Boundary-Consistent Graph Neural Networks for Topological Flux Prediction

Anonymous authors
Paper under double-blind review

Abstract

Graph Neural Networks (GNNs) have achieved notable success in spatiotemporal modeling across diverse application domains. However, their efficacy in flux prediction (FP), where the goal is to model spatiotemporal fluid transport over networked physical systems, remains contentious. Recent studies report that GNNs can underperform even simple baselines in FP settings, leading to a claim that GNNs may be intrinsically ill-suited for such tasks.

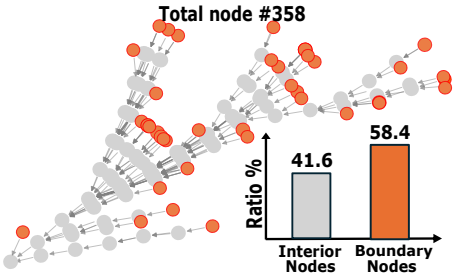
In this paper, we revisit this claim by dissecting the GNN learning dynamics on fluid transport networks, with an emphasis on its boundary regions. Specifically, we decompose the graph into boundary and interior nodes, where boundary nodes regulate the total influx and are the primary interface with external forcing. Our empirical and theoretical analyses reveal that dominant prediction errors concentrate at boundary nodes. From a dynamical-systems perspective, we interpret the boundary errors as the consequence of unmodeled external forcing, which causes degraded performance on boundaries. We therefore hypothesize that the observed performance degradation of GNNs was not caused by their expressivity; rather, it arises from the deficit of explicit external forcing modeling during training.

To validate this hypothesis, we propose `gTFP`, which learns ghost-node proxies to approximate unmodeled external forcing. Each boundary node is augmented with an associated ghost node that represents the latent forcing. This yields a ghost–boundary–interior coupled system, which we solve using an implicit fixed-point formulation. The resulting equilibrium *jointly* infers the external forcing and propagates it into the interior. This enriches standard GNN backbones with boundary-consistent representations while preserving interior message passing. Extensive experiments on two real-world fluid network datasets demonstrate that `gTFP` improves standard GNNs by reducing average MSE by 8.4% and 5.0%, and boundary-node MSE by 11.2% and 7.1%, respectively. For computational efficiency, we further introduce an explicit inverse-operator solver that amortizes the fixed-point inference and accelerates inference by up to 2×, depending on the backbone architecture.

1 Introduction

Flux prediction enjoys broad use in fluid systems (Kratzert et al., 2021; Jin et al., 2023), *e.g.,* flood forecasting (Jiang et al., 2025; Bentivoglio et al., 2025), hydrochemical modeling (Mangold & Tsang, 1991), estuarine circulation (Geyer & MacCready, 2014), among others. Since such systems unfold in both space and time, Graph Neural Network (GNN) (Zhou et al., 2020; Wu et al., 2020) emerges as a seemingly plausible modeling choice, given its demonstrated success in related tasks such as traffic forecasting (Jin et al., 2023) and energy transmission (Varbella et al., 2024).

Yet, recent studies reveal that GNNs often ignore the underlying fluid dynamics and learn absurd patterns from data, *e.g.,* predicting fluxes moving from downstream to upstream, which violate gravity and conservation laws (Kirschstein & Sun, 2024). As a result, some conclude that incorporating fluid system topology offers little benefit, as GNNs underperform even simple baselines like multilayer perceptrons (MLPs), which do not model graph structure at all (Kirschstein & Sun, 2024).



Avg. MSE per node

Model	Boundary	Interior	Diff. (%)
ResGCN	0.1408	0.0985	30.04%
ResGAT	0.1401	0.0836	40.33%
GCNII	0.1511	0.0891	41.03%
GNN error (avg.)	0.1217		–
MLP	0.1210	0.1030	14.89%
MLP error (avg.)	0.1135		–

Figure 1: A fluid network from the LamaH-CE2 dataset, where boundary nodes account for more than half of the graph topology. Although standard GNNs outperform a simple baseline (*e.g.,* MLP) on interior nodes, a misleading conclusion may arise due to their larger loss on boundary nodes, resulting in a higher overall average MSE. We hypothesize that this is because GNNs lacking explicit modeling of boundary condition, which is essential in classic PDE-based flux prediction models for networked fluid systems.

In this paper, we argue that such conclusions are premature. We revisit the prediction loss patterns from GNN-based flux models and find that the dominant errors lie at the *boundary nodes* that interface with unobserved external forcing. As Figure 1 illustrates, whereas boundary nodes account for more than half of the total nodes in a fluid network, existing models consistently yield much higher prediction errors on them compared to interior nodes. In fact, if we restrict evaluation to interior nodes alone, GNN predictors significantly outperform baseline MLPs. This discrepancy suggests that treating all nodes equally without distinguishing boundary from interior may underlie the perceived failure of GNNs in fluid systems.

To scrutinize this observation further, we analyze the GNN learning behavior from a dynamical-system perspective (Poli et al., 2019), where message-passing simulates the update of node states under local interaction rules (*e.g.,* a differential equation), while each such layer proceeds a discrete time step (*e.g.,* an Euler step). Boundary conditions in such systems constrain their solution space (LeVeque, 2007)(in our setting, this corresponds to exogenous inflow forcing at upstream sources). In GNN-based flux models, boundary nodes are discrete counterparts of these conditions, as they regulate influx into the entire network through external forcing. We hypothesize that such regulation power of missing external forcing acts as strong inductive bias, and incorporating it into the learning process is critical to unleashing the full potential of GNNs in flux prediction tasks. Our goal is not to enforce global PDE constraints, but to infer missing external forcing and reduce prediction errors at boundary nodes.

To instantiate and validate our hypothesis, we propose a novel computing framework, termed ***gTFP***, where we borrow the ghost node technique (Tseng & Ferziger, 2003) from finite-difference methods (FDM) to infer latent external forcing. Specifically, we approximate ghost nodes by extrapolating from boundary nodes and their immediate (downstream) neighbors. This imposes recursive coupling on message-passing, where each boundary node depends on ghost nodes, whose embeddings are in turn defined by interior and boundary nodes themselves. Unknown node representations thus appear on both sides of the update equations, making standard layer-wise GNN training inapplicable. To solve this, we adopt implicit GNN (Gu et al., 2020), which recasts message-passing as a fixed-point problem and seeks an equilibrium that aligns node representations with the structural relationships determined by boundary conditions.

Note, imposing different boundary conditions on fluid system may derive disparate structural couplings among ghost, boundary, and interior nodes. Each such coupling defines a unique augmentation of graph adjacency, which entails extensive craftsmanship to adapt the implicit GNN solver to every possible augmentation. To counter this, we further unify our framework by treating the inverse of the augmented adjacency as a learnable operator, lending a closed-form approximation to the implicit solution.

Specific contributions in this paper are summarized as follows.

(1) We decouple error sources in GNN-based flux models and find the dominant loss stems from missing explicit boundary-node modeling. The analysis is presented in Section 3.
(2) We propose `gTFP` to learn external forcing via ghost-node proxies; it outperforms GNN baselines by 6.68% on average and mitigates the boundary–interior loss gap by 11.03% on two real datasets. The details of `gTFP` are documented in Section 4.1, with its corresponding analysis in RQ.1 and RQ.2 of Section 5.

(3) We devise a unified operator-learning view within `gTFP` that avoids hand-crafted adjacency in implicit solvers, yielding up to 2× training speedup (backbone-dependent) without sacrificing accuracy. The design of this solver is in Section 4.3, with its comparative performance in RQ.3 – RQ.5 of Section 5.

2 Preliminaries

Notation and Problem Statement. Let $G = (V, E)$ denote a directed fluid network, where nodes V represent local observation points and edges E indicate the direction of flow. We define the adjacency $\mathbf{A} \in \{0,1\}^{|V|\times|V|}$, such that $\mathbf{A}_{i,j} = 1$ if there exists a directed edge from node v_i to v_j, and $\mathbf{A}_{i,j} = 0$ otherwise. Note that $\mathbf{A} \neq \mathbf{A}^\top$. A node v_j is said to be a neighbor of v_i, denoted $v_j \in \mathcal{N}(i)$, if $\mathbf{A}_{j,i} = 1$.

At time t, each node $v_i \in V$ is associated with a feature matrix $\mathbf{h}_i \in \mathbb{R}^{W\times d}$, which stores d physical measurements (*e.g.,* velocity, pressure, slope) over a historical window of W time steps. Stacking features across all nodes yields a tensor $\mathbf{H} = [\mathbf{h}_1, \dots \mathbf{h}_{|V|}]^\top \in \mathbb{R}^{|V|\times W\times d}$. The goal of flux prediction is to learn a predictive model f that forecasts a target physical quantity (*e.g.,* flux volume) at a future step $t + n$, with n the prediction horizon. Let $\mathbf{y} \in \mathbb{R}^{|V|}$ denote the ground-truth values of this target quantity. Our learning objective is to minimize the empirical loss $\ell\big(\mathbf{y}, f(\mathbf{H}, \mathbf{A})\big)$.

We define the *boundary nodes* V_{BN}, as those with zero in-degree, *i.e.,* $\deg^-(v_b) = 0,\ \forall v_b \in V_{\text{BN}}$. Intuitively, these node lie at the most upstream point of G, regulate the influx into the whole system as the primary interface with external forcing. The remaining node are referred to as *interior nodes*, defined as $V_{\text{IN}} = V \backslash V_{\text{BN}}$.

3 Problem Analysis

A paradoxical observation is that if the predictive model f is instantiated as GNN, it often underperforms simple baselines (*e.g.,* MLP) and shows little difference between predictions computed from $f(\mathbf{H}, \mathbf{A})$ and $f(\mathbf{H}, \mathbf{A}^\top)$. Here, as $\mathbf{A}$ encodes the ground-truth forward fluid flow, $\mathbf{A}^\top$ represents a physically-implausible, reversed flow. Kirschstein & Sun (2024) concludes that GNNs may not work in flux prediction tasks, as they fail to distinguish directional flow consistency, which is fundamental in physical systems.

3.1 GNNs as Neural Differential Equations.

We argue that such a conclusion is premature and advocate an alternative interpretation through the lens of numerical solvers (Liu et al., 2025). Write a standard message-passing update as

$$\mathbf{h}_i^{(l+1)} = \mathbf{h}_i^{(l)} + \textstyle\sum_{v_j\in\mathcal{N}(i)} \psi^{(l)}(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}), \quad \mathbf{h}_i^{(0)} = \mathbf{h}_i, \tag{1}$$

where $\mathbf{h}_i^{(l)}$ denotes the representation of node v_i at layer l, and $\psi^{(l)}$ is the message function aggregating information from its upstream neighbors $v_j \in \mathcal{N}(i)$. We view Eq. (1) as an explicit integration scheme, that mimics a partial differential equations (PDEs) solver (LeVeque, 2007), such as

$$u^{t+1}(x_i) = u^t(x_i) + \Delta t \cdot F(u^t(x_i), u^t(x_{i+1})), \tag{2}$$

$$\text{s.t.}\quad u^t(x_{i+1}) \approx B\big(u^t(x_i), \partial_x u^t(x_i)\big), \quad \forall v_i \in \mathcal{V}_{BN}. \tag{3}$$

where $u^t(x_i)$ is the state of a physical quantity (*e.g.,* flux volume) at location x_i and time t, and F represents the spatial derivative or transport dynamics. Consider, for example, a discretized advection equation (Chock, 1991) that implements $F(u^t(x_i), u^t(x_{i+1})) = \frac{\partial}{\partial x}(u^t(x_{i+1}) - u^t(x_i))$, which models the propagation of the quantity through its upstream neighbor x_{i+1}. We can observe an algebraic similarity between Eq. (1) and Eq. (2), where the GNN layer index l parallels the time step t, and the message function ψ acts as the discrete derivative F across the graph topology. At boundary locations, Eq. (3) uses a boundary operator $B(\cdot)$ to approximate the out-of-domain upstream value required to evaluate F.

A natural question: if PDE solvers operate robustly and in a topology-aware manner for fluid systems, and message-passing mimics their computational structure, then *why do GNN empirically fail on the same task?*

External Forcing Deficit. We hypothesize that message passing lacks an explicit mechanism to infer unobserved external forcing at upstream boundary nodes; we refer to this as an external forcing deficit. PDE solvers explicitly enforce such conditions, as seen in Eq. (3), which closes the missing upstream state so that the flux term F in Eq. (2) is well-defined at the boundary location x_b. Intuitively, when x_b lies at the spatial boundary, its upstream neighbor x_{b+1} is undefined, invalidating the computation of the flux term F in Eq. (2) that depends on $u^t(x_{b+1})$. The boundary condition in Eq. (3) complements this by approximating the out-of-domain upstream value from the local state $u^t(x_b)$. To wit, we instantiate the boundary operator $B(\cdot)$ in Eq. (3) with a Robin-type boundary condition (Busse et al., 2017):

$$u^t(x_{b+1}) = \omega_1 \cdot u^t(x_b) + \omega_2 \cdot \partial u^t(x_b)/\partial x, \tag{4}$$

with $\omega_1, \omega_2 \in \mathbb{R}$, which closes the dynamical system by postulating an interpolation between x_b and its spatial derivative to approximate the undefined, out-of-boundary x_{b+1}.

In contrast, GNNs lack a mechanism to infer the missing external forcing at graph boundaries. For a boundary node v_b which, by definition, has no incoming edge and thus no upstream neighbor, namely $\mathcal{N}(b) = \emptyset$. As a result, the boundary node has no upstream inputs to reflect external forcing, Eq. (1) collapses to $\mathbf{h}_b^{(l+1)} = \mathbf{h}_b^{(l)} + \psi^{(l)}(\mathbf{h}_b^{(l)}, \mathbf{0})$, meaning that the update of v_b depends solely on its own features and receives no information from the graph topology. This isolation over successive layers leads to degraded boundary node embeddings and, eventually, to substantial prediction errors.

3.2 Empirical Validation.

To validate our hypothesis, we analyze and compare the prediction losses of ResGAT (Residual Graph Attention Networks) and MLP by separating the errors incurred at boundary versus interior nodes. The results are summarized in Table 1.

We make two observations from these results. ***First***, the overall mean squared error (MSE) misleadingly suggests that the GNN underperforms an MLP (.1166 > .1135), when in fact the GNN performs substantially better in regions where it can leverage graph topology. The failure of GNN is mainly attributed to the boundary nodes, incurring a .1401 MSE. This large boundary loss obscures the otherwise strong performance of GNN-based flux prediction on interior nodes, where MSE drops to .0836. ***Second***, the seemingly similar overall losses using the ground-truth forward flow $\mathbf{A}$ (.1166) and the reverse flow $\mathbf{A}^\top$ (.1179) are deceptive. The forward model is disproportionately penalized by boundary node errors, which suppress its average performance and mask its superiority over the reversed model. When focusing on interior nodes only, the forward model achieves an MSE of .0836, outperforming the reversed model at .0939. This aligns with physical intuition and demonstrates the positive impact of graph topology on learning meaningful representations in regions where directional flow information is available.

Table 1: Node-wise MSE Comparison

Node Type	**MSE For. (A)**	**MSE Rev. ($\mathbf{A}^\top$)**
Boundary (V_{BN})	.1401	.1350
Interior (V_{IN})	.0836	.0939
All nodes (V)	.1166	.1179
MLP	.1135	

Note, these MSE results are normalized, where .001 change means 1% flux volume change. For a mid-size river network (Discharge: 100m^3/s), a .01 MSE error in discharge prediction equals a daily volume discrepancy of 86,400m^3, which equals to ∼35 Olympic pools. This may cause critical failure with cascading consequences, *e.g.,* threaten the survival of aquatic species (Poff et al., 1997), lead to dangerous underestimations of pollution risk (Whitehead et al., 2009), and be amplified into financial losses for hydropower and navigation (Lehner et al., 2005; Jonkeren et al., 2007).

4 The gTFP Approach

This section presents our gTFP framework. Standard message-passing GNNs do not explicitly model upstream boundary influence, often yielding degraded accuracy near system boundaries. To address this, we introduce ghost nodes as boundary-aware proxies that infer missing external forcing. We then solve the induced coupled boundary–interior update using either an implicit or an explicit solver. In Section 4.1, we describe

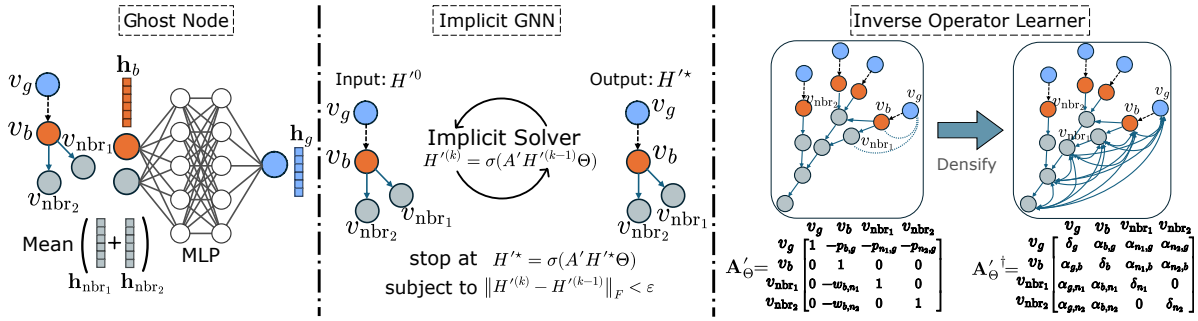


Figure 2: Overview of the proposed **gTFP** framework. Left: Ghost-node construction (Section 4.1). For each boundary node v_b, we introduce a corresponding ghost node v_g. Its embedding $\mathbf{h}_g$ is learned from the boundary embedding $\mathbf{h}_b$ and the aggregated embeddings of its downstream interior neighbors $\mathbf{h}_{\text{nbr}}$. Middle: Implicit solver (Section 4.2). We perform implicit message passing on the augmented graph with ghost nodes using a fixed-point solver, converging to $\mathbf{H}'^\star$. Shared parameters Θ are applied consistently across updates. Right: Explicit inverse-operator learning (Section 4.3). We learn an explicit inverse operator on the densified adjacency $\mathbf{A}'^{\dagger}_{\Theta}$, enabling layer-wise updates through a learnable inverse mapping.

how to construct and learn ghost-node embeddings from surrounding interior nodes to infer the missing external forcing at boundaries. Section 4.2 formulates an implicit solver that integrates ghost nodes into a fixed-point framework inspired by numerical PDE methods. Section 4.3 further provides a computationally efficient, explicit counterpart by learning a dense inverse operator, allowing layer-wise updates that preserve boundary-awareness while avoiding the cost of fixed-point iteration.

4.1 Learn Ghost Node Proxies

To remedy the external forcing deficit in GNNs, we learn ghost-node proxies from data via a numerical ghost-node construction (Tseng & Ferziger, 2003). Let v_g denote a virtual ghost node corresponding to a boundary node v_b, and v_{nbr} denote the interior (downstream) neighbor of v_b, such that $v_{\text{nbr}} \in \{v_j \mid v_b \in \mathcal{N}(j)\}$. We draw insights from PDE solvers to learn the embedding of v_g from the coupling among v_g, v_b, and v_{nbr}. Specifically, we instantiate the boundary operator $B(\cdot)$ in Eq. (3) with a Robin-type boundary condition (Busse et al., 2017) and discretize it to derive: (we compare Dirichlet & Neumann variants in App. A.1).

$$\mathbf{h}_g^{(l)} = \omega_1 \cdot \mathbf{h}_b^{(l)} + \omega_2 \cdot (\mathbf{h}_b^{(l)} - \mathbf{h}_{\text{nbr}}^{(l)})/\Delta x, \tag{5}$$

where $\mathbf{h}_{\text{nbr}}^{(l)}$ is the embedding of v_{nbr} at the l-th layer. In implementation, we parameterize ω_1 and ω_2 via an MLP, defined as $\mathbf{h}_g^{(l)} = \text{MLP}(\text{Concat}(\mathbf{h}_b^{(l)}, \mathbf{h}_{\text{nbr}}^{(l)}); \theta_{\text{gh}})$. Main steps for ghost-node learning are presented in Algorithm A.7.1 (in App. A.7).

The total number of such ghost nodes equates to boundary nodes. Defining the set of ghost nodes $V_{\text{GH}} = \{v_g\}$, we have $|V_{\text{GH}}| = |V_{\text{BN}}|$. To proceed message-passing, we define a graph augmentation operator $\mathcal{A}$, which takes the original graph as inputs and augments it with ghost nodes as follows.

$$(\mathbf{A}', \mathbf{H}') = \mathcal{A}(\mathbf{A}, \mathbf{H}), \tag{6}$$

where $\mathbf{H}' = [\mathbf{H}, \{\mathbf{h}_g\}_{|V_{\text{BN}}|}]^\top \in \mathbb{R}^{(|V|+|V_{\text{GH}}|)\times W\times d}$ denotes the augmented node feature matrix, and $\mathbf{A}' \in \{0,1\}^{(|V|+|V_{\text{GH}}|)\times(|V|+|V_{\text{GH}}|)}$ is the augmented adjacency, such that $\mathbf{A}'_{i,j} = 1$ if v_i is an (upstream) neighbor of v_j, namely $v_i \in \mathcal{N}(j)$, and $\mathbf{A}'_{i,j} = 0$ otherwise.

The augmentation in Eq. (6) connects each ghost node to a downstream boundary neighbor in a deterministic way, which challenges layer-by-layer message passing. Specifically in numerical methods, to improve numerical stability one often adopts implicit time discretization, which enforces the Robin-type boundary condition via ghost nodes through a time-coupled relation involving both boundary and ghost states at the new time level (Tseng & Ferziger, 2003), e.g.,

$$(1 - \frac{\delta_1^2 \Delta t}{\delta_2})u^{t+1}(x_b) + (\frac{\delta_1 \Delta t}{\delta_2})u^{t+1}(x_{\text{g}}) = u^t(x_b), \tag{7}$$

where $\delta_1, \delta_2 \in \mathbb{R}$ are two physical coefficients. The derivation from Eq. (2) and Eq. (3) to Eq. (7) are deferred to App. A.2 due to the space limit. Drawing analogy to Eq. (7), learning the ghost node embeddings is constrained by $\alpha_1 \cdot \mathbf{h}_b^{(l+1)} + \alpha_2 \cdot \mathbf{h}_g^{(l+1)} = \mathbf{h}_b^{(l)}$, $\exists \alpha_1, \alpha_2 \in \mathbb{R}$, resulting in the message-passing on boundary node as:

$$\mathbf{h}_b^{(l+1)} = \mathbf{h}_b^{(l)} + \sum_{v_g \in \mathcal{N}(b)} \psi^{(l)}(\mathbf{h}_b^{(l)}, \mathbf{h}_g^{(l+1)}) \tag{8}$$

As the implicit update in Eq. (8) shows, computing the boundary embedding $\mathbf{h}_b^{(l+1)}$ requires the ghost embedding $\mathbf{h}_g^{(l+1)}$. Meanwhile, we enforce the ghost-proxy relation Eq. (5) at layer $l+1$ to close the boundary condition within the same layer update, which makes $\mathbf{h}_g^{(l+1)}$ depend on $\mathbf{h}_b^{(l+1)}$ in return. Together, $\mathbf{h}_b^{(l+1)}$ and $\mathbf{h}_g^{(l+1)}$ are jointly determined by a coupled system, which we reformulate in the next subsection to motivate an implicit fixed-point solver.

To proceed, we can rewrite the analogous implicit update for interior nodes from Eq. (8). For interior nodes $\forall v_i \in V_{\text{IN}}$, they are governed by the standard implicit scheme as

$$\alpha_1 \, \mathbf{h}_i^{(l+1)} - \alpha_2 \, \mathbf{h}_{i+1}^{(l+1)} = \mathbf{h}_i^{(l)}, \tag{9}$$

where $\mathbf{h}_i^{(l+1)}$ and $\mathbf{h}_{i+1}^{(l+1)}$ are the embeddings of node v_i and its upstream neighbor v_{i+1} at the $(l+1)$-th layer, respectively. Moreover, rearranging the ghost proxy in Eq. (5) into an implicit form yields

$$\Big(\omega_1 + \frac{\omega_2}{\Delta x}\Big)\mathbf{h}_b^{(l+1)} - \Big(\frac{\omega_2}{\Delta x}\Big)\mathbf{h}_{\text{nbr}}^{(l+1)} - \mathbf{h}_g^{(l+1)} = 0. \tag{10}$$

Assembling these equations yields the unified augmented system:

$$\begin{pmatrix} \alpha_1 & 0 & \cdots & 0 & \alpha_2 \\ -\alpha_2 & \alpha_1 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & -\alpha_2 & \alpha_1 & 0 \\ \omega_1 + \dfrac{\omega_2}{\Delta x} & \dfrac{\omega_2}{\Delta x} & \cdots & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{h}_b^{(l+1)} \\ \mathbf{h}_{M[b]}^{(l+1)} \\ \vdots \\ \mathbf{h}_1^{(l+1)} \\ \mathbf{h}_g^{(l+1)} \end{pmatrix} = \begin{pmatrix} \mathbf{h}_b^{(l)} \\ \mathbf{h}_{M[b]}^{(l)} \\ \vdots \\ \mathbf{h}_1^{(l)} \\ 0 \end{pmatrix}, \tag{11}$$

where $M[b] \in \mathbb{N}$ denotes the number of interior nodes on the branch which starts at a boundary node v_b. The indices $M[b], \ldots, 1$ enumerate the interior nodes along this branch, starting from the interior node $v_{M[b]}$, being an immediate neighbor of v_b, and following the flow direction toward the farthest end node v_1. For multiple upstream branches, we assemble one such block in App. A.3. This reduces Eq. (11) to the compact form $\mathbf{A}' \, \mathbf{H}'^{(l+1)} = \mathbf{H}'^{(l)}$, with $\mathbf{A}' \in \mathbb{R}^{(M[b]+2)\times(M[b]+2)}$. This leads to a coupled system (LeVeque, 2007), necessitating an implicit solver via fixed-point iteration.

4.2 Implicit Fixed-Point Solver

We treat the ghost-augmented boundary–interior coupling in Eq. (11) as a coupled system and solve it using an implicit fixed-point solver (i.e., Implicit GNN (Gu et al., 2020)). This step jointly updates the boundary and interior embeddings by solving for a consistent equilibrium, similar to implicit schemes in numerical PDE solvers, and serves as a benchmark for the explicit inverse-operator solver in Sec. 4.3. We define an equilibrium mapping on the ghost-augmented embeddings $\mathbf{H}'$ and use its fixed point as the implicit update.

At $l+1$, we learn the predictor and solver parameters by minimizing the prediction loss on the *original* nodes:

$$\min_{f,\Theta,\theta_{\text{gh}},C} \; \ell\Big(\mathbf{y},\; f\Big(\mathbf{H}^{(l+1)}\Big)\Big), \qquad \mathbf{H}^{(l+1)} := \Pi_A\big(\mathbf{H}_\star'^{(l+1)}\big). \tag{12}$$

Here $\Pi_A(\cdot)$ projects embeddings from the augmented space (including ghost nodes) back to the original node indices in G. This projection is used because ghost nodes are unlabeled, so supervision is defined only on the original nodes. We use $\mathbf{H}'$ to denote ghost-augmented node features, where the ghost components are

produced by the proxy module in Eq. (5) parameterized by θ_{gh}. Here Θ and C are learnable parameters in the IGNN equilibrium equation, whose roles are specified in Eq. (13).

To approximately realize the coupled update implied by Eq. (11) within a learnable GNN architecture, we define $\mathbf{H}_\star'^{(l+1)}$ as the fixed point of

$$\mathbf{H}_\star'^{(l+1)} = \sigma\Big(\mathbf{A}'\,\mathbf{H}_\star'^{(l+1)}\,\Theta + C\,\mathbf{H}'^{(l)}\Big)\,, \tag{13}$$

where $\sigma(\cdot)$ is a non-linear activation. The weights Θ control the embedding transformation in the equilibrium equation, while C conditions the equation on the layer input $\mathbf{H}'^{(l)}$. Eq. (13) provides a learnable fixed-point solver for the same-layer closure of the ghost-augmented boundary–interior coupling, thereby realizing the coupled update implied by Eq. (11).

Since Eq. (13) typically has no closed-form solution, we approximate $\mathbf{H}_\star'^{(l+1)}$ via K inner fixed-point iterations. We initialize $\mathbf{H}'^{(k=0)} = \mathbf{H}'^{(l)}$ and iterate for $k = 0, 1, \ldots, K-1$: $\mathbf{H}'^{(k+1)} = \sigma\big(\mathbf{A}'\,\mathbf{H}'^{(k)}\,\Theta + C\,\mathbf{H}'^{(l)}\big)$. After K steps, we take the numerical equilibrium solution as $\mathbf{H}_\star'^{(l+1)} := \mathbf{H}'^{(K)}$. Here k indexes the inner fixed-point iterations used to compute the equilibrium at $l{+}1$. The inner loop can stop early when $\|\mathbf{H}'^{(k)} - \mathbf{H}'^{(k-1)}\|_F < \varepsilon$, or stop at the maximum iteration budget K. Gradients w.r.t. Θ and C are obtained through the fixed-point computation, while gradients w.r.t. θ_{gh} backpropagate through the ghost components of $\mathbf{H}'$ generated by Eq. (5) (Gu et al., 2020; Chen et al., 2023).

To ensure existence and uniqueness of the fixed point (and hence stable convergence of the inner iterations), we follow (Gu et al., 2020) and impose a Perron–Frobenius (PF) spectral constraint so that the equilibrium mapping is a contraction. Intuitively, if the effective spectral strength induced by $\mathbf{A}'$ and Θ is too large, the fixed-point iteration may diverge.

To analyze the spectral strength of the bilinear term $\mathbf{A}'\mathbf{H}'\Theta$, we vectorize it as $\text{vec}(\mathbf{A}'\mathbf{H}'\Theta) = \big(\Theta^\top \otimes \mathbf{A}'\big)\text{vec}(\mathbf{H}')$, where $\otimes$ denotes the Kronecker product, which isolates the roles of the augmented $\mathbf{A}'$ and the propagation weights Θ (Schacke, 2004). Note that the constant input term $C\,\mathbf{H}'^{(l)}$ does not affect contraction since it does not depend on $\mathbf{H}'$. Let $\lambda_{\text{pf}}(\mathbf{A}')$ be the PF eigenvalue (largest eigenvalue) of $\mathbf{A}'$ (Berman & Plemmons, 1994) and let $\|\Theta\|_\infty$ be the maximum absolute row-sum norm, which provides a convex upper bound on its spectral norm (Zheng & Wang, 2008). Using the standard IGNN bound (Gu et al., 2020), we enforce the strict contraction condition $\lambda_{\text{pf}}(\mathbf{A}')\,\|\Theta\|_\infty < 1$, which guarantees a unique fixed point and stabilizes implicit updates. In practice, we adopt the PF solver/normalization strategy in Gu et al. (2020) to maintain this constraint during training. Algorithm A.7.2 (App. A.7) summarizes the main steps.

4.3 Explicit Inverse-Operator Solver

While the ghost nodes compensate for the external forcing deficit, and Eq. (13) provides an implicit yet effective solution for them, this solution suffers from two key efficiency limitations. First, as shown in Figure 4a, although the ghost node-enhanced implicit GNN reduces the prediction loss on boundary nodes by 7.2 %, it suffers from high computational cost, running approximately 13× slower than standard layer-wise message-passing GNNs. Second, the structure of the augmented adjacency $\mathbf{A}'$ can vary *w.r.t.* the modeling choice of boundary condition. In practice, different numerical schemes, *e.g.,* Robin-type as we used in Eq. (4), can be used to impose boundary constraint on the same physical system. As such, our implicit solver that assumes a fixed $\mathbf{A}'$ loses flexibility in adapting to such variations to learn ghost node proxies.

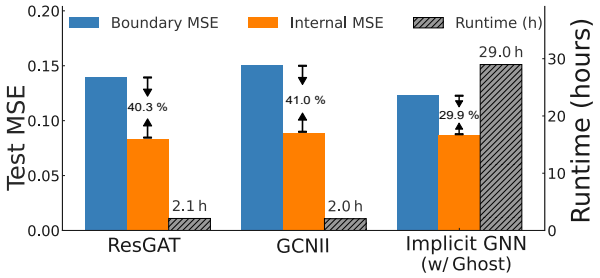


Figure 3: Boundary vs. interior MSE (two left axes) and wall-clock runtime (right axis) for two backbone GNNs and our proposed implicit solver with ghost nodes. Observe that while the implicit solver reduces prediction error at boundary nodes and mitigates the boundary-interior loss gap, it incurs a runtime overhead of over 14× compared to standard GNNs.

These limitations motivate us to tailor a layer-by-layer counterpart to the implicit solver in Eq. (13). By learning ghost nodes, it encodes boundary conditions and mitigates the external forcing deficit at boundaries, while preserving the computational efficiency of a standard GNN.

Observing Eqs. (5), (6), and (8), we note that although $\mathbf{A}'$ connects each ghost node v_g only to its downstream boundary node v_b, the computation of $\mathbf{h}_g$ in fact depends on both $\mathbf{h}_b$ and the interior neighbor embeddings $\mathbf{h}_{\text{nbr}}$. This reveals an implicit computational graph that spans $\mathbf{h}_g$, $\mathbf{h}_b$, and $\mathbf{h}_{\text{nbr}}$, lending to consider whether we can construct a weighted adjacency matrix $\mathbf{A}'_{\boldsymbol{\Theta}} \in \mathbb{R}^{(|V|+|V_{\text{GH}}|)\times(|V|+|V_{\text{GH}}|)}$ that captures these interactions in a layer-wise message-passing regime.

To this end, we decompose the augmented adjacency $\mathbf{A}'$ into two subgraphs. The ***first*** is a standard GNN over the augmented node set $V \cup V_{\text{GH}}$, where each node (whether interior, boundary, or ghost) participates in message passing. We let $w_{i,j}$ denote the message-passing weight from node v_j to v_i, with $v_i, v_j \in V \cup V_{\text{GH}}$. The ***second*** is a bipartite subgraph linking ghost nodes V_{GH} to original nodes V, which reflects how ghost nodes are constructed from the boundary condition. Let $p_{i,j}$ denote the interpolation weight from an original node $v_i \in V$ to a ghost node $v_j \in V_{\text{GH}}$, of which the edges are restricted to connect nodes across the bipartition. This allows us to encode various boundary condition types in a unified message-passing framework. For example, write $\mathbf{h}_g$ and $\mathbf{h}_b$ the ghost and boundary node embeddings, respectively, and let $\mathbf{h}_{\text{nbr}_1}$, $\mathbf{h}_{\text{nbr}_2}$ denote the embeddings of the immediate and second-order interior (downstream) neighbors of boundary, respectively. Under a first-order condition, the ghost node proxy is defined as $\mathbf{h}_g = p_{b,g}\mathbf{h}_b + p_{n_1,g}\mathbf{h}_{\text{nbr}_1}$. For second-order, it becomes $\mathbf{h}_g = p_{b,g}\mathbf{h}_b + p_{n_1,g}\mathbf{h}_{\text{nbr}_1} + p_{n_2,g}\mathbf{h}_{\text{nbr}_2}$. Each such augmentation changes the structure of the rows in $\mathbf{A}'$ corresponding to v_g, introducing new learnable parameters in $\mathbf{A}'_{\boldsymbol{\Theta}}$. Upon these intuitions, we define (i,j)-th entry of $\mathbf{A}'_{\boldsymbol{\Theta}}$ as follows.

$$\mathbf{A}'_{\boldsymbol{\Theta}}[i,j] = \begin{cases} 1 & \text{if } i = j \\ p_{i,j} & \text{if } v_i \in V \text{ and } v_j \in V_{\text{GH}} \\ w_{i,j} & \text{if } v_i \in V \text{ and } v_j \in \mathcal{N}(i) \\ 0 & \text{otherwise} \end{cases}.$$

This $\mathbf{A}'_{\boldsymbol{\Theta}}$ reduces the implicit solver in Eq. (13) to an explicit solution linear system $\Pi_{\mathbf{A}}\big(\mathbf{A}'_{\boldsymbol{\Theta}}\mathbf{H}'^{(l+1)}\big) = \mathbf{H}^{(l)}$, which enjoys a closed-form solution $\mathbf{H}'^{(l+1)} = \mathbf{A}'^{\dagger}_{\boldsymbol{\Theta}}\mathbf{H}'$. Here, $\dagger$ denotes the Moore-Penrose inverse (Prasad & Bapat, 1992), and the ghost node-padded tensor $\mathbf{H}'$ is defined in Eq. (6). We present a concrete examples in App. A.3. One property of the inverse is producing dense matrices (Chamberlain et al., 2021), which reflects the global coupling of the system and allows to bypass the limitation of using a sparse, hand-crafted $\mathbf{A}'$. To operationalize this idea, we learn $\mathbf{A}'^{\dagger}_{\boldsymbol{\Theta}}$ by approximating the inverse operator through a trainable GNN as

$$\begin{aligned} &\min_{f,\theta_{\text{gh}},\Psi} \ell\big(\mathbf{y},\ f\big(\Pi_A(\mathbf{H}'^{(l+1)})\big)\big), \\ &\text{s.t. } \ \mathbf{H}'^{(l+1)} = \mathbf{A}'^{\dagger}_{\boldsymbol{\Theta}}\mathbf{H}'^{(l)}, \mathbf{A}'^{\dagger}_{\boldsymbol{\Theta}} = \Psi(\mathbf{A}'_{\boldsymbol{\Theta}}), \mathbf{A}'^{\dagger}_{\boldsymbol{\Theta}}\mathbf{A}'_{\boldsymbol{\Theta}} \approx \mathbf{I}, \end{aligned} \tag{14}$$

where the goal is to learn an inverse operator Ψ that approximates $\mathbf{A}'^{\dagger}_{\boldsymbol{\Theta}}$, and the regularization term $\mathbf{A}'^{\dagger}_{\boldsymbol{\Theta}}\mathbf{A}'_{\boldsymbol{\Theta}} \approx \mathbf{I}$ enforces an inverse constraint. We learn this operator because analytical inversion can be computationally expensive and unstable, especially as $\mathbf{A}'_{\boldsymbol{\Theta}}$ may vary across samples and training iterations. A learned Ψ provides an approximation that generalizes across boundary structures. In implementation, we can parameterize Ψ using a differentiable architecture such as a GNN or low-rank factorization. For the explicit inverse-operator learner, see Algorithm A.7.3 (in App. A.7).

5 Experiments

Datasets. We evaluate on two directed-network dataset. **(i) River.** A real world river network preprocessed from LamaH-CE2 (Klingler et al., 2021) over the Danube basin, providing hourly discharge and meteorological records. The graph has 358 nodes and 357 directed edges, partitioned into 209 boundary and 149 interior nodes. Each node has five features: discharge, surface air pressure, precipitation, temperature, and soil moisture. **(ii) Blood flow.** A simulated arterial-network dataset generated with *openBF* (Benemerito

Table 2: MSE comparison for River (left) and Blood (right), with Boundary/Interior breakdown. Shaded rows show group means: *Avg (Base)* averages the three baselines (GCNII, ResGCN, ResGAT), and $\texttt{gTFP}_{Avg}$ averages their ghost counterparts. *Diff (%)* is the relative difference between boundary and interior. The rightmost column reports the relative average runtime.

Flux Predictors	**River**				**Blood**				**Runtime (Avg., rel.)**
	Avg.	**Boundary**	**Interior**	**Diff. (%)**	**Avg.**	**Boundary**	**Interior**	**Diff. (%)**	
GCNII	0.1253	0.1511	0.0891	41.03	0.0674	0.1451	0.0363	74.98	×1.0
ResGCN	0.1232	0.1408	0.0985	30.04	0.0569	0.1140	0.0341	70.09	
ResGAT	0.1166	0.1401	0.0836	40.33	0.0482	0.1056	0.0252	76.14	
Avg (Base)	0.1217	0.1440	0.0904	37.13	0.0575	0.1216	0.0319	73.74	
$\texttt{gTFP}_{\text{GCNII}}$	0.1121	0.1274	0.0906	28.89	0.0638	0.1335	0.0359	73.11	×2.6
$\texttt{gTFP}_{\text{ResGCN}}$	0.1162	0.1313	0.0950	27.65	0.0543	0.1072	0.0331	69.12	
$\texttt{gTFP}_{\text{ResGAT}}$	0.1063	0.1251	0.0800	36.05	0.0457	0.0982	0.0247	74.85	
$\texttt{gTFP}_{\text{Avg}}$	0.1115	0.1279	0.0885	30.86	0.0546	0.1130	0.0312	72.36	
Implicit GNN	0.1119	0.1273	0.0902	29.14	0.0586	0.0743	0.0523	29.58	×13.1
Implicit GNN w/ Ghost	0.1082	0.1236	0.0866	29.94	0.0557	0.0708	0.0497	29.86	×13.8

et al., 2024), on a Circle of Willis model (Vrselja et al., 2014); we use a connected subnetwork with 14 nodes and 14 directed edges. Each node carries four features: flux, pressure, velocity, and cross-sectional area. For both datasets, model inputs are formed by concatenating the past W hours of features along the channel dimension; the prediction target is flux $\mathbf{y} \in \mathbb{R}^{|V|}$ at horizon $t{+}n$. All variables are independently normalized via z-score to ensure consistent scaling across nodes and variables.

Metrics. We evaluate models under a supervised node regression setup, following (Kirschstein & Sun, 2024; Jiang et al., 2025). Given W hours of historical flux data for all nodes, the goal is to predict the flux volume n hours ahead. In our setting, we use $W = 24$ and a forecast horizon of $n = 6$ hours. We compute the mean-squared error (MSE) on three node sets: all nodes $\ell(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{|V|}\sum_{v_i \in V}(\hat{y}_i - y_i)^2$, boundary nodes $\ell_{BN}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{|V_{\text{BN}}|}\sum_{v_i \in V_{\text{BN}}}(\hat{y}_i - y_i)^2$, and interior nodes $\ell_{IN}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{|V_{\text{IN}}|}\sum_{v_i \in V_{\text{IN}}}(\hat{y}_i - y_i)^2$.

Competitors. We compare with several GNN baselines, including residual variants of graph convolutional networks (ResGCN) (Wu et al., 2019), graph attention networks (ResGAT) (Veličković et al., 2017) and GCNII (Chen et al., 2020). They also serve as backbone for `gTFP`. We additionally include physics-aware graph models (MP-PDE (Brandstetter et al., 2022) and GNO (Li et al., 2020)) as stronger competitors. We further compare against a dense graph transformation as ablation study (Wang et al., 2025).

Implementation. We implement the Robin-type boundary condition to model the structural coupling among v_g, v_b, and v_{nbr}, as defined in Eq. (5). The message-passing layers are implemented using ResGCN, ResGAT and GCNII, with 128 dimensional node embedding, applying **ReLU** activation to every non-linear layer. We adapt the Picard search method (Paniconi & Putti, 1994) to accelerate the implicit GNN training, following Gu et al. (2020). We use a fully connected and trainable adjacency matrix to learn the inverse operator. For ablation study, we learn it over symmetric (*i.e.,* $\mathbf{A}'^{\dagger}_{\boldsymbol{\Theta}}$ in Eq. (14)) and asymmetric (*i.e.,* $\mathbf{A}'^{\dagger}\Theta$, $\mathbf{A}'$ in Eq. (6) and Θ in Eq. (13)) versions, following Wang et al. (2025), with detailed results and analysis deferred to RQ5. For the compared models, we implement their GNN architectures following Kirschstein & Sun (2024) and evaluate their results on both ground-truth, forward flow (*i.e.,* $\mathbf{A}$) and the physically-implausible, reverse flow (*i.e.,* $\mathbf{A}^{\top}$), which enables to verify whether adding boundary condition will improve their topology-awareness during training, with results deferred to RQ2.

Results and Analysis

Based on the results in Table 2, 3 and 4, we answer the following research questions (**RQ1–5**):

RQ1. *To what extent can ghost nodes compensate external forcing deficit in GNN training?*

Table 3: Topological comparison of Fwd, Rev, and gTFP on River and Blood. In each cell, the first line shows Avg with its change relative to Rev for the same backbone and dataset; the second line (with results parenthesized) shows Boundary with its change relative to Rev. ↑ indicates better (lower MSE), and ↓ indicates worse (higher MSE).

	River			Blood		
Backbone	**Fwd**	**Rev**	**gTFP**	**Fwd**	**Rev**	**gTFP**
GCNII	0.1253 (↑ 2.8%) (0.1511 (↓ 1.5%))	0.1289 (0.1488)	0.1121 (↑ 13.0%) (0.1274 (↑ 14.4%))	0.0674 (↓ 3.7%) (0.1451 (↓ 9.1%))	0.0650 (0.1330)	0.0638 (↑ 1.8%) (0.1335 (↓ 0.4%))
ResGCN	0.1232 (↑ 1.0%) (0.1408 (↓ 2.8%))	0.1245 (0.1369)	0.1162 (↑ 6.7%) (0.1313 (↑ 4.1%))	0.0569 (↓ 3.1%) (0.1140 (↓ 9.4%))	0.0552 (0.1042)	0.0543 (↑ 1.6%) (0.1072 (↓ 2.9%))
ResGAT	0.1166 (↑ 1.1%) (0.1401 (↓ 3.8%))	0.1179 (0.1350)	0.1063 (↑ 9.8%) (0.1251 (↑ 7.3%))	0.0482 (↓ 4.6%) (0.1056 (↓ 9.9%))	0.0461 (0.0961)	0.0457 (↑ 0.9%) (0.0982 (↓ 2.2%))

We assess this question by linking our learned ghost nodes via Eq. (5) to the boundary nodes of three GNN backbones (ResGAT, ResGCN, GCNII), and measuring their impact on boundary and overall MSE performance. Compared to the MLP baseline, on **River**, adding ghost nodes reduces the boundary MSE by 10.7%, 6.7%, and 15.7%, and reduces overall MSE by 8.83%, 5.68%, 10.53%, for ResGAT, ResGCN, and GCNII, respectively. All three backbones now outperform the MLP baseline (0.1135) on overall MSE by an average of 1.8%. On **Blood**, ghost nodes similarly reduce boundary MSE by 7.01%, 5.96%, and 7.99%, and decrease overall MSE by 5.19%, 4.57%, and 5.34% for ResGAT, ResGCN, and GCNII, respectively. The three backbones outperform the MLP baseline (0.0670) on overall MSE by an average of 17.5%. Averaged across the three GNN backbones, ghost nodes reduce overall MSE by 8.3% and 5.0%, while reducing boundary MSE by 11.0% and 6.9% on River and Blood, respectively. We further show that, although MP-PDE (Brandstetter et al., 2022) and GNO (Li et al., 2020) reduce overall MSE, they still leave a large boundary–interior gap, whereas gTFP achieves larger boundary-side gains (App. A.5.2). Such improvement indicates that adding ghost nodes consistently strengthens GNN performance relative to the MLP baseline, substantiating that the external forcing deficit has been indeed compensated through the learned proxies.

RQ2. *Will adding ghost nodes improve the topology-awareness of standard GNNs?*

We evaluate this by comparing the performance of each backbone under forward (true edge direction) and reverse (all edges flipped) graph variants, both before and after adding ghost nodes. We use the relative forward–reverse MSE difference in Table 3 as a metric of topology-awareness.

On **River**, without ghost nodes the forward-reverse gaps for the three GNN backbones are only 1.1%, 1.1%, and 2.9%, indicating minimal sensitivity to edge direction. After adding ghost nodes, these gaps increase substantially to 10.9%, 6.7%, and 15.0%, corresponding to approximately 8.9×, 6.3×, and 4.7× improvement in topology-awareness. On **Blood**, the baselines even *prefer* the reversed graph. We observe that the forward variants perform worse than their reversed counterparts by 4.56%, 3.08%, and 3.69%. With ghost nodes, this trend reverses, and the forward models outperform their counterparts by 0.87%, 1.63%, and 1.85%, demonstrating the gain of directional sensitivity. Averaged across the three backbones, ghost nodes increase the forward-reverse gap on **River** from 1.7% to 10.9%, and on **Blood** they turn a −3.8% forward deficit into a +1.5% forward advantage. As a consistency check, ghost-node gains nearly vanish in the reverse-flow setting in App. A.5.1. These results show that ghost nodes systematically sharpen GNN sensitivity to edge direction and substantially improve the topology-awareness of standard GNN architectures.

RQ3. *How effectively can the implicit GNN training defined in Eq. (13) learn ghost nodes compared to naïve layer-wise message passing?*

We compare naïve ghost-augmented GNNs against their implicit fixed-point and inverse-operator variants on the **River** and **Blood** benchmarks (Table 4), and evaluate relative MSE reductions for the three GNN backbones. On **River**, the basic implicit solver reduces overall error by 6.9% and 3.5% for $\texttt{gTFP}_{\text{ResGCN}}$ and $\texttt{gTFP}_{\text{GCNII}}$ relative to the naïve ghost baselines, while leaving $\texttt{gTFP}_{\text{ResGAT}}$ unchanged. On **Blood**, it

Table 4: Ablation of inverse-operator variants on River and Blood. Each entry reports the overall MSE (outside parentheses) and the boundary-node MSE (in parentheses). White rows use the directional inverse operator ($\mathbf{A}'^{\dagger}$), and gray rows use the bidirectional inverse operator ($\mathbf{A}'_{\boldsymbol{\Theta}}{}^{\dagger}$).

Inv. Op.	**River**		**Blood**	
	w/o Ghost	**w/ Ghost**	**w/o Ghost**	**w/ Ghost**
Implicit GNN	0.1119 (0.1273)	0.1082 (0.1236)	0.0586 (0.0743)	0.0557 (0.0708)
ResGAT + $\mathbf{A}'^{\dagger}$	0.1122 (0.1333)	0.1057 (0.1233)	0.0450 (0.0632)	0.0440 (0.0615)
ResGAT + $\mathbf{A}'_{\boldsymbol{\Theta}}{}^{\dagger}$	0.1160 (0.1358)	**0.1033 (0.1171)**	0.0451 (0.0627)	**0.0434 (0.0608)**
ResGCN + $\mathbf{A}'^{\dagger}$	0.1194 (0.1339)	0.1151 (0.1310)	0.0533 (0.0732)	0.0524 (0.0716)
ResGCN + $\mathbf{A}'_{\boldsymbol{\Theta}}{}^{\dagger}$	0.1195 (0.1378)	**0.1127 (0.1275)**	0.0531 (0.0728)	**0.0516 (0.0708)**
GCNII + $\mathbf{A}'^{\dagger}$	0.1225 (0.1482)	0.1109 (0.1279)	0.0638 (0.0869)	0.0619 (0.0838)
GCNII + $\mathbf{A}'_{\boldsymbol{\Theta}}{}^{\dagger}$	0.1237 (0.1503)	**0.1040 (0.1203)**	0.0630 (0.0851)	**0.0612 (0.0825)**

yields an additional 12.7% reduction for $\texttt{gTFP}_{\text{GCNII}}$. The stronger inverse-operator learner, with a richer connectivity $\mathbf{A}'_{\boldsymbol{\Theta}}{}^{\dagger}$, further reduces errors by 2.8%, 3.0%, and 7.2% on **River** and by 5.0%, 5.0%, and 4.1% on **Blood** across the three backbones.

Averaged across backbones, the basic implicit solver improves over naïve ghost node method by 3.8%, while the inverse-operator learner increases this improvement to 8.3%. These results show that the implicit GNN training learns ghost nodes more effectively than naïve layer-wise message passing, and lead to additional performance gains with the enriched connectivity $\mathbf{A}'_{\boldsymbol{\Theta}}$.

RQ4. *Can the explicit inverse operator learning reduce the runtime overhead of implicit computation without compromising prediction accuracy?*

Yes. From Table 4, the explicit inverse-operator learner ($\mathbf{A}'_{\boldsymbol{\Theta}}$) reduces ResGAT's MSE from 0.1082 (Implicit GNN w/ Ghost) to 0.1033 (4.5%) while shrinking the runtime ratio from 13.8 to 6.0, i.e., about 2.3× faster; on the GCNII backbone it lowers the error from 0.1082 to 0.1040 (3.9%) with a similar speed-up. On Blood, explicit inverse-operator learner maintains or improves accuracy, achieving relative improvements of 22.0% and 7.4% on ResGAT and ResGCN, respectively, over the Implicit GNN w/ Ghost reference (0.0557). The runtime acceleration is similar to that observed on River. These results indicate that directly learning the inverse operator achieves higher accuracy with significantly lighter computation than iteratively solving fixed-point equations. We further evaluate sparse explicit inverse-operator variants for a better accuracy–efficiency trade-off in App. A.5.3.

RQ5. *Is it better to learn inverse over the parametric adjacency $\mathbf{A}'_{\boldsymbol{\Theta}}$ or its non-parametric $\mathbf{A}'$, why?*

Table 4 shows that, with Ghost nodes enabled, replacing the symmetric adjacency $\mathbf{A}'_{\boldsymbol{\Theta}}$ with its directed counterpart $\mathbf{A}'$ consistently reduces MSE. On **River**, MSE decreases by 2.3%, 2.1%, and 6.2% for ResGAT, ResGCN, and GCNII, respectively. On **Blood**, the same choice yields smaller but still positive gains of 1.36%, 1.53%, and 1.13% for the three backbones.

Although the physical flow graph is strictly downstream-oriented, our bidirectional dense formulation better matches the *global* fixed-point operator implicit GNNs approximate, enabling long-range couplings beyond the observed sparse topology. This agrees with the benefits of dense graph transformations reported by Wang et al. (2025). Physically, both domains are open systems with unobserved exchanges (e.g., rainfall/groundwater/withdrawals in rivers; collateral and micro-circulatory paths in vasculature). Allowing bidirectional edges in the learned adjacency helps absorb these latent inflow–outflow processes, relaxes overly strict conservation biases in the observed graphs, and improves predictive accuracy across datasets.

6 Related Work

Graph Augmentation with Virtual Nodes. Adding virtual nodes to a graph is a known technique to improve expressivity and global information flow (Xu et al., 2019; Ying et al., 2021). A common instantiation is to introduce a single global node connected to all nodes, which has been used to enhance graph-level prediction (Baek et al., 2021), reduce oversquashing by providing shortcut routes for long-range interactions (Hwang et al., 2022), and assist physical simulations where global context or long-range coupling is beneficial (Bentivoglio et al., 2025; Mayr et al., 2023). However, such global augmentation is not tailored to *localized* uncertainty near open boundaries. In our setting, boundary nodes suffer from missing external forcing and their errors propagate downstream under directed transport. We therefore introduce *per-boundary* ghost nodes as boundary-specific proxies, explicitly targeting the external forcing deficit rather than improving global connectivity in a generic manner; the design is motivated by closing local boundary stencils instead of merely increasing model capacity.

Implicit Graph Neural Networks (IGNNs). IGNNs compute node embeddings as a fixed point of a nonlinear system (Gu et al., 2020), a paradigm further generalized by deep equilibrium models (Bai et al., 2021; 2020). This line of work provides a principled way to increase effective depth with parameter sharing, and its behavior has been interpreted through numerical diffusion (Chamberlain et al., 2021) and analyzed using monotone operator theory (Baker et al., 2023), with strategies proposed to mitigate oversmoothing and improve stability (Rusch et al., 2023). Despite these advances, a key limitation remains their reliance on iterative fixed-point solvers and convergence constraints, which can be computationally costly on large graphs. Our implicit formulation inherits the equilibrium perspective, while our explicit inverse-operator learner offers an efficient layer-wise alternative that avoids expensive iteration yet preserves boundary-aware coupling. This yields a solver-free forward pass with predictable cost while still capturing the long-range coupling induced by boundary proxies.

Boundary conditions in PIML. Boundary conditions (BCs) are central in physics-informed machine learning: prior work balances PDE and BC losses via adaptive weights or architectural constraints, and analyzes optimization and generalization failure modes (Raissi et al., 2019; McClenny & Braga-Neto, 2020; Wang et al., 2023; Krishnapriyan et al., 2021). Operator-learning methods such as FNO often assume periodic BCs, whereas graph-based models better accommodate irregular geometries and mixed/complex BCs (Li et al., 2021; Horie & Mitsume, 2022; Li et al., 2024). Other studies infer unknown BCs or latent forcings from data, typically posed as an inverse problem or a variational formulation (Horuz et al., 2022; Zhao et al., 2022; Frerix et al., 2021).

Ghost-cell view and inverse perspective. Classical numerical PDE solvers often introduce *ghost cells/nodes* to close boundary stencils, enabling localized boundary parameterization consistent with the interior discretization (Tseng & Ferziger, 2003). Our approach brings this idea to graph message passing: each boundary node is equipped with a learned ghost proxy, which supplies missing external forcing in a boundary-specific and transport-aware manner. In contrast to approaches that fix BCs or enforce them only through auxiliary losses, our ghost-node formulation *jointly* learns boundary terms together with interior dynamics in a data-driven framework, and integrates them into both implicit and explicit inference (Liu et al., 2025), enabling consistent boundary–interior coupling.

7 Conclusion

This paper revisits the empirical shortcomings of GNNs in topological flux prediction and challenges the prevailing conclusion that GNNs are fundamentally unsuitable for such tasks. By dissecting the prediction loss behavior on fluid networks, we demonstrate that the dominant source of error lies at boundary nodes. To compensate the external forcing deficit in GNN-based flux prediction, we propose a novel `gTFP` framework, which augments GNNs with ghost nodes and an implicit solver to incorporate physically consistent boundary conditions during training. To improve scalability, we devise an explicit solver that learns inverse operators, enabling efficient layer-wise computation. Experiment demonstrates that `gTFP` improves predictive accuracy and reduces the boundary-interior loss gap across multiple standard GNN backbones.

References

Jinheon Baek, Minki Kang, and Sung Ju Hwang. Accurate learning of graph representations with graph multiset pooling. In *International Conference on Learning Representations*, 2021.

Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Multiscale deep equilibrium models. In *Advances in Neural Information Processing Systems*, 2020.

Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Stabilizing equilibrium models by jacobian regularization. In *International Conference on Machine Learning*, 2021.

Justin Baker, Qingsong Wang, Cory D Hauck, and Bao Wang. Implicit graph neural networks: A monotone operator viewpoint. In *International Conference on Machine Learning*, pp. 1521–1548. PMLR, 2023.

I Benemerito, A Melis, Antoine Wehenkel, and A Marzo. openbf: an open-source finite volume 1d blood flow solver. *Physiological Measurement*, 45(12):125002, 2024.

Roberto Bentivoglio, Elvin Isufi, Sebastiaan N. Jonkman, and Riccardo Taormina. Multi-scale hydraulic graph neural networks for flood modelling. *Natural Hazards and Earth System Sciences*, 25:335–351, 2025.

Abraham Berman and Robert J Plemmons. *Nonnegative matrices in the mathematical sciences*. SIAM, 1994.

Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers. In *International Conference on Learning Representations (ICLR)*, 2022.

Christian Busse, Andrew P Kach, and Stephan M Wagner. Boundary conditions: What they are, how to explore them, why we need them, and when to consider them. *Organizational Research Methods*, 20(4): 574–609, 2017.

Benjamin P. Chamberlain, Nick Rowbottom, James Li, and Michael M. Bronstein. Grand: Graph neural diffusion. In *International Conference on Machine Learning (ICML)*, pp. 1407–1419, 2021.

Kaixuan Chen, Shunyu Liu, Tongtian Zhu, Ji Qiao, Yun Su, Yingjie Tian, Tongya Zheng, Haofei Zhang, Zunlei Feng, Jingwen Ye, et al. Improving expressivity of gnns with subgraph-specific factor embedded normalization. In *KDD*, pp. 237–249, 2023.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pp. 1725–1735. PMLR, 2020.

David P Chock. A comparison of numerical methods for solving the advection equation—iii. *Atmospheric Environment. Part A. General Topics*, 25(5-6):853–871, 1991.

Thomas Frerix, Dmitrii Kochkov, Jamie A Smith, Daniel Cremers, Michael P Brenner, and Stephan Hoyer. Variational data assimilation with a learned inverse observation operator. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

W Rockwell Geyer and Parker MacCready. The estuarine circulation. *Annual review of fluid mechanics*, 46 (1):175–197, 2014.

Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. Implicit graph neural networks. In *Advances in neural information processing systems*, volume 33, pp. 11984–11995, 2020.

Masanobu Horie and Naoto Mitsume. Physics-embedded neural networks: Graph neural pde solvers with mixed boundary conditions. In *Advances in Neural Information Processing Systems*, 2022. NeurIPS.

Coşku Can Horuz, Matthias Karlbauer, Timothy Praditia, Martin V Butz, Sergey Oladyshkin, Wolfgang Nowak, and Sebastian Otte. Infering boundary conditions in finite volume neural networks. In *International Conference on Artificial Neural Networks (ICANN)*, 2022.

EunJeong Hwang, Veronika Thost, Shib Sankar Dasgupta, and Tengfei Ma. An analysis of virtual nodes in graph neural networks for link prediction. In *The first learning on graphs conference*, 2022.

Haoyang Jiang, Jindong Wang, Xingquan Zhu, and Yi He. Topology-aware neural flux prediction guided by physics. *ICML*, 2025.

Guangyin Jin, Yuxuan Liang, Yuchen Fang, Zezhi Shao, Jincai Huang, Junbo Zhang, and Yu Zheng. Spatio-temporal graph neural networks for predictive learning in urban computing: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

Olaf Jonkeren, Piet Rietveld, and Jos van Ommeren. Climate change and inland waterway transport: welfare effects of low water levels on the river rhine. *Journal of Transport Economics and Policy (JTEP)*, 41(3): 387–411, 2007.

Nikolas Kirschstein and Yixuan Sun. The merit of river network topology for neural flood forecasting. In *ICML*, 2024.

Christoph Klingler, Karsten Schulz, and Mathew Herrnegger. Lamah| large-sample data for hydrology and environmental sciences for central europe. *Earth System Science Data Discussions*, 2021:1–46, 2021.

Frederik Kratzert, Daniel Klotz, Martin Gauch, Christoph Klingler, Grey Nearing, and Sepp Hochreiter. Large-scale river network modeling using graph neural networks. In *EGU General Assembly Conference Abstracts*, pp. EGU21–13375, 2021.

Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert M. Kirby, and Michael W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. In *Advances in Neural Information Processing Systems*, 2021.

Bernhard Lehner, Gregor Czisch, and Sara Vassolo. The impact of global change on the hydropower potential of europe: a model-based analysis. *Energy policy*, 33(7):839–855, 2005.

Randall J LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.

Tianyu Li, Shufan Zou, Xinghua Chang, Xiaogang Deng, et al. Finite volume graph network (fvgn): Predicting unsteady incompressible fluid dynamics via boundary-aware message aggregation. *Physics of Fluids*, 2024.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew L. Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

Zewen Liu, Xiaoda Wang, Bohan Wang, Zijie Huang, Carl Yang, and Wei Jin. Graph odes and beyond: A comprehensive survey on integrating differential equations with graph neural networks. *arXiv preprint arXiv:2503.23167*, 2025.

Donald C Mangold and Chin-Fu Tsang. A summary of subsurface hydrological and hydrochemical models. *Reviews of Geophysics*, 29(1):51–79, 1991.

Andreas Mayr, Sebastian Lehner, Arno Mayrhofer, Christoph Kloss, Sepp Hochreiter, and Johannes Brandstetter. Boundary graph neural networks for 3d simulations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 9099–9107, 2023.

Liam McClenny and Ulisses Braga-Neto. Self-adaptive physics-informed neural networks using a soft attention mechanism. In *AAAISpring Symposium on Combining Learning and Reasoning*, 2020.

Claudio Paniconi and Mario Putti. A comparison of picard and newton iteration in the numerical solution of multidimensional variably saturated flow problems. *Water Resources Research*, 30(12):3357–3374, 1994.

N LeRoy Poff, J David Allan, Mark B Bain, James R Karr, Karen L Prestegaard, Brian D Richter, Richard E Sparks, and Julie C Stromberg. The natural flow regime. *BioScience*, 47(11):769–784, 1997.

Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, 2019.

K Manjunatha Prasad and RB Bapat. The generalized moore-penrose inverse. *Linear Algebra and its Applications*, 165:59–69, 1992.

Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear pdes. *Journal of Computational Physics*, 378:686–707, 2019.

T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

Kathrin Schacke. On the kronecker product. *Master's thesis, University of Waterloo*, 2004.

Yu-Heng Tseng and Joel H Ferziger. A ghost-cell immersed boundary method for flow in complex geometry. *Journal of computational physics*, 192(2):593–623, 2003.

Anna Varbella, Kenza Amara, Blazhe Gjorgiev, Mennatallah El-Assady, and Giovanni Sansavini. Powergraph: A power grid benchmark dataset for graph neural networks. *Advances in Neural Information Processing Systems*, 37:110784–110804, 2024.

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

Zvonimir Vrselja, Hrvoje Brkic, Stefan Mrdenovic, Radivoje Radic, and Goran Curic. Function of circle of willis. *Journal of Cerebral Blood Flow & Metabolism*, 34(4):578–584, 2014.

Hongjun Wang, Jiyuan Chen, Yinqiang Zheng, and Xuan Song. Accelerating flood warnings by 10 hours: the power of river network topology in ai-enhanced flood forecasting. *npj Natural Hazards*, 2(1):45, 2025.

Jian Wang, Yifan Mo, Bashar Izzuddin, and Chang-Won Kim. Exact dirichlet boundary physics-informed neural network (epinn) for solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 414:116184, 2023.

Paul G Whitehead, Robert L Wilby, Richard W Battarbee, Martin Kernan, and Andrew John Wade. A review of the potential impacts of climate change on surface water quality. *Hydrological sciences journal*, 54(1):101–123, 2009.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in neural information processing systems*, 34:28877–28888, 2021.

Qingqing Zhao, David B Lindell, and Gordon Wetzstein. Learning to solve pde-constrained inverse problems with graph networks. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, 2022.

Baodong Zheng and Liancheng Wang. Spectral radius and infinity norm of matrices. *Journal of mathematical analysis and applications*, 346(1):243–250, 2008.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

A Appendix

A.1 Different Boundary Conditions

To connect our ghost-node parameterization in Eq. (5) to classical boundary conditions, we instantiate three canonical types on the river graph, namely, Dirichlet, Neumann, and Robin.

Dirichlet ($u = g_a$). Yields $u_g = u_b = g_a$. On the graph this degenerates to $h_g^{(l)} = h_b^{(l)}$, i.e., a *fixed* boundary input. This choice does not match the time-varying inflow/pressure in our datasets and therefore performs worst.

Neumann ($\partial_x u = g_a$). A first-order discretization gives $u_g = u_b - g_a\,\Delta x$. On the graph, one can use a learned form $h_g^{(l)} = \mathrm{MLP}(h_b^{(l)}; \theta)$, where the effect of g_a is implicitly encoded into the parameters θ.

Robin. In Eq. (5), $h_g^{(l)} = \omega_1\, h_b^{(l)} + \omega_2\, \frac{h_b^{(l)} - h_{\mathrm{nbr}}^{(l)}}{\Delta x}$, which jointly uses the boundary value and the boundary–downstream difference. We consider two options: (i) a linear two-parameter variant that learns only ω_1, ω_2; and (ii) the *MLP* variant used in the main paper. Our sensitivity study over MLP depth/width (Table 10) shows low sensitivity.

Model	Boundary MSE	Interior MSE	Overall MSE
ResGCN	0.1408	0.0985	0.1232
Dirichlet	0.1415	0.0990	0.1238
Neumann	0.1406	0.0983	0.1230
Robin (2-param)	0.1374	0.0974	0.1213
Robin (MLP)	0.1313	0.0950	0.1162

Table 5: Ablation on boundary conditions for ResGCN on the river network.

To connect our ghost-node parameterization in Eq. (5) to classical boundary conditions, we design experiments on three canonical types in the river graph, namely, Dirichlet, Neumann, and Robin. As summarized in Table 5, relative to the vanilla forward ResGCN, Dirichlet-Ghost slightly increases overall MSE by 0.5%, while Neumann-Ghost yields a negligible 0.2% decrease. The simplified two-parameter Robin-Ghost reduces boundary and overall MSE by 2.4% and 1.5%, respectively, and the MLP Robin-Ghost achieves the largest gains, lowering boundary and overall MSE by 6.7% and 5.7%. These quantitative trends support our choice of a Robin-style, learnable ghost mapping as the default.

A.2 Derivation for $\mathbf{A}'_{\boldsymbol{\Theta}}$

Notation (Symbol Table). To keep the appendix consistent with the main text, we list the symbols used below. Only the Robin coefficients have been renamed from (α, β) to (w_1, w_2); the advection speed a is unchanged, and the spatial step is uniformly denoted by Δx.

- $u(x,t)$: scalar state; u_i^n is the discrete state at grid index i and time level n.

- $a > 0$: advection speed in the governing PDE (kept as is).
- Δx: spatial grid spacing; Δt: time step.
- $\sigma := \dfrac{a\,\Delta t}{\Delta x}$: Courant number for advection.
- w_1, w_2: Robin boundary coefficients used only in the boundary condition.
- u_L: prescribed boundary trace entering the Robin condition; when needed we use u_L^{n+1} to indicate the time level.
- $\mathbf{A}'_{\boldsymbol{\Theta}}$: implicit system matrix assembled from interior and boundary discrete equations at time level $n+1$.

This section provides a detailed derivation for the components of the implicit system matrix $\mathbf{A}'_{\boldsymbol{\Theta}}$, establishing the theoretical foundation for the `gTFP` framework discussed in the main paper. By using the 1D advection equation as a canonical example, this derivation serves to:

- Justify the interpretation of GNN message-passing as a numerical discretization of a physical system's spatial dynamics.
- Demonstrate how boundary conditions introduce specific mathematical constraints that are absent in standard GNN formulations.
- Show how these discrete equations naturally form the $\mathbf{A}'_{\boldsymbol{\Theta}}$, which is the core problem our implicit solver addresses.

The derivations are based on two fundamental principles:

- The Governing PDE: The 1D advection equation, $\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0$, where $a > 0$.
- The Boundary Condition: Robin-type condition at the boundary $x = 0$, given by $w_1 u + w_2 \frac{\partial u}{\partial x} = u_L$.

A.2.1 Derivation of the Interior Equations

Here is the derivation of the interior equations:

Target Equation:

$$(1+\sigma)u_i^{n+1} - \sigma u_{i-1}^{n+1} = u_i^n \quad \text{for } i = 1, \ldots, N$$

Method: Apply the standard implicit first-order upwind scheme to the governing PDE at interior node i.

1. **Discretize the Time Derivative:** We approximate the partial derivative with respect to time, $\frac{\partial u}{\partial t}$, using a first-order forward difference:

$$\frac{\partial u}{\partial t} \approx \frac{u_i^{n+1} - u_i^n}{\Delta t}$$

2. **Discretize the Spatial Derivative (Implicitly):** The term ***implicit*** signifies that the spatial derivative is evaluated at the future time step, $n+1$. The term ***upwind*** (for $a > 0$) means we use a backward difference, looking at the node from which the flow originates $(i-1)$.

$$\frac{\partial u}{\partial x} \approx \frac{u_i^{n+1} - u_{i-1}^{n+1}}{\Delta x}$$

3. **Combine and Simplify:** Substitute the approximation back into the governing PDE, $\frac{\partial u}{\partial t}+a\frac{\partial u}{\partial x}=0$:

$$\frac{u_i^{n+1}-u_i^n}{\Delta t}+a\left(\frac{u_i^{n+1}-u_{i-1}^{n+1}}{\Delta x}\right)=0$$

Multiply the entire equation by Δt to clear the denominator:

$$(u_i^{n+1}-u_i^n)+\frac{a\Delta t}{\Delta x}(u_i^{n+1}-u_{i-1}^{n+1})=0$$

Let $\sigma=\frac{a\Delta t}{\Delta x}$ be the Courant number. Substituting σ gives:

$$u_i^{n+1}-u_i^n+\sigma(u_i^{n+1}-u_{i-1}^{n+1})=0$$

4. **Rearrange:** Group all the unknown terms (at time $n+1$) on the left side and the known terms (at time n) on the right side.

$$u_i^{n+1}+\sigma u_i^{n+1}-\sigma u_{i-1}^{n+1}=u_i^n$$

Factoring out u_i^{n+1} yields the final target equation:

$$(1+\sigma)u_i^{n+1}-\sigma u_{i-1}^{n+1}=u_i^n$$

A.2.2 Derivation of the Boundary Condition Equation

Here is the Derivation of the Boundary Condition Equation:

Target Equation:

$$\left(w_1-\frac{w_2}{\Delta x}\right)u_0^{n+1}+\left(\frac{w_2}{\Delta x}\right)u_1^{n+1}-u_L^{n+1}=0$$

Method: This equation arises directly from discretizing the Robin boundary condition itself, without involving the PDE.

1. **State the Boundary Condition:** The Robin condition at node $i=0$ and at the future time step $n+1$ is:

$$w_1\,u_0^{n+1}+w_2\left(\frac{\partial u}{\partial x}\right)\bigg|_{i=0}^{n+1}=u_L^{n+1}$$

2. **Discretize the Spatial Derivative:** At the boundary $i=0$, we cannot use a backward difference. The natural choice is a first-order forward difference, using nodes 0 and 1:

$$\left(\frac{\partial u}{\partial x}\right)\bigg|_{i=0}^{n+1}\approx\frac{u_1^{n+1}-u_0^{n+1}}{\Delta x}$$

3. **Combine and Rearrange:** Substitute the discretized derivative back into the boundary condition equation:

$$w_1\,u_0^{n+1}+w_2\left(\frac{u_1^{n+1}-u_0^{n+1}}{\Delta x}\right)=u_L^{n+1}$$

Distribute the term $w_2/\Delta x$:

$$w_1\,u_0^{n+1}+\frac{w_2}{\Delta x}u_1^{n+1}-\frac{w_2}{\Delta x}u_0^{n+1}=u_L^{n+1}$$

Group the coefficients for u_0^{n+1} and move all terms to the left side to obtain the final form:

$$\left(w_1-\frac{w_2}{\Delta x}\right)u_0^{n+1}+\left(\frac{w_2}{\Delta x}\right)u_1^{n+1}-u_L^{n+1}=0$$

A.2.3 Derivation of the PDE at the Boundary

Here is the Derivation of the PDE at the Boundary:

Target Equation:

$$\Big(1 - \frac{a\,\Delta t\,w_1}{w_2}\Big)u_0^{n+1} + \Big(\frac{a\,\Delta t}{w_2}\Big)u_L^{n+1} = u_0^n$$

Method: This derivation cleverly combines the PDE and the BC. The key is to use the boundary condition to eliminate the spatial derivative term from the discretized PDE.

1. **Discretize the PDE at the Boundary ($i = 0$):** First, write the implicit discretization of the PDE at node $i = 0$:

$$\frac{u_0^{n+1} - u_0^n}{\Delta t} + a\left(\frac{\partial u}{\partial x}\right)\Bigg|_{i=0}^{n+1} = 0$$

   This equation contains the spatial derivative term, which we need to handle.

2. **Isolate the Derivative from the Boundary Condition:** Return to the Robin condition from the previous section:

$$w_1\,u_0^{n+1} + w_2\left(\frac{\partial u}{\partial x}\right)\Bigg|_{i=0}^{n+1} = u_L^{n+1}.$$

   We can rearrange this to solve for the derivative term:

$$w_2\left(\frac{\partial u}{\partial x}\right)\Bigg|_{i=0}^{n+1} = u_L^{n+1} - w_1\,u_0^{n+1}$$

$$\left(\frac{\partial u}{\partial x}\right)\Bigg|_{i=0}^{n+1} = \frac{u_L^{n+1} - w_1\,u_0^{n+1}}{w_2}$$

3. **Substitute and Simplify:** Now, substitute the expression for the derivative from Step 2 into the discretized PDE from Step 1:

$$\frac{u_0^{n+1} - u_0^n}{\Delta t} + a\left(\frac{u_L^{n+1} - w_1\,u_0^{n+1}}{w_2}\right) = 0$$

   Multiply the entire equation by Δt:

$$(u_0^{n+1} - u_0^n) + \frac{a\Delta t}{w_2}(u_L^{n+1} - w_1\,u_0^{n+1}) = 0$$

   Distribute the term $\frac{a\Delta t}{w_2}$:

$$u_0^{n+1} - u_0^n + \frac{a\Delta t}{w_2}u_L^{n+1} - \frac{a\Delta t\,w_1}{w_2}u_0^{n+1} = 0$$

4. **Rearrange:** Finally, group the unknown terms $(n+1)$ on the left side and the known term (n) on the right side.

$$\Big(1 - \frac{a\Delta t\,w_1}{w_2}\Big)u_0^{n+1} + \Big(\frac{a\Delta t}{w_2}\Big)u_L^{n+1} = u_0^n$$

In summary, the equations derived for the interior nodes (from the PDE) and the boundary nodes (from the boundary condition) collectively define the rows of the augmented system matrix $\mathbf{A}'_{\boldsymbol{\Theta}}$. The core contribution of the `gTFP` framework lies in its ability to learn an efficient, implicit operator.

A.3 Illustrating the Augmented Matrix

The core of our implicit solver revolves around constructing the system $\mathbf{A}'_{\boldsymbol{\Theta}}$. The structure of the system matrix $\mathbf{A}'_{\boldsymbol{\Theta}}$ is a direct reflection of the underlying physical topology. This section provides concrete examples to illustrate this crucial connection. We will demonstrate the structure of $\mathbf{A}'_{\boldsymbol{\Theta}}$ in complex, coupled ones.

Now, we alter the topology by introducing a new, seventh node, v_m, into which both systems merge. The flow paths become: $v_{b1} \to v_{d1} \to v_m$ and $v_{b2} \to v_{d2} \to v_m$.

This introduces physical coupling, as the state of the merge point v_m now depends on inputs from both upstream branches. We augment our state vector with the these node:

$$\mathbf{H}'^{(\mathbf{t+1})} = \begin{bmatrix} \mathbf{h}_{g1}^{(t+1)}, & \mathbf{h}_{b1}^{(t+1)}, & \mathbf{h}_{d1}^{(t+1)}, & \\ \mathbf{h}_{g2}^{(t+1)}, & \mathbf{h}_{b2}^{(t+1)}, & \mathbf{h}_{d2}^{(t+1)}, & \mathbf{h}_{m}^{(t+1)} \end{bmatrix}^T$$

The new 7×7 matrix, $\mathbf{A}'_{\boldsymbol{\Theta}}$, is no longer block-diagonal. The coupling appears precisely in the equation governing the new node v_m.

$$\mathbf{A}'_{\boldsymbol{\Theta}} = \left(\begin{array}{ccc|ccc|c} 1 & -p_{1b} & -p_{1d} & 0 & 0 & 0 & 0 \\ -w_{b1,g1} & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -w_{d1,b1} & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & -p_{2b} & -p_{2d} & 0 \\ 0 & 0 & 0 & -w_{b2,g2} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -w_{d2,b2} & 1 & 0 \\ \hline 0 & 0 & -w_{m,d1} & 0 & 0 & -w_{m,d2} & \mathrm{I} \end{array}\right)$$

Analysis of the Coupling: The first six rows and columns largely retain the decoupled structure. The critical change is in the **last row**, which defines the update for v_m. The non-zero elements in columns 3 and 6, $-w_{m,d1}$ and $-w_{m,d2}$, are the mathematical signature of the physical merge. These elements, which reside in the previously zero off-diagonal block area, now link the two subsystems together through the dynamics of v_m. This clearly demonstrates how a local change in graph topology induces a specific, predictable change in the global system matrix.

A.4 From Closed Subgraphs to Open systems

Problem overview. The 14 evaluation nodes originate from a larger, 30-node system. Training and evaluating them as an isolated subgraph implicitly imposes a closed-system inductive bias: only intra-graph interactions are modeled, while upstream/downstream exchanges and external forcings are omitted. This bias disproportionately affects Boundary nodes, where missing cross-boundary inputs manifest as inflated errors. To mitigate this, we introduce ghost nodes that learn a data-driven proxy for the absent external coupling at the boundary, effectively "half-opening" the subgraph.

Discussion. Table 6 shows that when the same 14 nodes are trained and evaluated as a closed subgraph, the boundary penalty relative to the overall average is very high (+82.6%), indicating that the omission of external interactions disproportionately affects boundary behavior. Evaluating the identical nodes inside the full graph substantially reduces this penalty to +10.6%, reflecting the richer upstream/downstream context and constraints available in the larger system. When only the subgraph is available, adding ghost nodes still narrows the gap by learning a data-driven ghost node proxy: the penalty drops from +82.6% to +70.4%, while the overall Avg also improves. Altogether, these results support the view that closed subgraphs amplify boundary difficulty, embedding the subsystem in the larger graph naturally balances errors, and ghost nodes provide a lightweight way to approximate external interactions when access to external nodes is not feasible.

A.5 Additional Experiments

We report four additional ablations that complement the main results: (i) forward vs. reverse settings of ghost nodes, (ii) physics-aware GNN backbones, (iii) sparse inverse operators, and (iv) sensitivity to the ghost MLP θ_{gh}.

Table 6: Results for the same 14 nodes under three settings (Subgraph / Fullgraph / Ghost-Subgraph). $\%\Delta_{\text{Boundary}}$ denotes the boundary increase relative to Avg, with smaller values indicating a weaker boundary penalty.

Setting	**Avg**	**Interior**	**Boundary**	$\%\boldsymbol{\Delta}_{\text{Boundary}}$
Subgraph	0.0530	0.0355	0.0968	+82.6%
Fullgraph	0.0930	0.0891	0.1029	+10.6%
Ghost-Subgraph	0.0476	0.0342	0.0811	+70.4%

A.5.1 Reverse-Flow Setting of Ghost Nodes (Ref to RQ2.)

Our central claim concerns the physically correct *forward* setting, where upstream boundary nodes lack incoming information and thus suffer from an external forcing deficit that ghost nodes are designed to compensate. In the *reverse* setting, the adjacency is flipped, so these nodes already receive inputs from their downstream neighbors; in this regime a ghost node no longer corresponds to a meaningful physical boundary condition. We therefore treat reverse experiments only as a consistency check. As shown in Table 7, on the forward graph adding ghost nodes to ResGCN reduces boundary and overall MSE by 6.7% and 5.7%, respectively, whereas on the reverse graph it slightly increases boundary and overall MSE by 0.8% and 1.0%. This quantitative asymmetry supports our interpretation that ghost nodes specifically repair the external forcing deficit rather than acting as a generic over-parameterization.

Model	Boundary MSE	Interior MSE	Overall MSE
ResGCN (Fwd)	0.1408	0.0985	0.1232
ResGCN (Rev)	0.1369	0.1071	0.1245
$\texttt{gTFP}_{\text{ResGCN}}$ (Fwd)	0.1313	0.0950	0.1162
$\texttt{gTFP}_{\text{ResGCN}}$ (Rev)	0.1380	0.1084	0.1257

Table 7: Forward-versus-reverse comparison of ResGCN with and without ghost nodes on the river network. Ghost nodes substantially reduce boundary and overall MSE only in the physically meaningful forward setting, and slightly degrade performance when the flow direction is reversed.

A.5.2 Physics-Aware GNN Backbones (Ref to RQ1.)

We further ask whether using physics-aware backbones alone can resolve the external forcing deficit. To this end, we add two strong baselines: a message-passing PDE solvers (**MP PDE**) and a Graph Neural Operator (**GNO**). As shown in Table 8, all physics-aware models improve over vanilla ResGCN in overall MSE: MP PDE and GNO reduce overall error by 4.5% and 0.7%, respectively, whereas our ghost-based $\texttt{gTFP}_{\text{ResGCN}}$ and Explicit-Inverse Operator (ResGCN) achieve larger reductions of 5.7% and 8.5%.

However, their boundary nodes remain substantially less accurate than interior nodes. Relative to vanilla ResGCN, MP PDE and GNO reduce boundary MSE by only 5.4% and 0.9% and shrink the boundary–interior gap by just 8.8% and 2.1%, respectively, while $\texttt{gTFP}_{\text{ResGCN}}$ and Explicit-Inverse Operator (ResGCN) reduce boundary MSE by 6.7% and 9.4% and reduce the boundary–interior gap by 11.0% and 9.8%, respectively. These quantitative trends support our claim that the main bottleneck lies in the missing external forcing, rather than the choice of GNN architecture, and that our ghost module acts as a plug-and-play component that effectively complements physics-aware backbones.

Model	Boundary MSE	Interior MSE	*Diff. (%)*	Overall MSE
ResGCN	0.1408	0.0985	42.9	0.1232
MP PDE Solver	0.1332	0.0957	39.2	0.1176
GNO	0.1395	0.0982	42.1	0.1223
$\texttt{gTFP}_{\text{ResGCN}}$	0.1313	0.0950	38.2	0.1162
ResGCN + $\mathbf{A}'_{\boldsymbol{\Theta}}{}^{\dagger}$	0.1275	0.0919	38.7	0.1127

Table 8: Comparison of ResGCN (Fwd), MP PDE solver, GNO, $\texttt{gTFP}_{\text{ResGCN}}$, and Inverse Operator (ResGCN) on the river network. All baselines improve over vanilla ResGCN, while $\texttt{gTFP}_{\text{ResGCN}}$ and Inverse Operator (ResGCN) yield the lowest overall errors, especially on the boundary. *Diff. (%)* is the relative difference between boundary and interior MSE.

A.5.3 Sparse Inverse Operator (Ref to RQ3. and RQ4.)

To reduce the $\mathcal{O}(|V|^2 d)$ cost of a fully dense inverse while preserving the global coupling of the explicit solver, we parameterize the inverse $(A'_{\Theta})^{\dagger}$ in a low-rank way:

$$(A'_{\Theta})^{\dagger} \;\approx\; S \;+\; UV^{\top},$$

where S inherits the sparsity pattern of the original graph, and $U, V \in \mathbb{R}^{|V|\times r}$ with $r \ll |V|$ form a low-rank correction. For a feature matrix $X \in \mathbb{R}^{|V|\times d}$,

$$(S + UV^{\top})X \;=\; SX \;+\; U\big(V^{\top}X\big),$$

which costs $\mathcal{O}(|E|d + 2|V|rd)$ time and stores $\approx |E| + 2|V|r$ parameters, remaining *near-linear* in $|E|$ without ever materializing a dense $|V|\times|V|$ matrix. We keep the inverse-consistency regularizer

$$\mathcal{L}_{\text{reg}} = \lambda \big\| (A'_{\Theta})^{\dagger} A'_{\Theta} - I \big\|_F^2,$$

and a sweep over $\lambda \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$ yields at most 2% variation in MSE, indicating low sensitivity.

Table 9 compares dense and sparse inverse variants. Relative to vanilla ResGCN, the dense explicit inverse reduces boundary and overall MSE by 9.4% and 8.5%, respectively, but incurs a $\times 6.0$ runtime overhead. The sparse inverse operators with ranks r=8 and r=32 further reduce boundary MSE by 21.9% and 22.9% and overall MSE by 25.0% and 25.6%, respectively, while requiring only $\times 2.8$ and $\times 3.1$ runtime. These results show that the sparse, low-rank inverse operator attains the best boundary and overall accuracy with substantially lower cost than the dense explicit operator, offering a favorable accuracy and efficiency trade-off.

Model	Boundary MSE	Interior MSE	Overall MSE	Runtime (rel.)
ResGCN	0.1408	0.0985	0.1232	$\times 1.0$
$\texttt{gTFP}_{\text{ResGCN}}$	0.1313	0.0950	0.1162	$\times 2.3$
ResGCN + $\mathbf{A}'_{\boldsymbol{\Theta}}{}^{\dagger}$	0.1275	0.0919	0.1127	$\times 6.0$
Implicit GNN	0.1236	0.0866	0.1082	$\times 13.8$
Sparse Inv. Op. (r=8)	0.1099	0.0678	0.0924	$\times 2.8$
Sparse Inv. Op. (r=32)	0.1086	0.0677	0.0916	$\times 3.1$
Sparse Inv. Op. (r=64)	0.1087	0.0684	0.0919	$\times 3.3$

Table 9: Boundary vs. interior vs. overall MSE on the river network, together with relative runtime. The sparse inverse operator attains the best accuracy with only moderate cost, closing most of the boundary gap while remaining near-linear in the number of edges.

A.5.4 Sensitivity to the Ghost MLP θ_{gh}

We examine how sensitive the ghost-node performance is to the capacity of the MLP (ghost node proxy). We vary the depth L and width H of the MLP and report boundary, interior, and average MSE. As shown

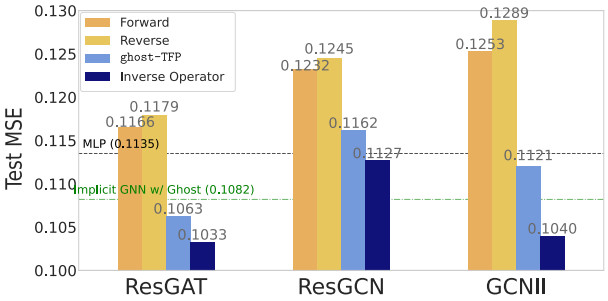
in Table 10, all four configurations yield very similar errors: relative to the default setting (L=1, H=128), boundary MSE varies by at most 0.6%, interior MSE by at most 0.7%, and overall MSE by at most 0.6%. This indicates that our method does not rely on a carefully tuned ghost MLP and is robust to the particular choice of θ_{gh}.

Depth (L)	Width (H)	Boundary MSE	Interior MSE	Overall MSE (Avg)
1	64	0.1321	0.0956	0.1169
1	**128**	**0.1313**	**0.0950**	**0.1162**
2	64	0.1306	0.0946	0.1157
2	128	0.1310	0.0948	0.1160

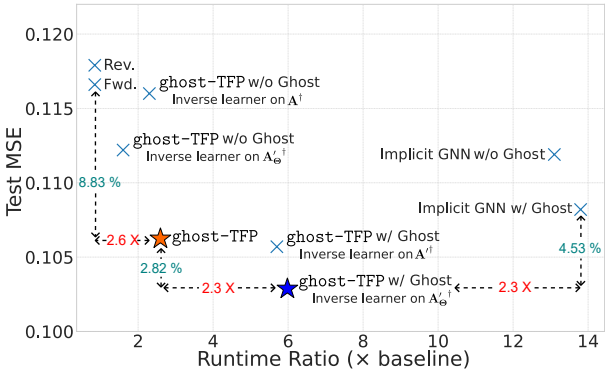
Table 10: Sensitivity of ghost ResGCN (MLP) to the depth and width of the ghost mapping MLP on the river network. All configurations achieve nearly identical boundary and interior errors, indicating low sensitivity to θ_{gh}.

A.6 Supplementary Figures

These figures complement the main results by contrasting the message flow topology and efficiency. The bar chart compares Forward and Reverse variants across backbones, showing small gaps between the two, while `gTFP` and the inverse learner consistently reduce test MSE. The scatter plot relates test MSE to runtime, where points closer to the lower-left indicate better accuracy–efficiency trade-offs; `gTFP` occupies this region.



(a) Forward vs. Reverse GNNs (left two bars) highlight the similarity from different message flow direction, while `gTFP` and our inverse learner (right two bars) drive errors lower and widen the gap with reverse flow.



(b) Each point shows model's test MSE against its runtime ratio relative to the ResGAT Forward baseline (1×). Points that fall lower and further left deliver both lower error and faster inference, highlighting superior accuracy–efficiency trade-offs.

A.7 Algorithms

Here are the 3 algorithms for the `gTFP` Approach.

Algorithm A.7.1: Ghost-TFP with Standard GNN Backbone

Initialize : Graph $G = (V, E)$ and its node features H, Set of boundary nodes V_{BN},
Number of GNN layers L, Upstream neighbor set of node v_j is $\mathcal{N}(j)$,
Graph augmentation operator $\mathcal{A}$, Ground-truth labels y,
Model components: GNN backbone f_{GNN}, Predictor MLP f_{pred}, Ghost MLP,
Loss function $\mathcal{L}$ and Projection operator Π_A.

Parameters: Learnable parameters θ_{gb} for the Ghost MLP,
θ_{GNN} for the GNN backbone, and θ_{pred} for the predictor.

$V_{GH}, H_{GH}, E_{GH} \leftarrow \emptyset, \emptyset, \emptyset$;
for *each boundary node* $v_b \in V_{BN}$ **do**
  $v_{nbr} \leftarrow v_{nbr} \in \{v_j \mid v_b \in \mathcal{N}(j)\}$; `// Get interior neighbor`
  $h_b, h_{nbr} \leftarrow H[v_b, :], H[v_{nbr}, :]$;
  $h_g \leftarrow \text{MLP}(\text{Concat}(h_b, h_{nbr}); \theta_{gb})$; `// Learn ghost embedding`
  Create v_g and add to V_{GH}; Append h_g to H_{GH};
  Add edge (v_g, v_b) to E_{GH}
$H' \leftarrow \text{Concat}(H, H_{GH})$; `// Augmented features`
$A' \leftarrow \mathcal{A}(V \cup V_{GH}, E \cup E_{GH})$; `// Augmented adjacency`
$H'^{(L)} \leftarrow f_{\text{GNN}}(A', H'; \theta_{\text{GNN}})$
$H^{(L)} \leftarrow \Pi_A(H'^{(L)})$; `// Project back to original nodes`
$\hat{y} \leftarrow f_{\text{pred}}(H^{(L)}; \theta_{\text{pred}})$
$\mathcal{L}_{loss} \leftarrow \mathcal{L}(y, \hat{y})$;
Update $\theta_{gb}, \theta_{\text{GNN}}, \theta_{\text{pred}}$ using gradients of $\mathcal{L}_{loss}$;
return *Prediction* $\hat{y}$

Algorithm A.7.2: Implicit Ghost-Boundary Message-Passing

Initialize : Augmented graph $G' = (A', H')$, Ground truth y,
Predictor model f_{pred}, Loss function $\mathcal{L}$.

Parameters: Shared weight matrix Θ with constraint $||\Theta||_\infty \leq 1/\lambda_{pf}(A')$,
Learnable parameters θ_{pred} for the predictor.
Max iterations K, Convergence tolerance $\epsilon > 0$.

$H'^{(0)} \leftarrow H'$;
for $k = 1$ ***to*** K **do**
  $H'^{(k)} \leftarrow \sigma(A' H'^{(k-1)} \Theta)$;
  if $||H'^{(k)} - H'^{(k-1)}|| < \epsilon$ **then**
    break;
$H'^{*} \leftarrow H'^{(k)}$; `// Equilibrium embeddings satisfying Eq. (9)`
$H^{*} \leftarrow \Pi_A(H'^{*})$; `// Project back to original node set`
$\hat{y} \leftarrow f_{\text{pred}}(H^{*}; \theta_{pred})$;
$\mathcal{L}_{loss} \leftarrow \mathcal{L}(y, \hat{y})$;
$\nabla_{H'^{*}} \mathcal{L}_{loss} \leftarrow$ Compute gradient from loss up to the equilibrium point;
$g^{(0)} \leftarrow \mathbf{0}$; `// Initialize implicit gradient`
$D \leftarrow \sigma'(A' H'^{*} \Theta)$; `// Jacobian of activation at equilibrium`
for $k = 1$ ***to*** K **do**
  $g^{(k)} \leftarrow D \odot ((A')^T g^{(k-1)} \Theta^T + \nabla_{H'^{*}} \mathcal{L}_{loss})$;
  if $||g^{(k)} - g^{(k-1)}|| < \epsilon$ **then**
    break;
$g^{*} \leftarrow g^{(k)}$; `// Converged implicit gradient` $\nabla_Z l$
Compute $\nabla_\Theta \mathcal{L}_{loss}$ and $\nabla_{\theta_{gh}} \mathcal{L}_{loss}$ using g^{*} via auto-differentiation;
Update parameters $\Theta, \theta_{gh}, \theta_{pred}$;
return *Prediction* $\hat{y}$

Algorithm A.7.3: Explicit Adjacency-Inverse Solver

Initialize : Augmented graph $G' = (V', E')$, Initial augmented features H',
Ground truth y, Loss function $\mathcal{L}$, Projection operator Π_A.
Parameters: Number of layers L, Regularization strength λ,
Learnable parameters θ_Ψ for Inverse operator Ψ ,
Parametric adjacency θ_A for A'_Θ , Predictor θ_{pred} for f_{pred} .

Construct learnable parametric adjacency ; `// upstream ↔ all downstream`
A'_Θ using parameters θ_A;
$(A'_\Theta)^\dagger \leftarrow \Psi(A'_\Theta; \theta_\Psi)$; `// Approximate inverse with learned operator`
$H'^{(0)} \leftarrow H'$;
for $l = 0$ ***to*** $L - 1$ **do**
  $H'^{(l+1)} \leftarrow \sigma((A'_\Theta)^\dagger H'^{(l)})$; `// Apply GNN layer with non-linearity`
$H^{(L)} \leftarrow \Pi_A(H'^{(L)})$; `// Project back to original node set`
$\hat{y} \leftarrow f_{\text{pred}}(H^{(L)}; \theta_{pred})$;
$\mathcal{L}_{\text{node}} \leftarrow \mathcal{L}(y, \hat{y})$; `// Node prediction loss`
$\mathcal{L}_{\text{reg}} \leftarrow \lambda ||(A'_\Theta)^\dagger A'_\Theta - I||_F^2$; `// Inverse constraint regularization`
$\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{node}} + \mathcal{L}_{\text{reg}}$;
Update $\theta_\Psi, \theta_A, \theta_{pred}$ using gradients of $\mathcal{L}_{\text{total}}$;
return *Prediction* $\hat{y}$