-preserving machine learning”. In: *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191.

Bottou, Léon (2010). “Large-Scale Machine Learning with Stochastic Gradient Descent”. In: *Proceedings of COMPSTAT’2010*. Ed. by Yves Lechevallier and Gilbert Saporta. Heidelberg: Physica-Verlag HD, pp. 177–186. doi: 10.1007/978-3-7908-2604-3_16.

Chen, Zhengdao et al. (2020). “Can graph neural networks count substructures?” In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS ’20. Vancouver, BC, Canada: Curran Associates Inc. isbn: 9781713829546.

Chen, Zhiyuan et al. (Feb. 2018). “Machine Learning Techniques for Anti-Money Laundering (AML) Solutions in Suspicious Transaction Detection: A Review”. In: *Knowledge and Information Systems*. doi: 10.1007/s10115-017-1144-z.

Corso, Gabriele et al. (2020). “Principal neighbourhood aggregation for graph nets”. In: NeurIPS 2020.

Dean, Jeffrey et al. (2012). “Large Scale Distributed Deep Networks”. In: *Neural Information Processing Systems*.

Dwork, Cynthia (2025). “Differential privacy”. In: *Encyclopedia of Cryptography, Security and Privacy*. Springer, pp. 649–652.

Egressy, Béni et al. (2024). “Provably powerful graph neural networks for directed multigraphs”. In: AAAI’24/IAAI’24/EAAI’24. doi: 10.1609/aaai.v38i10.29069.

Eliasof, Moshe et al. (23–29 Jul 2023). “Graph Positional Encoding via Random Feature Propagation”. In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, pp. 9202–9223.

Fan, Jiani et al. (2026). “Deep Learning Approaches for Anti-Money Laundering on Mobile Transactions: Review, Framework, and Directions”. In: *IEEE Internet of Things Journal* 13.6, pp. 10407–10431. doi: 10.1109/JIOT.2026.3653434.

Gilmer, Justin et al. (2017). “Neural Message Passing for Quantum Chemistry”. In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. Proceedings of Machine Learning Research. JMLR.org, pp. 1263–1272.

Goyal, Priya et al. (2017). “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour”. In: *ArXiv* abs/1706.02677.

Hamilton, William L, Rex Ying, and Jure Leskovec (2017). “Inductive Representation Learning on Large Graphs”. In: *Advances in Neural Information Processing Systems*. Vol. 30, pp. 1024–1034.

Huang, Yinan et al. (2023). “Boosting the Cycle Counting Power of Graph Neural Networks with I^2-GNNs”. In: *The Eleventh International Conference on Learning Representations*.

Jaume, Guillaume et al. (2019). *edGNN: a Simple and Powerful GNN for Directed Labeled Graphs*. arXiv: 1904.08745 [cs.LG].

Karypis, George and Vipin Kumar (1998). “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs”. In: *SIAM Journal on Scientific Computing* 20.1, pp. 359–392. doi: 10.1137/S1064827595287997.

Kingma, Diederik P. and Jimmy Ba (2015). “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations (ICLR)*. San Diego, CA, USA. arXiv: 1412.6980 [cs.LG].

Kipf, Thomas N. and Max Welling (2017). “Semi-Supervised Classification with Graph Convolutional Networks”. In: *International Conference on Learning Representations (ICLR)*.

Kurshan, Eren, Hongda Shen, and Haojie Yu (2020). “Financial Crime & Fraud Detection Using Graph Computing: Application Considerations & Outlook”. In: *2020 Second International Conference on Transdisciplinary AI (TransAI)*, pp. 125–130. doi: 10.1109/TransAI49837.2020.00029.

Lei, Runze et al. (2023). “Federated learning over coupled graphs”. In: *IEEE Transactions on Parallel and Distributed Systems* 34.4, pp. 1159–1172.

Li, Anran et al. (2024). “Historical Embedding-Guided Efficient Large-Scale Federated Graph Learning”. In: *Proceedings of the ACM on Management of Data (SIGMOD)*.

Li, Xunkai et al. (Jan. 2025). “OpenFGL: A Comprehensive Benchmark for Federated Graph Learning”. In: *Proc. VLDB Endow.* 18.5, pp. 1305–1320. issn: 2150-8097. doi: 10.14778/3718057.3718061.

Liu, Zhi et al. (2023). “Subgraph Federated Learning with Global Graph Reconstruction”. In: *Proceedings of the 7th International Joint Conference on Web and Big Data (APWeb-WAIM)*. Berlin, Heidelberg: Springer, pp. 158–173. doi: 10.1007/978-981-97-2303-4_11.

McMahan, H. B. et al. (2016). “Communication-Efficient Learning of Deep Networks from Decentralized Data”. In: *International Conference on Artificial Intelligence and Statistics*.

Naman, Pranjal and Yogesh Simmhan (2026). “OptimES: Optimizing federated learning using remote embeddings for graph neural networks”. In: *Journal of Parallel and Distributed Computing* 211, p. 105227. issn: 0743-7315. doi: https://doi.org/10.1016/j.jpdc.2026.105227.

Nicholls, J., A. Kuppa, and N.-A. Le-Khac (2021). “Financial Cybercrime: A Comprehensive Survey of Deep Learning Approaches to Tackle the Evolving Financial Crime Landscape”. In: *IEEE Access* 9, pp. 163965–163986.

Robbins, Herbert and Sutton Monro (1951). “A Stochastic Approximation Method”. In: *The Annals of Mathematical Statistics* 22.3, pp. 400–407. doi: 10.1214/aoms/1177729586.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). “Learning representations by back-propagating errors”. In: *Nature* 323, pp. 533–536.

Safdari, Akbar et al. (2015). “Social impact of money laundering”. In: *Asian Journal of Research in Social Sciences and Humanities* 5.1, pp. 1–16.

Sato, Ryoma, Makoto Yamada, and Hisashi Kashima (2019). “Approximation ratios of graph neural networks for combinatorial problems”. In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc.

Scarselli, Franco et al. (2009). “The Graph Neural Network Model”. In: *IEEE Transactions on Neural Networks* 20, pp. 61–80.

Shoeybi, Mohammad et al. (2019). “Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism”. In: *ArXiv* abs/1909.08053.

Tahmasebi, Behrooz, Derek Lim, and Stefanie Jegelka (25–27 Apr 2023). “The Power of Recursion in Graph Neural Networks for Counting Substructures”. In: *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*. Ed. by Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent. Vol. 206. Proceedings of Machine Learning Research. PMLR, pp. 11023–11042.

Vignac, Clément, Andreas Loukas, and Pascal Frossard (2020). “Building powerful and equivariant graph neural networks with message-passing”. In: *CoRR* abs/2006.15107. arXiv: 2006.15107.

Weber, Mark et al. (2018). “Scalable Graph Learning for Anti-Money Laundering: A First Look”. In: *ArXiv* abs/1812.00076.

Wu, Chuhan et al. (2022). “A federated graph neural network framework for privacy-preserving personalization”. In: *Nature Communications* 13.1, p. 3091.

Wu, Lingfei et al. (2022). “Graph neural networks: foundation, frontiers and applications”. In: *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 4840–4841.

Xu, Keyulu et al. (2019). “How Powerful are Graph Neural Networks?” In: *International Conference on Learning Representations (ICLR)*.

Yao, Yuan et al. (2023). “FedGCN: Convergence-Communication Tradeoffs in Federated Training of Graph Convolutional Networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)*.

You, Jiaxuan et al. (May 2021). “Identity-aware Graph Neural Networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35, pp. 10737–10745. doi: 10.1609/aaai.v35i12.17283.

Zhang, Ke, Lichao Sun, et al. (2024). “Deep Efficient Private Neighbor Generation for Subgraph Federated Learning”. In: *arXiv preprint arXiv:2401.04336*.

Zhang, Ke, Carl Yang, et al. (2021). “Subgraph Federated Learning with Missing Neighbor Generation”. In: *Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021)*. NeurIPS ’21. Red Hook, NY, USA: Curran Associates, Inc.

A

Enabling Cross-Client Edges in Client Subgraphs

Table A.1 illustrates the impact of extending the client subgraphs with cross-client edges under the two federated splits, Louvain and METIS, for client counts $k \in \{3, 5, 10, 15\}$. Results are reported for the 6-layer Multi-PNA variant under fully local training.

Table A.1: PR-AUC (%) for $k \in \{3, 5, 10, 15\}$ clients on Louvain and METIS partitions. Results are reported for the 6-layer Multi-PNA model under two settings: (a) client subgraphs *without* cross-client edges and (b) client subgraphs augmented *with* cross-client edges. The underlying training regime is fully local training.

(a) Fully local training **without** cross-client edges

Partition Size (k)	C2	C3	C4	C5	C6	S-G	B-C	Macro	Δ
Louvain Splits									
$k = 3$	92.84±7.57	91.99±4.96	93.99±2.12	79.69±0.49	81.37±0.57	89.60±3.72	82.51±3.41	87.43	–
$k = 5$	55.00±7.79	70.23±3.77	85.01±1.31	79.68±0.18	80.22±0.49	71.64±0.64	73.27±0.58	73.58	–
$k = 10$	43.83±0.51	63.04±0.41	82.69±0.12	78.27±0.04	77.57±0.46	69.86±0.25	70.85±0.89	69.44	–
$k = 15$	41.60±0.36	63.20±0.10	81.54±0.20	77.19±0.39	75.21±0.62	68.83±0.15	69.50±0.59	68.15	–
METIS Splits									
$k = 3$	40.57±0.73	58.93±0.19	70.64±0.36	60.21±0.30	53.16±0.43	63.92±0.17	63.89±0.35	58.76	–
$k = 5$	38.50±0.76	56.65±0.23	67.32±0.24	54.65±0.38	47.50±0.26	60.81±0.57	62.00±0.22	55.35	–
$k = 10$	38.30±0.62	54.85±0.28	64.09±0.16	50.03±0.47	40.70±0.93	58.46±0.37	59.91±0.21	52.33	–
$k = 15$	37.11±0.50	53.30±0.34	63.75±0.42	49.18±0.59	39.27±0.69	57.50±0.44	58.90±0.35	51.29	–

(b) Fully local training **with** cross-client edges

Partition Size (k)	C2	C3	C4	C5	C6	S-G	B-C	Macro	Δ
Louvain Splits									
$k = 3$	98.94±0.18	97.86±0.08	98.00±0.27	84.01±1.41	86.63±0.27	96.51±0.17	94.02±3.18	93.71	**+6.28**
$k = 5$	86.95±0.91	88.37±0.32	92.40±0.23	83.55±0.25	85.30±0.67	81.81±0.86	76.72±2.91	85.01	**+11.43**
$k = 10$	48.90±3.19	65.30±0.95	83.90±0.33	82.48±0.33	84.01±0.51	71.46±0.41	71.89±0.48	72.56	**+3.12**
$k = 15$	43.50±0.17	64.15±0.25	83.57±0.15	80.94±0.18	81.17±0.16	69.74±0.39	71.21±0.20	70.61	**+2.46**
METIS Splits									
$k = 3$	98.47±0.44	95.74±0.13	94.08±0.19	81.71±1.19	84.08±0.20	90.32±0.76	81.55±5.85	89.42	**+30.66**
$k = 5$	90.17±8.78	87.91±5.55	87.56±1.19	80.47±0.14	82.40±0.14	73.22±0.29	71.86±1.38	81.94	**+26.59**
$k = 10$	63.97±5.71	70.58±2.05	81.82±0.67	79.38±0.41	79.81±0.28	69.43±0.57	70.77±0.30	73.68	**+21.35**
$k = 15$	48.96±3.95	63.76±0.45	80.00±0.37	78.06±0.27	76.73±0.71	68.68±0.51	69.72±0.42	69.42	**+18.13**

As shown in the table, including cross-client edges in message passing during local training improves performance across all client sizes and for both partitioning schemes. The final column reports the absolute macro PR-AUC improvement obtained by retaining cross-client edges relative to the corresponding configuration without cross-client edges. The effect is most pronounced under the METIS partition. At $k = 3$, macro PR-AUC rises from 58.76% to 89.42% ($+30.66$pp improvement), driven largely by C2, which climbs from 40.57% to 98.47% ($+57.90$pp). Under the Louvain partition, the same setting improves far more modestly, from 87.43% to 93.71% ($+6.28$pp). The contrast follows from where

each partitioning strategy places its partition boundaries: Louvain cuts along low-density seams, so few cross-client edges exist to be restored and little boundary connectivity is lost when they are removed. METIS produces a much denser edge-cut, so a large fraction of each client's local structure depends on cross-client edges, and excluding them removes correspondingly more of the connectivity needed for accurate pattern detection.

A consistent trend is observed across both settings, *with* and *without* cross-client edges: macro PR-AUC decreases as the number of clients grows. As the graph is partitioned across more clients, each client owns a smaller fraction of the global structure and a larger proportion of relevant neighborhoods crosses partition boundaries, making the underlying patterns progressively harder to observe.

Furthermore, the impact of cross-client edges is not uniform across pattern types. The largest gains occur for the low-order cycles, particularly C2 and C3, whose detection depends on short closed walks that are easily broken when edges crossing partition boundaries are removed. When cross-client edges are retained, both patterns recover to near-perfect performance at small client counts, with C2 reaching 98.47% on METIS and 98.94% on Louvain at $k = 3$. In contrast, removing cross-client edges causes performance to deteriorate sharply under the denser METIS partitioning, where C2 falls to 40.57% and C3 to 58.93%. The higher-order cycles (C5 and C6) exhibit a smaller relative dependence on cross-client edges than the short-cycles. Scatter-gather and biclique patterns also benefit from restored cross-client connectivity, although their gains are less dramatic than those observed for C2 and C3.

Without these edges, even immediate-neighborhood structures such as the 2-hop Cycle-2 pattern become incomplete, leading to a substantial degradation in detection performance. Retaining cross-client edges restores the local connectivity required to observe these patterns correctly, allowing the model to recover the true neighborhood structure on which accurate pattern detection depends.

Together, these results provide empirical support for assumption (A1), demonstrating that extending client subgraphs with cross-client edges substantially improves the observability of immediate neighborhood signals in the federated setting and is a key prerequisite for preserving local structural information across partition boundaries.

B

Centralized PNA Model Performance

This appendix reports the performance of the centralized PNA model under both minority-class F1 and minority-class PR-AUC. F1 scores are included throughout this appendix to enable comparison with Egressy et al. (2024), on whose benchmark these experiments are based. Minority-class PR-AUC counterparts are provided for the hyperparameter tuning experiments to confirm that the metric rankings are consistent.

The first section reports the incremental performance gains from the architectural adaptations of Egressy et al. (2024). The second section tunes the resulting model's depth, fanout, and class-weighting strategy on both F1 and PR-AUC.

All experiments train on the synthetic directed multigraph described in Chapter 3, without any client partitioning.

B.1. Model Variants and Adaptations

The incremental performance improvements obtained by augmenting the PNA model with each adaptation are summarized in Table B.1. All models were trained using a shared set of hyperparameters, namely: a hidden dimension of 64, two GNN layers, two message-passing hops (i.e., one hop of graph propagation per layer), a dropout rate of 0.1, a learning rate of 0.001, and a weight decay of 0.0001.

Additional hyperparameters were introduced for the extended variants of the model. For mini-batch training with neighborhood sampling, a batch size of 32 and a fixed fanout of $[10, 4]$ neighbors were used. When incorporating ego IDs, the ego-embedding dimension was set equal to the batch size. For experiments that included port-number information, the port-embedding dimension was set to 8.

The results in Table B.1 show that both port IDs and ego IDs improve the performance of the PNA model under the mini-batch training setting, consistent with the findings of (Egressy et al., 2024). In particular, adding ego IDs yields a significant improvement on the cycle detection tasks, most notably on $C2$, increasing the macro minority-class F1 from 76.47% to 84.04%.

In contrast, incorporating ego IDs under full-batch training reduces performance relative to the corresponding model without ego IDs. This difference can be explained by the role ego IDs play in mini-batch neighborhood sampling. In the mini-batch setting, ego IDs act as relative positional encodings with respect to the sampled seed node, allowing the model to distinguish the center node from its surrounding neighborhood. Under full-batch training, however, all nodes are processed simultaneously, meaning that every node effectively becomes a seed node and receives its own unique ego embedding. Consequently, ego IDs no longer encode relative positional information and instead behave as node-specific identity features. This shifts the model toward memorizing graph-specific node identities rather than learning transferable structural patterns, thereby reducing its ability to generalize across graphs.

Table B.1: Minority-class F1 scores (%) for the synthetic subgraph detection tasks, averaged over two runs with different seeds. The table is divided into two settings: the upper block shows the standard PNA model and its incremental adaptations under full-batch training; the lower block reports the same incremental adaptations under mini-batch training with neighborhood sampling. The rightmost column reports the macro-averaged minority-class F1. Hyperparameters follow the description above the table.

Centralized Synthetic Subgraph Detection Minority-Class F1 Results												
	deg-in	**deg-out**	**fan-in**	**fan-out**	**C2**	**C3**	**C4**	**C5**	**C6**	**S-G**	**B-C**	**Macro F1**
Full-batch Setting												
PNA (Corso et al., 2020)	99.10	0.00	94.63	0.00	0.32	37.66	63.13	51.01	49.90	64.96	61.19	47.45
+ Reverse MP (Egressy et al., 2024)	99.61	99.88	95.35	96.02	15.02	49.87	75.34	71.66	76.69	62.62	62.07	73.10
+ Port IDs (Sato, Yamada, and Kashima, 2019)	95.61	96.00	99.40	98.53	6.81	52.78	75.44	72.85	75.42	62.16	64.07	72.64
+ Ego IDs (You et al., 2021)	93.64	94.47	96.6	98.94	0.62	42.05	72.26	65.98	62.04	53.93	62.87	67.58
Mini-batch Setting												
PNA + Reverse MP	99.98	99.86	96.20	95.40	25.07	57.59	74.69	71.00	75.44	66.63	65.71	73.23
+ Port IDs	99.99	99.99	100.00	99.98	27.69	56.98	74.30	71.50	75.69	66.46	68.62	76.47
+ Ego IDs	100.00	99.97	99.99	99.99	97.62	63.94	75.68	73.67	73.30	68.99	71.34	84.04

B.2. Hyperparameter Tuning

B.2.1. Minority-Class F1 Results

Table B.2 summarizes the minority-class F1 scores obtained during hyperparameter tuning. Each configuration varies one of the following components: (1) model depth, adjusted by changing the number of layers; (2) neighborhood size, controlled by the number of neighbors sampled per hop; or (3) the class-weighting strategy.

The results indicate that increasing the model depth to six layers enables the PNA model to capture larger structural patterns, boosting performance especially on the detection of Cycle-3, Cycle-4, scatter-gather, and biclique patterns. Moreover, enabling class weighting tends to improve minority-class F1 scores by giving additional emphasis to positive samples during training.

The two best-performing hyperparameter configurations use six layers, and hence six message-passing hops, with neighborhood sizes set to $[10, 10, 10, 10, 5, 5]$ and $[15, 15, 10, 10, 5, 5]$ with class weighting enabled. In these settings, the model achieves a macro F1 score of 95.52% and 95.16% respectively. Most sub-tasks exceed 97.00% F1, with the exception of the Cycle-5 and Cycle-6 detection tasks.

These two six-layer configurations are carried forward for the cost-vs-performance comparison in the minority-class PR-AUC analysis below.

Table B.2: Minority-class F1 scores (%) for the **parameter-tuning experiments**. In all configurations, model and optimizer hyperparameters follow Table 6.1, with the model depth, fanout, and class-weighting strategy varied as indicated in the leftmost column. All values are averaged over two seeds.

Parameter Tuning Minority-Class F1 Results												
	deg-in	**deg-out**	**fan-in**	**fan-out**	**C2**	**C3**	**C4**	**C5**	**C6**	**S-G**	**B-C**	**Macro F1**
L=2, neigh=[10,4], w=None	100.00	99.98	99.99	99.99	74.60	57.84	74.58	72.61	75.82	67.94	67.77	81.01
L=2, neigh=[10,4], w=auto	100.00	99.97	99.99	99.99	97.62	63.94	75.68	73.67	73.30	68.99	71.34	84.04
L=2, neigh=[15,10], w=None	99.98	99.99	100.00	100.00	96.26	57.04	74.78	72.79	76.46	66.82	69.06	83.02
L=2, neigh=[15,10], w=auto	100.00	99.97	99.98	99.99	96.84	64.42	75.83	72.98	72.97	69.41	70.95	83.94
L=3, neigh=[10,10,5], w=None	99.99	100.00	99.99	100.00	98.01	97.37	76.93	73.86	78.44	66.31	65.48	86.94
L=3, neigh=[10,10,5], w=auto	99.95	100.00	99.99	100.00	99.32	98.54	76.71	74.95	75.45	69.94	71.04	87.81
L=3, neigh=[15,10,5], w=None	99.98	99.87	99.99	100.00	95.59	95.18	76.74	73.96	79.12	68.09	69.14	87.06
L=3, neigh=[15,10,5], w=auto	99.94	100.00	99.99	99.99	99.24	98.33	76.71	74.23	72.67	69.95	70.37	87.40
L=6, neigh=[10,10,10,10,5,5], w=auto	99.98	99.99	99.97	100.00	99.21	98.94	98.38	87.12	72.83	97.13	97.19	95.52
L=6, neigh=[15,15,10,10,5,5], w=auto	99.92	99.98	99.99	100.00	99.42	98.92	98.39	81.07	74.39	97.45	97.19	95.16

Notes. **L**: number of layers used; **neigh**: number of neighbors sampled per hop; **w**: class-weighting strategy. When set to *auto*, the loss function uses a per-task positive class weight computed from the training labels, defined as $\text{pos\_weight}_t = \frac{\#\text{negatives}_t}{\#\text{positives}_t}$ for each task t.

B.2.2. Minority-Class PR-AUC Results

Table B.3 reports minority-class PR-AUC scores for the same hyperparameter configurations. The model rankings are consistent with the F1 results: performance increases with model depth, confirming that deeper models better capture the multi-hop structural patterns present in the dataset.

For configurations with the same number of layers, increasing the neighborhood sample size provides only marginal gains. For the six-layer model, the larger fanout $[15, 15, 10, 10, 5, 5]$ improves macro PR-AUC by only 0.12 percentage points over $[10, 10, 10, 10, 5, 5]$ (98.27% vs. 98.15%). For the three-layer model, the $[15, 10, 5]$ and $[10, 10, 5]$ configurations with auto weighting achieve identical macro PR-AUC of 92.56%.

Since neighborhood-sampled GNN training scales with a factor of r^L, increasing the per-layer fanout r_l raises both computational and memory cost (see Appendix C for a detailed complexity analysis) without a meaningful improvement in performance. The smaller fanout configurations are therefore preferred.

The 6-layer model with $[10, 10, 10, 10, 5, 5]$ fanout serves as the centralized benchmark referenced in Chapter 7.

Table B.3: Minority-class PR-AUC scores (%) for the **parameter-tuning experiments**. In all configurations, model and optimizer hyperparameters follow Table 6.1, with the model depth, fanout, and class-weighting strategy varied as indicated in the leftmost column. All values are averaged over two seeds.

Parameter Tuning Minority-Class PR-AUC Results												
	deg-in	**deg-out**	**fan-in**	**fan-out**	**C2**	**C3**	**C4**	**C5**	**C6**	**S-G**	**B-C**	**Macro PR-AUC**
L=2, neigh=[10,4], w=None	100.00	100.00	100.00	100.00	98.07	66.78	86.15	85.11	87.89	75.68	77.46	88.83
L=2, neigh=[10,4], w=auto	100.00	100.00	100.00	100.00	97.96	66.28	86.23	85.18	88.20	75.39	76.86	88.74
L=2, neigh=[15,10], w=None	100.00	100.00	100.00	100.00	96.12	66.54	85.73	85.16	87.82	75.57	77.29	88.57
L=2, neigh=[15,10], w=auto	100.00	100.00	100.00	100.00	97.77	67.21	86.39	85.43	88.09	76.35	77.52	88.98
L=3, neigh=[10,10,5], w=None	100.00	100.00	100.00	100.00	97.33	97.19	87.16	86.85	89.83	77.30	78.31	92.18
L=3, neigh=[10,10,5], w=auto	100.00	100.00	100.00	100.00	99.31	98.88	87.31	86.76	89.91	77.62	78.41	92.56
L=3, neigh=[15,10,5], w=None	100.00	99.99	100.00	100.00	98.28	97.86	87.13	86.95	89.88	77.62	78.37	92.37
L=3, neigh=[15,10,5], w=auto	100.00	100.00	100.00	100.00	99.17	98.84	87.32	86.80	89.88	77.53	78.65	92.56
L=6, neigh=[10,10,10,10,5,5], w=auto	100.00	100.00	100.00	100.00	99.63	99.60	99.68	93.51	88.57	99.34	99.30	98.15
L=6, neigh=[15,15,10,10,5,5], w=auto	100.00	100.00	100.00	100.00	99.85	99.56	99.80	94.19	88.94	99.46	99.18	98.27

Notes. **L**: number of layers used; **neigh**: number of neighbors sampled per hop; **w**: class-weighting strategy. When set to *auto*, the loss function uses a per-task positive class weight computed from the training labels, defined as $\text{pos\_weight}_t = \frac{\#\text{negatives}_t}{\#\text{positives}_t}$ for each task t.

C

Space and Time Complexity of Neighborhood Sampling

Assuming that the neighborhood size r_l is the same across all layers, i.e., $r_l = r$ for any layer, the per-batch time complexity of neighborhood sampling in GNN training can be expressed as (Hamilton, Ying, and Leskovec, 2017)

$$O(br^L F^2),$$

where b is the batch size, r is the fanout (the number of sampled neighbors per layer), L is the number of GNN layers (i.e., the number of hops), and F is the feature dimension. Over an epoch, each node is visited once across many batches. Therefore, the time complexity per epoch becomes

$$\text{epoch cost} = \frac{N}{b} \cdot O(br^L F^2) = O(Nr^L F^2),$$

where N is the total number of nodes.

Since neighborhood sampling collects approximately $O(br^L)$ nodes per batch and each node stores an F-dimensional feature vector, the memory required to store all sampled nodes across L hops can be described as

$$O(br^L F).$$

Hence, the memory complexity during neighborhood-sampled GNN training is

$$O(br^L F + LF^2),$$

where the additional LF^2 term accounts for storing the model parameters across L layers.

It is important to note that r^L appears as a multiplicative factor in both the space and time complexities of neighborhood-sampled GNN training. As a result, the complexities grow exponentially with the number of GNN layers L, and increasing the neighborhood size r has a direct impact on increasing both memory and computational cost.

This observation directly motivates the choice of smaller early-layer fanouts in the hyperparameter configuration selected in the Results section (see Section B.2), where the $[10, 10, 10, 10, 5, 5]$ fanout was preferred over larger alternatives due to its lower computational and memory cost without sacrificing predictive performance.