RECASTNET: ADAPTING NEURAL SOLVERS TO NETWORK TOPOLOGY CHANGES WITHOUT RETRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Many scientific and engineering applications require repeatedly solving large-scale networked systems under varying operating conditions and topologies, with local component equations coupled through the network. Neural solvers can accelerate these computations, but training them to cover the combinatorially many possible topologies is costly and often impractical. We present **Neural Network Recasting (RecastNet)**, which shifts topology adaptation from training to inference: a neural solver is trained only for solution prediction on a single base topology and analytically recast under unseen topology changes. Our key finding is that the input–output Jacobian of the base-topology neural solver can recover the underlying network Laplacian inverse, allowing low-rank topology changes to be incorporated analytically. This yields zero-shot adaptation to unseen topology changes with only lightweight test-time computation. Experiments on synthetic networks and real-world electric grids show that RecastNet consistently outperforms graph-learning and topology-adaptation baselines under topology changes of increasing severity. Overall, RecastNet offers a new paradigm for topology adaptation through analytical recasting rather than cross-topology training.

1 INTRODUCTION

Solving large systems of linear or nonlinear equations is a fundamental task across scientific and engineering applications. A broad and important class of these problems has a *networked structure*: each node or component is governed by its local equations, while network connections couple these equations into a global system (Strogatz, 2001; Olfati-Saber et al., 2007; Van der Schaft & Maschke, 2013). Such structure is ubiquitous in electronic circuits (Ho et al., 1975), electric power systems (Dorfler & Bullo, 2012), fluid networks (Srinivasan et al., 2024), transportation networks (Patriksson, 2015), and other networked infrastructures. In practice, these equations are rarely solved only once: solutions are repeatedly required under changing operating conditions and topology configurations, as routinely encountered in contingency screening (Tasseff et al., 2022; Sun, 2024), reliability assessment (Zhu et al., 2020; Dini et al., 2022), and decision-making (Fan et al., 2024; Brunel et al., 2025). For classical numerical solvers, each new instance requires solving the large-scale system anew, making repeated evaluation computationally expensive at scale (Hindmarsh et al., 2005; Gardner et al., 2022).

Neural solvers offer an appealing alternative to these repetitive numerical solves: once trained, they can efficiently predict solutions for new instances through neural network inference (Gulraiz et al., 2025; Muhammad et al., 2025). Existing neural solvers range from parameter-to-solution mappings learned with conventional architectures such as MLPs and ResNets (He et al., 2016; Schotthöfer et al., 2022; Liu et al., 2025) to solution operators learned between function spaces (Liu & Tang, 2025; Berner et al., 2026; Lee et al., 2026). A persistent challenge, however, is *generalization across network topologies*. Structural shifts can substantially degrade learned graph representations and predictions (Liu et al., 2023b; Bao et al., 2025). Prior work has tackled this challenge with graph neural networks (Sanchez-Gonzalez et al., 2020; Pfaff et al., 2021; Dan et al., 2026), topology- or geometry-aware neural operators (Liu et al., 2023a; Wen et al., 2026), graph transformers with spectral attention (Kreuzer et al., 2021; Shehzad et al., 2026), and graph adaptation under structural shifts (Bao et al., 2025). A more detailed review is provided in Appendix A. Although these approaches use different mechanisms to account for topology changes, they share a prevailing strategy: *the effects of topology changes are ultimately captured through neural network training*.

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

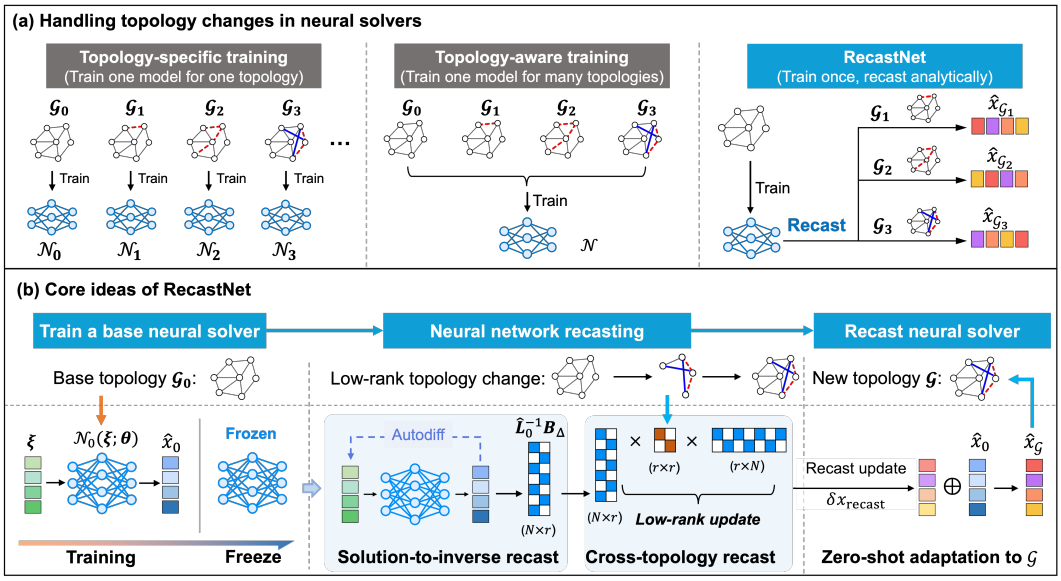


Figure 1: **Overview of RecastNet. (a)** Existing neural solvers handle topology changes by training separate models or a single model using data from multiple topologies, whereas RecastNet trains only on a base topology and analytically recasts the pretrained solver to unseen topologies. **(b)** RecastNet recovers the network Laplacian inverse from the input–output Jacobian of the base-topology neural solver and combines it with the low-rank topology change to obtain the recast solution, thereby accounting for topology changes without retraining or fine-tuning.

We take a different view. In many networked infrastructures, topology changes are not arbitrary: they often affect only a small part of the network, such as through the addition, removal, or modification of a limited number of links or components[1]. For graph-Laplacian representations (van der Schaft, 2017; Di Nino et al., 2026), such changes induce low-rank modifications of the Laplacian matrix, whose effects can be characterized analytically using low-rank matrix identities, most notably the Sherman–Morrison–Woodbury (SMW) formula (Sherman & Morrison, 1950; Woodbury, 1950). This observation leads us to ask: *Can the effect of such topology changes be incorporated **analytically** into a neural solver, rather than captured **numerically** through its training process?*

Our Contributions. We present ***Neural Network Recasting (RecastNet)***, a new approach to topology adaptation that analytically recasts a neural solver trained on a single base topology for unseen topology changes. This enables zero-shot topology adaptation without training across topologies or subsequent retraining or fine-tuning. Our main contributions are:

— *Solution-to-inverse recasting.* We establish that the input–output Jacobian of a trained neural solver can recover the inverse of the underlying network Laplacian. This architecture-agnostic recasting reveals a new capability of neural solvers beyond direct solution prediction and provides the foundation for topology recasting.

— *RecastNet for linear systems.* We derive a closed-form solution recasting for linear networked systems and characterize its recastability under unseen topology changes. We further show that this recasting requires the Laplacian inverse only along the topology-change directions, enabling efficient inference with low-dimensional matrix operations.

— *RecastNet for nonlinear systems.* We extend the recasting principle to nonlinear networked systems through Dual-RecastNet, which augments the base-topology neural solver with an auxiliary neural solver for Laplacian-inverse recasting. The auxiliary solver is also trained only on the base topology and requires no additional nonlinear solution data, enabling zero-shot recasting of nonlinear solutions under topology changes.

— *Experimental validation.* We validate RecastNet across synthetic networks and real-world power grids spanning diverse graph structures, system sizes, nodal relations, and topology-change severi-

[1]Low-rank topology changes do not necessarily imply small solution changes. As shown in Appendix B, even a single-edge removal can substantially alter the network solution.

ties. It maintains accurate zero-shot adaptation under increasingly severe topology changes and consistently outperforms the compared baselines: it recovers a median of 93.4% of topology-induced solution changes on synthetic networks, maintains below 0.8% median error under challenging line outages in AC power flow, and achieves 18–150× inference speedups over numerical solvers.

2 PRELIMINARIES

Graph notation of networked systems. Consider a network graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. After selecting a reference node, denote the number of non-reference nodes as n and the number of links as m. We denote the reduced incidence matrix as $\boldsymbol{B} \in \mathbb{R}^{n\times m}$ and the diagonal link-weight matrix as $\boldsymbol{W}(\boldsymbol{\xi}_{\mathrm{L}}) = \mathrm{diag}(\boldsymbol{w}(\boldsymbol{\xi}_{\mathrm{L}}))$, where $\boldsymbol{\xi}_{\mathrm{L}}$ represents the link conditions. The network coupling is characterized by the *grounded weighted graph Laplacian* (van der Schaft, 2017; Di Nino et al., 2026):

$$\boldsymbol{L}_{\mathcal{G}}(\boldsymbol{\xi}_{\mathrm{L}}) = \boldsymbol{B}\boldsymbol{W}(\boldsymbol{\xi}_{\mathrm{L}})\boldsymbol{B}^{\top} \tag{1}$$

For a connected network, $\boldsymbol{L}_{\mathcal{G}}$ is nonsingular.

Each node i is characterized by its local state x_i and local conditions $\boldsymbol{\xi}_i$, together with shared system conditions $\boldsymbol{\xi}_{\mathrm{sh}}$ that may affect multiple components. Under the graph-Laplacian representation, networked systems with linear or nonlinear local components take the following forms[2]:

Network with linear local components

$$\begin{aligned} &\boldsymbol{L}_{\mathcal{G}}(\boldsymbol{\xi}_{\mathrm{L}})\boldsymbol{x} = \boldsymbol{b}, \\ &b_i = b_i(\boldsymbol{\xi}_i, \boldsymbol{\xi}_{\mathrm{sh}}),\ \ i = 1, \dots, n \end{aligned} \tag{2a}$$

Network with nonlinear local components

$$\begin{aligned} &\boldsymbol{L}_{\mathcal{G}}(\boldsymbol{\xi}_{\mathrm{L}})\boldsymbol{x} = \boldsymbol{b}, \\ &b_i = f_i(x_i; \boldsymbol{\xi}_i, \boldsymbol{\xi}_{\mathrm{sh}}),\ \ i = 1, \dots, n \end{aligned} \tag{2b}$$

Here, $\boldsymbol{x}$ denotes the system solution and $\boldsymbol{b}$ the nodal injections. In the linear case, each b_i is determined by local and shared conditions; in the nonlinear case, b_i additionally depends on the local state x_i through the nonlinear relation f_i. In practice, each new instance of Eq. (2) requires solving the coupled network system anew, with computational cost growing rapidly with network size and further compounded by iterative procedures in nonlinear cases (Gardner et al., 2022).

Base-topology neural solver. We collect all varying system conditions as $\boldsymbol{\xi} = (\boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_n, \boldsymbol{\xi}_{\mathrm{sh}}, \boldsymbol{\xi}_{\mathrm{L}}) \in \mathbb{R}^p$. For a base topology $\mathcal{G}_0$ with solution $\boldsymbol{x}_0(\boldsymbol{\xi})$, a neural network $\mathcal{N}_0(\cdot;\boldsymbol{\theta})$ is trained to approximate the base-topology solution mapping $\boldsymbol{\xi} \mapsto \boldsymbol{x}_0(\boldsymbol{\xi})$, i.e., $\boldsymbol{x}_0(\boldsymbol{\xi}) \approx \mathcal{N}_0(\boldsymbol{\xi};\boldsymbol{\theta})$. We refer to $\mathcal{N}_0$ as the *base-topology neural solver*[3]. Such a learning task is typically straightforward when the topology is fixed (Pan et al., 2020; Han et al., 2024; Afrin & Ardakanian, 2026).

Low-rank topology changes. Many practical topology changes affect only a limited number of links or components and therefore induce *low-rank updates to the grounded graph Laplacian*. For a change from the base topology $\mathcal{G}_0$ to $\mathcal{G}$, the Sherman–Morrison–Woodbury (SMW) formula gives:

Low-rank topology change $\qquad\qquad$ *Sherman–Morrison–Woodbury (SMW) update*

$$\boldsymbol{L}_{\mathcal{G}} = \boldsymbol{L}_{\mathcal{G}_0} + \boldsymbol{B}_{\Delta}\boldsymbol{\Delta}\boldsymbol{B}_{\Delta}^{\top} \quad\Longrightarrow\quad \boldsymbol{L}_{\mathcal{G}}^{-1} = \boldsymbol{L}_{\mathcal{G}_0}^{-1} - \boldsymbol{L}_{\mathcal{G}_0}^{-1}\boldsymbol{B}_{\Delta}\left(\boldsymbol{\Delta}^{-1} + \boldsymbol{B}_{\Delta}^{\top}\boldsymbol{L}_{\mathcal{G}_0}^{-1}\boldsymbol{B}_{\Delta}\right)^{-1}\boldsymbol{B}_{\Delta}^{\top}\boldsymbol{L}_{\mathcal{G}_0}^{-1} \tag{3}$$

Here, $\boldsymbol{B}_{\Delta} \in \mathbb{R}^{n\times r}$ and $\boldsymbol{\Delta} \in \mathbb{R}^{r\times r}$ characterize the topology change. When $r \ll n$, the SMW update requires only an $r \times r$ matrix inversion given $\boldsymbol{L}_{\mathcal{G}_0}^{-1}$.

3 METHODOLOGY

RecastNet consists of two recasting operations on the frozen base-topology neural solver $\mathcal{N}_0$. First, *solution-to-inverse recasting* recovers the base-topology Laplacian inverse from $\mathcal{N}_0$. Second, *cross-topology solution recasting* combines the recovered inverse with the low-rank topology change through the SMW formula to obtain solutions under a changed topology. The same principle applies to both linear and nonlinear networked systems.

In the following discussion, $\widehat{\cdot}$ denotes a quantity predicted by neural networks or approximated through recasting, while its unhatted counterpart denotes the corresponding exact quantity.

[2]For notational simplicity, Eq. (2) is presented with a scalar state per node. Vector-valued local states extend naturally through block-structured formulations in RecastNet, while preserving zero-shot recasting and low-rank computational structure. Our AC power flow experiments in Section 4 provide a concrete example.

[3]This work does not restrict the architecture of $\mathcal{N}_\theta$; the only requirement is that $\mathcal{N}_0$ be differentiable with respect to $\boldsymbol{\xi}$, as is typically the case for standard neural network models.

3.1 SOLUTION-TO-INVERSE RECASTING

Jacobian-based inverse recasting. We begin with the network equation in Eq. (2a). Given a base-topology neural solver $\mathcal{N}_0(\boldsymbol{\xi};\boldsymbol{\theta}) \approx \boldsymbol{x}_0(\boldsymbol{\xi})$, we show that the *base-topology Laplacian inverse* $\boldsymbol{L}_{\mathcal{G}_0}^{-1}(\boldsymbol{\xi})$ can be recovered from the input–output Jacobian of $\mathcal{N}_0$, and characterize its recovery condition and approximation error.

***Theorem* 1** (*Solution-to-inverse recasting*)**.** *Assume that $\boldsymbol{L}_{\mathcal{G}_0}$, $\boldsymbol{b}$, and $\mathcal{N}_0$ are differentiable with respect to $\boldsymbol{\xi}$. Denote $\boldsymbol{J}_{\mathcal{N}_0}(\boldsymbol{\xi}) := \partial\mathcal{N}_0(\boldsymbol{\xi};\boldsymbol{\theta})/\partial\boldsymbol{\xi}$ as the input–output Jacobian of $\mathcal{N}_0$, and define a sensitivity matrix $\boldsymbol{S}_{\mathcal{N}_0}(\boldsymbol{\xi}) := [\boldsymbol{s}_1, \ldots, \boldsymbol{s}_p]$ with $\boldsymbol{s}_j := \partial\boldsymbol{b}/\partial\xi_j - (\partial\boldsymbol{L}_{\mathcal{G}_0}/\partial\xi_j)\mathcal{N}_0$. If $\boldsymbol{S}_{\mathcal{N}_0}(\boldsymbol{\xi})$ has full row rank, the base-topology Laplacian inverse can be recast as:*

$$\widehat{\boldsymbol{L}}_{\mathcal{G}_0}^{-1}(\boldsymbol{\xi}) := \boldsymbol{J}_{\mathcal{N}_0}(\boldsymbol{\xi})\boldsymbol{S}_{\mathcal{N}_0}(\boldsymbol{\xi})^{\dagger} \approx \boldsymbol{L}_{\mathcal{G}_0}^{-1}(\boldsymbol{\xi}) \tag{4}$$

where † denotes the Moore–Penrose pseudoinverse.

The complete proof is provided in Appendix C.1.1. The key idea is to differentiate the base-topology network equation $\boldsymbol{L}_{\mathcal{G}_0}(\boldsymbol{\xi})\boldsymbol{x}_0(\boldsymbol{\xi}) = \boldsymbol{b}(\boldsymbol{\xi})$ with respect to each condition ξ_j, yielding a collection of derivative relations that share the same $\boldsymbol{L}_{\mathcal{G}_0}$. When the sensitivity matrix $\boldsymbol{S}_{\mathcal{N}_0}$ has full row rank, these relations span the n-dimensional space, making the complete Laplacian inverse identifiable. In the formulation of Eq. (2a), the node-wise conditions in $\boldsymbol{\xi}$ provide the variations needed to support this rank condition in practical settings.

Remark 1. Full inverse recovery is sufficient but not necessary for topology recasting. When $\boldsymbol{S}_{\mathcal{N}_0}$ is rank deficient, the inverse action remains recoverable on the subspace spanned by the available sensitivity directions, so recasting remains possible when the topology-change directions lie in this subspace. Further details are provided in Appendix C.1.2.

Remark 2. Eq. (4) gives the general form of solution-to-inverse recasting, but RecastNet does not require the full pseudoinverse $\boldsymbol{S}_{\mathcal{N}_0}^{\dagger}$. For low-rank topology changes, RecastNet requires only a low-dimensional inversion along the topology-change directions. See Section 3.2 for details.

Error bound of inverse recasting. In practice, $\mathcal{N}_0$ only approximates the true solution mapping. The accuracy of Eq. (4) therefore depends on how this approximation error propagates through the recasting process. Appendix C.1.3 derives the corresponding error bound.

Together, these results show that a lightweight "repurposing" of the base-topology neural solver can provide the Laplacian-inverse information needed for topology adaptation, without additional training or parameter updates. This forms the basis for RecastNet.

3.2 ZERO-SHOT RECASTNET FOR LINEAR SYSTEMS

We next develop cross-topology solution recasting for the linear system in Eq. (2a). Given the base-topology neural solver $\mathcal{N}_0$ and a low-rank topology change from $\mathcal{G}_0$ to $\mathcal{G}$, we show how the recovered Laplacian-inverse information yields a *zero-shot solution mapping* for $\mathcal{G}$.

Cross-topology solution recasting. For a low-rank topology change $\boldsymbol{L}_{\mathcal{G}} = \boldsymbol{L}_{\mathcal{G}_0} + \boldsymbol{B}_{\Delta}\boldsymbol{\Delta}\boldsymbol{B}_{\Delta}^{\top}$, we define the *directional Laplacian-inverse action* as:

$$\boldsymbol{H}_{\Delta} := \boldsymbol{L}_{\mathcal{G}_0}^{-1}\boldsymbol{B}_{\Delta} \tag{5}$$

which captures the Laplacian-inverse action along the topology-change directions $\boldsymbol{B}_{\Delta}$.

For a system condition $\boldsymbol{\xi}$, the base-topology neural solver predicts $\mathcal{N}_0(\boldsymbol{\xi};\boldsymbol{\theta}) \approx \boldsymbol{L}_{\mathcal{G}_0}^{-1}\boldsymbol{b}$. Applying the SMW formula in Eq. (3), the solution under the changed topology, $\boldsymbol{x}_{\mathcal{G}} = \boldsymbol{L}_{\mathcal{G}}^{-1}\boldsymbol{b}$, can be written as:

$$\widehat{\boldsymbol{x}}_{\mathcal{G}} = \mathcal{N}_0(\boldsymbol{\xi};\boldsymbol{\theta}) - \boldsymbol{H}_{\Delta}\left(\boldsymbol{\Delta}^{-1} + \boldsymbol{B}_{\Delta}^{\top}\boldsymbol{H}_{\Delta}\right)^{-1}\boldsymbol{B}_{\Delta}^{\top}\mathcal{N}_0(\boldsymbol{\xi};\boldsymbol{\theta}) \tag{6}$$

Recast solution · Low-rank topology change · Base-topology prediction · Directional Laplacian-inverse action

Eq. (6) expresses the *recast solution* as the base-topology prediction with a topology-induced correction that involves the base-topology Laplacian inverse only through $\boldsymbol{H}_{\Delta}$. Thus, rather than recasting the full $\boldsymbol{L}_{\mathcal{G}_0}^{-1}$ in Eq. (4), RecastNet only needs the directional inverse action $\boldsymbol{H}_{\Delta}$.

To obtain $\boldsymbol{H}_\Delta$ efficiently, we exploit the structure of the sensitivity matrix defined in Theorem 1. For a given topology change, we retain the local, shared, and affected-link conditions to form $\boldsymbol{S}_{\mathcal{N}_0,\Delta}$ and $\boldsymbol{J}_{\mathcal{N}_0,\Delta}$. As shown in Appendix C.2.1, this preserves the Laplacian-inverse information required for $\boldsymbol{H}_\Delta$ while exposing a diagonal-plus-low-rank structure, leading to the following result.

Proposition 2 (*Linear solution recasting*)**.** *Consider a rank-r topology change $\boldsymbol{\Delta}L = \boldsymbol{B}_\Delta \boldsymbol{\Delta} \boldsymbol{B}_\Delta^\top$.*

(i) *Its directional Laplacian-inverse action can be recast from the base-topology neural solver as $\widehat{\boldsymbol{H}}_\Delta = \boldsymbol{J}_{\mathcal{N}_0,\Delta} \boldsymbol{S}^\dagger_{\mathcal{N}_0,\Delta} \boldsymbol{B}_\Delta$. Substituting $\widehat{\boldsymbol{H}}_\Delta$ in Eq. (6) gives the zero-shot recast solution under $\mathcal{G}$.*

(ii) *$\widehat{\boldsymbol{H}}_\Delta$ can be evaluated without explicitly forming $\boldsymbol{S}^\dagger_{\mathcal{N}_0,\Delta}$. Exploiting the diagonal-plus-low-rank structure of $\boldsymbol{S}_{\mathcal{N}_0,\Delta}$, its construction reduces to a matrix inversion of dimension at most $(p_{\mathrm{sh}} + r)$. Consequently, the zero-shot solution recasting in Eq. (6) requires only low-dimensional matrix inversions whose sizes scale with the topology-change rank r, rather than the network dimension n.*

Proposition 2 completes the linear solution recasting: combining the frozen base-topology neural solver with the recast directional Laplacian-inverse action yields the solution for an unseen topology $\mathcal{G}$ in a zero-shot manner. The RecastNet inference requires only automatic differentiation and low-dimensional matrix operations beyond the base neural-solver forward pass, all readily supported by modern ML frameworks (Paszke et al., 2019). Appendix C.2 provides the derivation of Proposition 2 and details the batched inference implementation.

Linear recastability. We next formalize the recastability of a neural solver under topology changes. Given an error threshold $\epsilon > 0$, we define $\mathcal{N}_0$ to be *ϵ-recastable* from $\mathcal{G}_0$ to $\mathcal{G}$ if the relative error of the recast solution remains within ϵ:

$$\mathcal{N}_0 \text{ is } \epsilon\textit{-recastable for } \mathcal{G}_0 \to \mathcal{G}, \quad \text{if} \quad \frac{\|\widehat{\boldsymbol{x}}_{\mathcal{G}}(\boldsymbol{\xi}) - \boldsymbol{x}_{\mathcal{G}}(\boldsymbol{\xi})\|_2}{\|\boldsymbol{x}_{\mathcal{G}}(\boldsymbol{\xi})\|_2} \le \epsilon. \tag{7}$$

The following corollary provides a sufficient condition for ϵ-recastability.

Corollary 3 (*Sufficient condition for ϵ-recastability*)**.** *For a topology change $\mathcal{G}_0 \to \mathcal{G}$, define the base-topology prediction error $\boldsymbol{\varepsilon}_0 := \mathcal{N}_0(\boldsymbol{\xi};\boldsymbol{\theta}) - \boldsymbol{x}_0$ and the directional inverse-action error $\boldsymbol{E}_{H,\Delta} := \widehat{\boldsymbol{H}}_\Delta - \boldsymbol{H}_\Delta$. Then the recast-solution error is bounded by:*

$$\|\widehat{\boldsymbol{x}}_{\mathcal{G}} - \boldsymbol{x}_{\mathcal{G}}\|_2 \le c_{0,\Delta}\|\boldsymbol{\varepsilon}_0\|_2 + c_{H,\Delta}\|\boldsymbol{E}_{H,\Delta}\|_2 := \mathcal{E}_\Delta \tag{8}$$

where $c_{0,\Delta}$ and $c_{H,\Delta}$ are topology-dependent error-propagation coefficients, with upper bounds given in Appendix C.2.3. Therefore, $\mathcal{E}_\Delta \le \epsilon\|\boldsymbol{x}_{\mathcal{G}}\|_2$ guarantees that $\mathcal{N}_0$ is ϵ-recastable *from $\mathcal{G}_0$ to $\mathcal{G}$.*

The derivation is provided in Appendix C.2.3, showing that recastability depends not only on the accuracy of the frozen neural solver and the recovered inverse action, but also on how a given topology change amplifies these errors. As a result, the same frozen base-topology neural solver can exhibit different recastability across different topology changes.

3.3 Dual-RecastNet for Nonlinear Systems

We next extend RecastNet to the nonlinear systems in Eq. (2b). Unlike the linear case, the nodal injections $\boldsymbol{b}$ in nonlinear systems depend on the unknown solution itself, i.e., $\boldsymbol{L}_{\mathcal{G}}\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x},\boldsymbol{\xi})$. As a result, the closed-form recasting in Section 3.2 does not directly yield the solution under the changed topology. We therefore introduce *Dual-RecastNet*, which combines the base-topology neural solver with an auxiliary neural solver for Laplacian-inverse recasting, enabling zero-shot solution recasting in the nonlinear setting.

Topology impacts on nonlinear solutions. For the same low-rank topology change $\boldsymbol{L}_{\mathcal{G}} = \boldsymbol{L}_{\mathcal{G}_0} + \Delta\boldsymbol{L}$, denote the corresponding solution change as $\Delta\boldsymbol{x}_{\mathcal{G}} := \boldsymbol{x}_{\mathcal{G}} - \boldsymbol{x}_0$. A local expansion of Eq. (2b) around the base-topology solution gives:

$$\Delta\boldsymbol{x}_{\mathcal{G}} \approx \boldsymbol{L}_{\mathcal{G}}^{-1}\left(-\Delta\boldsymbol{L}\,\boldsymbol{x}_0 + \boldsymbol{D}_0\Delta\boldsymbol{x}_{\mathcal{G}}\right) \tag{9}$$

where $\boldsymbol{D}_0 := \partial\boldsymbol{f}/\partial\boldsymbol{x}|_{\boldsymbol{x}=\boldsymbol{x}_0}$ denotes the Jacobian of $\boldsymbol{f}$. Eq. (9) requires both the base-topology solution $\boldsymbol{x}_0$ and the changed-topology Laplacian inverse $\boldsymbol{L}_{\mathcal{G}}^{-1}$. The former is provided by the base-topology neural solver, while the latter follows from $\boldsymbol{L}_{\mathcal{G}_0}^{-1}$ through the SMW formula. The challenge is therefore to recover $\boldsymbol{L}_{\mathcal{G}_0}^{-1}$: since the base-topology neural solver learns the full nonlinear solution mapping $\boldsymbol{\xi} \mapsto \boldsymbol{x}_0$, the Laplacian inverse is not directly exposed for recasting.

Dual neural solvers. This motivates Dual-RecastNet, which couples two neural solvers with complementary roles: a *base-topology neural solver* $\mathcal{N}_0^{\text{sol}} : \boldsymbol{\xi} \mapsto \boldsymbol{x}_0$ that predicts the base-topology nonlinear solution, and an *auxiliary neural solver* $\mathcal{N}_0^{\text{aux}} : \boldsymbol{b} \mapsto \delta\boldsymbol{x}$ trained on pairs satisfying $\boldsymbol{L}_{\mathcal{G}_0}\delta\boldsymbol{x} = \boldsymbol{b}$ (see Appendix C.3 for details of $(\boldsymbol{b}, \delta\boldsymbol{x})$ pairs[4]). Because $\mathcal{N}_0^{\text{aux}}$ is governed directly by $\boldsymbol{L}_{\mathcal{G}_0}$, Theorem 1 allows us to recast the base-topology Laplacian inverse $\widehat{\boldsymbol{L}}_{\mathcal{G}_0}^{-1}$. For a low-rank topology change, applying the SMW formula to $\widehat{\boldsymbol{L}}_{\mathcal{G}_0}^{-1}$ yields the changed-topology Laplacian inverse. Combined with the base-topology neural solver, this yields the *nonlinear solution recasting*:

Nonlinear recast solution

$$\widehat{\boldsymbol{x}}_{\mathcal{G}}^{(k+1)} = \widehat{\boldsymbol{x}}_{\mathcal{G}}^{(k)} + \widehat{\boldsymbol{L}}_{\mathcal{G}}^{-1}\left(\boldsymbol{f}(\widehat{\boldsymbol{x}}_{\mathcal{G}}^{(k)}, \boldsymbol{\xi}) - \boldsymbol{L}_{\mathcal{G}}\widehat{\boldsymbol{x}}_{\mathcal{G}}^{(k)}\right), \quad \widehat{\boldsymbol{x}}_{\mathcal{G}}^{(0)} = \mathcal{N}_0^{\text{sol}}(\boldsymbol{\xi}; \boldsymbol{\theta}_{\text{sol}}). \tag{10}$$

Changed-topology Laplacian inverse

Base-topology nonlinear prediction

Nonlinear recastability. For nonlinear systems, the recast solution is state-dependent (see Eq. (10)), so recastability further depends on whether the recasting steps converge to the true solution under topology $\mathcal{G}$. We provide the following sufficient condition for such convergence.

***Corollary* 4** (*Sufficient condition for nonlinear recastability*)**.** *Define the inverse-recasting error* $\boldsymbol{E}_{\mathcal{G}} := \widehat{\boldsymbol{L}}_{\mathcal{G}}^{-1} - \boldsymbol{L}_{\mathcal{G}}^{-1}$ *and the nonlinear Jacobian* $\boldsymbol{D}(\boldsymbol{x}) := \partial\boldsymbol{f}/\partial\boldsymbol{x}$*. If there exists a neighborhood* $\mathcal{U}$ *of* $\boldsymbol{x}_{\mathcal{G}}$ *such that* $\sup_{\boldsymbol{x}\in\mathcal{U}} \left\|\boldsymbol{L}_{\mathcal{G}}^{-1}\boldsymbol{D}(\boldsymbol{x}) + \boldsymbol{E}_{\mathcal{G}}\left(\boldsymbol{D}(\boldsymbol{x}) - \boldsymbol{L}_{\mathcal{G}}\right)\right\|_2 < 1$*, then the recasting in Eq. (10) locally converges to the true solution* $\boldsymbol{x}_{\mathcal{G}}$ *under topology* $\mathcal{G}$*.*

Corollary 4 shows that nonlinear recastability depends on both the local nonlinear behavior and the accuracy of the recast Laplacian inverse. Notably, they affect the local convergence condition and rate, but not the final solution $\boldsymbol{x}_{\mathcal{G}}$ directly. Detailed discussions are provided in Appendix C.3.

4 Experiments

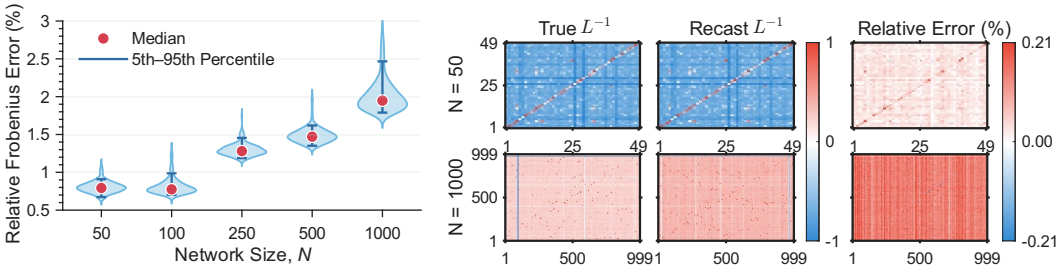
We evaluate RecastNet on synthetic networks and real-world electric power grids, covering linear and nonlinear systems, diverse network structures and sizes, different nodal relations, and topology changes of varying severity. RecastNet is compared with: 1) *Base MLP*, the frozen base-topology neural solver directly applied to unseen topologies without recasting (Liang et al., 2025); 2) *Graph Convolutional Network (GCN)*, which incorporates network topology through graph convolutions (Wu et al., 2020); 3) *Topology-Aware MLP*, which takes topology information as input, with optional *fine-tuning* for individual target topologies (Gupta et al., 2022; Lim et al., 2021); and 4) *PowerFlowNet*, a power-system-specific message-passing graph neural network (Lin et al., 2024). Unlike RecastNet, which is trained only on the base topology, baselines 2–4 incorporate topology information into training through model inputs or topology-dependent training data. Numerical solvers, specifically SuperLU and Newton–Raphson (Virtanen et al., 2020), provide ground-truth solutions and runtime references. Further experimental details are provided in Appendix D.

4.1 Performance on Synthetic Networks

We first evaluate RecastNet on three representative synthetic network families (Adam-Day et al., 2024): Erdős–Rényi network (ER; random graph), stochastic block model (SBM; community-structured graph), and Barabási–Albert network (BA; preferential-attachment network).

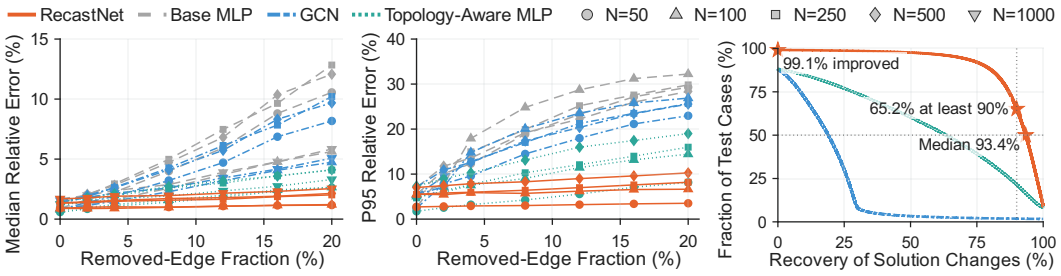
Fidelity of solution-to-inverse recasting. Solution-to-inverse recasting is the key mechanism that enables RecastNet to repurpose a neural solver beyond direct solution prediction. Figure 2 validates this mechanism on ER networks with 50–1000 nodes under the linear nodal relation. As shown in Figure 2a, the recast inverse remains accurate as network size increases, with only a moderate increase in relative Frobenius error. Figure 2b further compares the recast and exact inverses entry-wise for the 50- and 1000-node networks: the two matrices closely agree across entries, and even for the 1000-node network, the entry-wise relative errors remain within $\pm 0.21\%$. These results directly support Theorem 1, showing that the network Laplacian inverse can be accurately recast from a trained neural solver.

[4]Importantly, training the auxiliary neural solver requires no additional nonlinear solution data. Intermediate quantities generated alongside the base-topology nonlinear training instances can be retained to construct the required $(\boldsymbol{b}, \delta\boldsymbol{x})$ pairs; see Appendix C.3 for details.



(a) Relative error of the recast inverse. (b) Entry-wise comparison between the recast and exact inverses

Figure 2: **Accuracy of solution-to-inverse recasting. (a)** The recast inverse remains accurate as network size increases. **(b)** Entry-wise comparisons show close agreement with the exact inverse even at 1000 nodes.



(a) Median relative solution error. (b) P95 relative solution error. (c) Solution-change recovery.

Figure 3: **Accuracy of RecastNet on ER networks. (a)–(b)** RecastNet maintains low median and P95 solution errors as the fraction of removed edges increases, while the baselines degrade substantially. **(c)** RecastNet more accurately captures the topology-induced solution changes, achieving substantially higher solution-change recovery (see the metric definition in Eq. (53)).

Accuracy of RecastNet solutions. We evaluate the solution accuracy of RecastNet under unseen topology changes. Figures 3a and 3b report the median and P95 relative errors on ER networks of different sizes, with topology changes affecting up to 20% of the edges. As the topology change becomes more severe, the Base MLP, GCN, and Topology-Aware MLP degrade substantially, whereas RecastNet maintains consistently low errors across network sizes. Figure 3c further evaluates how accurately each method recovers topology-induced solution changes, with the Base MLP corresponding to zero recovery (see Appendix D.3). As shown, RecastNet achieves at least 90% recovery in 65.2% of the test cases, compared with 21.6% of the test cases for the Topology-Aware MLP. For half of the cases, recovery reaches at least 93.4% with RecastNet, versus 63.4% with the Topology-Aware MLP. Moreover, RecastNet outperforms the Base MLP in 99.1% of the cases.

Table 1 extends the evaluation to ER, SBM, and BA networks with different nodal relations. Across 10%–20% topology changes, RecastNet maintains solution errors of 1.18–4.51% in capturing topology-induced solution changes, with gains of 84.6–95.4%.

Table 1: **Accuracy of RecastNet across networks and nodal relations.** Results are reported on 250-node ER, SBM, and BA networks, with topology changes affecting 10% and 20% of the edges.

	ER (10%)		ER (20%)		SBM (10%)		SBM (20%)		BA (10%)		BA (20%)	
Nodal Relation[1]	Error[2]	Gain[3]	Error	Gain	Error	Gain	Error	Gain	Error	Gain	Error	Gain
Linear	2.60	92.1	3.09	91.8	2.58	93.9	3.32	90.9	1.18	93.7	1.51	92.8
Quadratic	2.73	91.9	3.19	91.9	2.62	95.0	3.39	91.2	1.19	95.2	1.59	93.0
Trigonometric	2.95	92.5	3.77	89.4	2.88	92.1	3.98	87.2	1.21	95.4	1.87	90.2
Mixed	3.11	92.5	4.51	84.6	3.09	89.1	4.22	85.5	1.61	93.1	2.01	92.0

[1] Linear, quadratic, trigonometric, and mixed denote the nodal relations $b_i(\xi_i)$ in Eq. (2); see Appendix D.
[2] Error (%) denotes the relative error between the recast and true solutions; lower is better.
[3] Gain of recasting (%) measures how much of the additional solution error after a topology change is reduced by recasting; higher is better, as defined in Eq. (52).

4.2 Performance on Real-World Power Networks

We next evaluate RecastNet on standard power system benchmarks from MATPOWER (Zimmerman et al., 2010), spanning systems from 57 to 1,354 buses. Topology changes represent transmission-line outages and network reconfigurations in grid operation.

Accuracy on real-world power grids. Table 2 evaluates RecastNet on four benchmark grids under unseen $N-1$ line outages. We consider both DC and AC power flow, covering linearized and nonlinear power system models. Across both settings, RecastNet achieves substantially lower median and P95 relative solution errors and higher recovery of topology-induced solution changes than the Topology-Aware MLP and PowerFlowNet baselines. These results validate RecastNet's effectiveness for power flow applications at realistic grid scales.

Table 2: Accuracy of RecastNet on real-world power grids. Results report relative solution errors (%) under unseen $N-1$ line contingencies for DC and AC power flow.

	RecastNet (DC)		Topo-Aware MLP (DC)		RecastNet (AC)		PowerFlowNet (AC)	
System	Median[1]	P95[2]	Median	P95	Median	P95	Median	P95
IEEE 57-Bus	0.40	3.94	1.10	5.43	0.01	1.13	0.40	0.83
IEEE 118-Bus	0.27	1.93	1.43	4.40	0.01	0.31	1.33	3.33
Carolina 500-Bus	0.70	7.07	1.76	9.77	0.19	2.42	1.50	2.85
European 1354-Bus	0.22	1.59	0.47	2.08	0.46	1.04	3.88	7.62

[1,2] Median (%) and P95 (%) denote the median and 95th-percentile relative solution errors (%), respectively.

Recasting under severe line outages and grid reconfiguration. We further evaluate RecastNet on AC power flow under topology changes with increasing operational impact. In power grids, disconnecting a transmission line forces its power flow to be redistributed through the remaining network, potentially causing a large change in the operating state (Sadik & Narimani, 2023). We therefore group outages by pre-contingency flow to represent outage impact; the high-impact group has a median flow $6.3\times$ that of the low-impact group. As shown in Figure 4, the median solution errors after two recasts remain only 0.057%, 0.173%, and 0.352% for low-, medium-, and high-impact outages, respectively. We additionally consider grid reconfiguration, where multiple transmission lines are switched simultaneously, producing a more substantial change in network topology. Even under this setting, the median solution error remains only 0.732%. These results show that RecastNet maintains high accuracy across increasingly impactful line outages and more substantial multi-line topology changes.

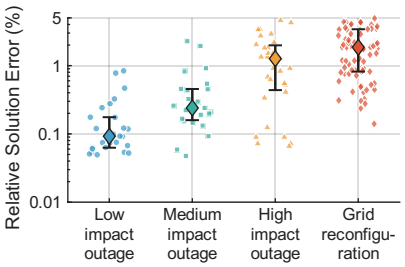


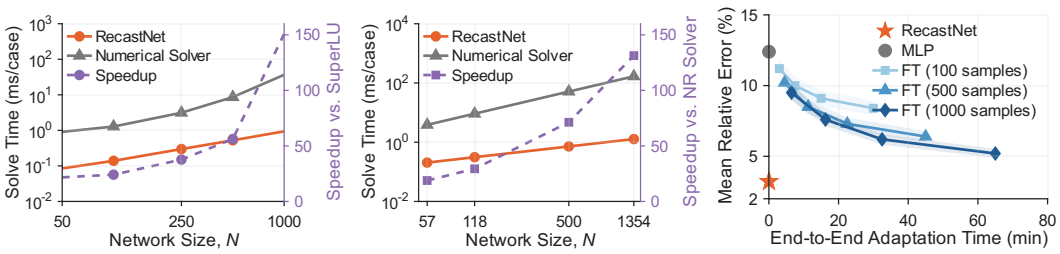
Figure 4: **RecastNet under line outages and grid reconfiguration.** Relative solution errors on the South Carolina 500-bus AC power flow after two recasts.

4.3 Computational Efficiency of RecastNet

The computational efficiency of RecastNet is evaluated in terms of both inference and topology-adaptation costs. Figures 5a and 5b show that, despite the additional recasting operations, RecastNet retains efficient batched inference and achieves substantial speedups over numerical solvers in both linear and nonlinear settings. Figure 5c further compares the cost of adapting to unseen topologies: RecastNet requires no target-topology training but achieves higher accuracy; fine-tuning incurs additional data-generation and training costs that grow with the number of target-topology samples, while providing only limited accuracy improvements.

4.4 Empirical Characterization of Recastability

We finally examine the recastability conditions established in Corollaries 3 and 4. Following the ϵ-recastability definition in Eq. (7), we group the tested topology changes by their achieved recast-solution error into high ($\epsilon \leq 1\%$), middle (1%–2%), and low (2%–10%) recastability levels.

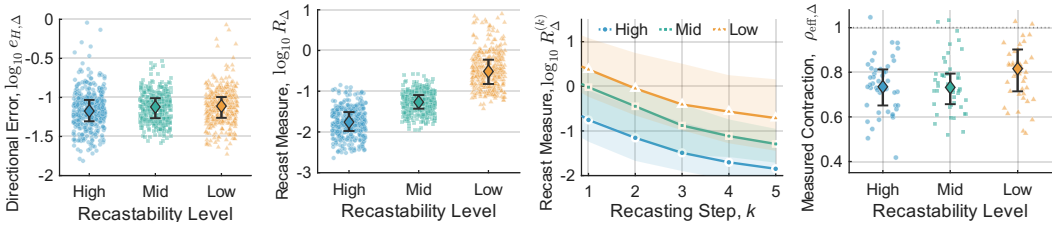


(a) Inference time (linear). (b) Inference time (nonlinear). (c) Target-topology training time.

Figure 5: **Computational efficiency of RecastNet. (a)–(b)** With a batch size of 512, RecastNet achieves 26–150× and 18–130× inference speedups over numerical solvers for linear and nonlinear systems, respectively. **(c)** RecastNet requires no target-topology training, whereas fine-tuning incurs increasing end-to-end adaptation cost.

For linear recasting, we examine why recastability varies across topology changes using DC power flow. Corollary 3 bounds the recast-solution error by $E_\Delta = c_{0,\Delta}\|\boldsymbol{\epsilon}_0\|_2 + c_{H,\Delta}\|\boldsymbol{E}_{H,\Delta}\|_2$, where $c_{0,\Delta}$ and $c_{H,\Delta}$ characterize how the two errors propagate under a given topology change. Figures 6a–6b therefore examine the directional inverse-action error $e_{H,\Delta} := \|\widehat{\boldsymbol{H}}_\Delta - \boldsymbol{H}_\Delta\|_F / \|\boldsymbol{H}_\Delta\|_F$ and the recastability measure $R_\Delta := E_\Delta / \|\boldsymbol{x}_\mathcal{G}\|_2$. While the median $\log_{10} e_{H,\Delta}$ is nearly unchanged across the three groups (-1.17 to -1.11), $\log_{10} R_\Delta$ increases markedly from -1.76 to -0.51. Thus, differences in linear recastability appear to be driven less by the accuracy of the recovered directional inverse-action, but more by how that error is amplified under different topology changes.

For nonlinear recasting, we further examine how the recastability measure evolves as recasting proceeds using AC power flow. As shown in Figure 6c, R_Δ consistently decreases, indicating progressively improved recastability and solution accuracy. Figure 6d further evaluates the contraction of the recasting map, which governs convergence as derived in Corollary 4. The measured contraction factors are predominantly below one, indicating local contractive behavior, while the larger median factor in the low-recastability group (0.82 versus 0.73) indicates moderately weaker contraction.



(a) Directional inverse-action error. (b) Linear recastability measure. (c) Nonlinear recastability measure. (d) Empirical contraction in nonlinear recasting.

Figure 6: **Empirical characterization of recastability. (a)–(b)**: Similar directional inverse-action errors can lead to markedly different recastability because their effects are amplified differently across topology changes. **(c)–(d)**: For nonlinear recasting, recastability improves across recasting steps, while contraction below one indicates locally contractive behavior across all three groups.

5 Conclusion

This work introduces RecastNet, a framework for adapting neural solvers to unseen topology changes without retraining. RecastNet shifts topology adaptation from learning to analytical recasting at inference, allowing a neural solver trained on a single base topology to be repurposed for changed topologies. We develop a unified recasting framework for both linear and nonlinear networked systems and characterize when such recasting remains accurate under topology changes. Across synthetic networks and real-world power grids, RecastNet maintains accurate zero-shot solutions under diverse and severe topology changes while requiring only lightweight test-time computation. Overall, our results establish neural network recasting as a new paradigm for topology adaptation, where structural changes are handled analytically rather than learned from cross-topology data. Future work will extend RecastNet to networked dynamical systems.

AI USE STATEMENT

In this work, the authors used generative AI tools to assist with language editing and improving readability, as well as with code drafting and debugging during research execution. The authors did not use generative AI tools to develop the research ideas or methodology, or design the experiments.

All AI-assisted text was reviewed and edited by the authors, and AI-assisted code was independently checked and validated before use. The authors take responsibility for the final content of this work, including text, code, claims, or other artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Our work follows the ICLR Code of Ethics. It does not involve any human or animal subjects, experiments, or tests. It does not use any privacy-sensitive, security-sensitive, or legally restricted data or information. The experiments use synthetic networks and standard publicly available power system benchmarks in accordance with proper licensing and usage guidelines. We are not aware of any ethical or societal risks specific to this work. This work is only for standard research purposes.

REPRODUCIBILITY STATEMENT

We are dedicated to ensuring the reproducibility of our results. The main paper describes the core method and experimental protocol, while the appendix provides details on the datasets, model architectures, baseline methods, evaluation metrics, training and implementation settings, and mathematical derivations. We will release the data, models, code, and detailed instructions for reproducing the experiments upon publication.

REFERENCES

Sam Adam-Day, Michael Benedikt, İsmail İ Ceylan, and Ben Finkelshtein. Almost surely asymptotically constant graph neural networks. *Advances in Neural Information Processing Systems*, 37:124843–124886, 2024.

Afia Afrin and Omid Ardakanian. Storm-opf: Robustness benchmarking for ac-opf methods. In *Proceedings of the 17th ACM International Conference on Future and Sustainable Energy Systems*, pp. 521–533, 2026.

Wenxuan Bao, Zhichen Zeng, Zhining Liu, Hanghang Tong, and Jingrui He. Matcha: Mitigating graph structure shifts with test-time adaptation. In *International Conference on Learning Representations*, volume 2025, pp. 56906–56944, 2025.

Julius Berner, Miguel Liu-Schiaffini, Jean Kossaifi, Valentin Duruisseaux, Boris Bonev, Kamyar Azizzadenesheli, and Anima Anandkumar. Principled approaches for extending neural architectures to function spaces for operator learning. *Nature Machine Intelligence*, pp. 1–9, 2026.

Lucas Brunel, Mathieu Balesdent, Loïc Brevault, Rodolphe Le Riche, and Bruno Sudret. A survey on multi-fidelity surrogates for simulators with functional outputs: Unified framework and benchmark. *Computer Methods in Applied Mechanics and Engineering*, 435:117577, 2025.

Tingting Dan, Xinwei Huang, Won Hwa Kim, and Guorong Wu. Explore in-context message passing operator for graph neural networks in a mean field game. *Advances in Neural Information Processing Systems*, 38:115533–115557, 2026.

Leonardo Di Nino, Gabriele D'Acunto, Sergio Barbarossa, and Paolo Di Lorenzo. Learning the structure of connection graphs. In *ICASSP 2026-2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 76–80. IEEE, 2026.

Mehdi Dini, Amin Mohammadikaleibar, Saeed Hashemi, and Vahid Nourani. Stochastic long-term reliability of water distribution networks using monte carlo simulation. *Urban Water Journal*, 19 (2):151–160, 2022.

Balthazar Donon, Rémy Clément, Benjamin Donnot, Antoine Marot, Isabelle Guyon, and Marc Schoenauer. Neural networks for power flow: Graph neural solver. *Electric Power Systems Research*, 189:106547, 2020a.

Balthazar Donon, Benjamin Donnot, Isabelle Guyon, Zhengying Liu, Antoine Marot, Patrick Panciatici, and Marc Schoenauer. Leap nets for system identification and application to power systems. *Neurocomputing*, 416:316–327, 2020b.

Balthazar Donon, Zhengying Liu, Wenzhuo Liu, Isabelle Guyon, Antoine Marot, and Marc Schoenauer. Deep statistical solvers. *Advances in Neural Information Processing Systems*, 33:7910–7921, 2020c.

Florian Dorfler and Francesco Bullo. Kron reduction of graphs with applications to electrical networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(1):150–163, 2012.

Hong Fan, Xu Huang, Diwei Wang, and Boyang Zhou. Communication network robust routing optimization in an integrated energy cyber–physical system based on a random denial-of-service attack. *Frontiers in Energy Research*, 12:1382887, 2024.

David J Gardner, Daniel R Reynolds, Carol S Woodward, and Cody J Balos. Enabling new flexibility in the sundials suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 48(3):1–24, 2022.

Asif Gulraiz, Syed Sajjad Haider Zaidi, Haseeb Gulraiz, Bilal Muhammad Khan, Majid Ali, and Baseem Khan. Leveraging dense layer hybrid graph neural networks for managing overvoltage in pv-dominated distribution systems. *Results in Engineering*, 27:106169, 2025.

Jayesh Gupta, Sai Vemprala, and Ashish Kapoor. Learning modular simulations for homogeneous systems. *Advances in Neural Information Processing Systems*, 35:14852–14864, 2022.

Jiayu Han, Wei Wang, Chao Yang, Mengyang Niu, Cheng Yang, Lei Yan, and Zuyi Li. Frmnet: a feasibility restoration mapping deep neural network for ac optimal power flow. *IEEE Transactions on Power Systems*, 39(5):6566–6577, 2024.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

Chung-Wen Ho, Albert Ruehli, and Pierce Brennan. The modified nodal approach to network analysis. *IEEE Transactions on circuits and systems*, 22(6):504–509, 1975.

Hans Hao-Hsun Hsu, Shikun Liu, Han Zhao, and Pan Li. Structural alignment improves graph test-time adaptation. *arXiv preprint arXiv:2502.18334*, 2025.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Wei Jin, Tong Zhao, Jiayuan Ding, Yozen Liu, Jiliang Tang, and Neil Shah. Empowering graph representation learning with test-time graph transformation. *arXiv preprint arXiv:2210.03561*, 2022.

Dawid Kopiczko, Tijmen Blankevoort, and Yuki Asano. Vera: Vector-based random matrix adaptation. In *International Conference on Learning Representations*, volume 2024, pp. 6815–6835, 2024.

Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in neural information processing systems*, 34:21618–21629, 2021.

Jin Lee, Ziming Liu, Xinling Yu, Yixuan Wang, Haewon Jeong, Murphy Niu, and Zheng Zhang. Kano: Kolmogorov-arnold neural operator. In *International Conference on Learning Representations*, volume 2026, pp. 71003–71031, 2026.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. Relora: High-rank training through low-rank updates. In *International Conference on Learning Representations*, volume 2024, pp. 49405–49421, 2024.

Dong Liang, Guirong Li, Xiaoyu Liu, Lin Zeng, Hsiao-Dong Chiang, and Shouxiang Wang. Bayesian state estimation for partially observable distribution networks via power flow-informed neural networks. *International Journal of Electrical Power & Energy Systems*, 170:110886, 2025.

Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. *Advances in neural information processing systems*, 34:20887–20902, 2021.

Nan Lin, Stavros Orfanoudakis, Nathan Ordonez Cardenas, Juan S Giraldo, and Pedro P Vergara. Powerflownet: Power flow approximation using message passing graph neural networks. *International Journal of Electrical Power & Energy Systems*, 160:110112, 2024.

Ning Liu, Siavash Jafarzadeh, and Yue Yu. Domain agnostic fourier neural operators. *Advances in neural information processing systems*, 36:47438–47450, 2023a.

Shikun Liu, Tianchun Li, Yongbin Feng, Nhan Tran, Han Zhao, Qiang Qiu, and Pan Li. Structural reweighting improves graph domain adaptation. In *International Conference on Machine Learning*, pp. 21778–21793. PMLR, 2023b.

Shikun Liu, Deyu Zou, Han Zhao, and Pan Li. Pairwise alignment improves graph domain adaptation. *arXiv preprint arXiv:2403.01092*, 2024.

Xiaoyi Liu and Hao Tang. Difffno: Diffusion fourier neural operator. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 150–160. IEEE, 2025.

Ziti Liu, Yang Liu, Xunshi Yan, Wen Liu, Han Nie, Shuaiqi Guo, and Chen-an Zhang. Automatic network structure discovery of physics informed neural networks via knowledge distillation. *Nature Communications*, 16(1):9558, 2025.

Yulong Mao, Kaiyu Huang, Changhao Guan, Ganglin Bao, Fengran Mo, and Jinan Xu. Dora: Enhancing parameter-efficient fine-tuning with dynamic rank distribution. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11662–11675, 2024.

Zhanfeng Mo, Long-Kai Huang, and Sinno Jialin Pan. Parameter and memory efficient pretraining via low-rank riemannian optimization. In *International Conference on Learning Representations*, volume 2025, pp. 7907–7936, 2025.

Jan Muhammad, Ali H Tedjani, Ejaz Hussain, and Usman Younas. Exploring the exact solutions to the nonlinear systems with neural networks method. *Scientific Reports*, 15(1):36818, 2025.

Reza Olfati-Saber, J Alex Fax, and Richard M Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

Xiang Pan, Tianyu Zhao, Minghua Chen, and Shengyu Zhang. Deepopf: A deep neural network approach for security-constrained dc optimal power flow. *IEEE Transactions on Power Systems*, 36(3):1725–1735, 2020.

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

Michael Patriksson. *The traffic assignment problem: models and methods*. Courier Dover Publications, 2015.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

Raksha Ramakrishna and Anna Scaglione. Grid-graph signal processing (grid-gsp): A graph signal processing framework for the power grid. *IEEE Transactions on Signal Processing*, 69:2725–2739, 2021.

Ana Rivera Him, Anvita Bhagavathula, Alvaro Carbonero, and Priya Donti. PF Δ: A benchmark dataset for power flow under load, generation, and topology variations. *Advances in Neural Information Processing Systems*, 38, 2026.

Nafis Sadik and Mohammad Rasoul Narimani. Impact of higher-order structures in power grids' graph on line outage distribution factor. In *2023 North American Power Symposium (NAPS)*, pp. 1–6. IEEE, 2023.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pp. 8459–8468. Pmlr, 2020.

Steffen Schotthöfer, Tianbai Xiao, Martin Frank, and Cory D Hauck. Structure preserving neural networks: A case study in the entropy closure of the boltzmann equation. In *International Conference on Machine Learning*, pp. 19406–19433, 2022.

Ahsan Shehzad, Feng Xia, Shagufta Abid, Ciyuan Peng, Shuo Yu, Dongyu Zhang, and Karin Verspoor. Graph transformers: A survey. *IEEE transactions on neural networks and learning systems*, 2026.

Jack Sherman and Winifred J Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950.

Shriram Srinivasan, Nishant Panda, and Kaarthik Sundar. On the existence of steady-state solutions to the equations governing fluid flow in networks. *IEEE control systems letters*, 8:646–651, 2024.

Steven H Strogatz. Exploring complex networks. *nature*, 410(6825):268–276, 2001.

Kai Sun. *Cascading Failures in Power Grids*. Springer, 2024.

Byron Tasseff, Carleton Coffrin, Russell Bent, Kaarthik Sundar, and Anatoly Zlotnik. Natural gas maximal load delivery for multi-contingency analysis. *Computers & Chemical Engineering*, 168: 108032, 2022.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. Dylora: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 3274–3287, 2023.

Arjan van der Schaft. Modeling of physical network systems. *Systems & Control Letters*, 101: 21–27, 2017.

Arjan J Van der Schaft and Bernhard M Maschke. Port-hamiltonian systems on graphs. *SIAM Journal on Control and Optimization*, 51(2):906–937, 2013.

Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

Hongyi Wang, Scott Sievert, Shengchao Liu, Zachary Charles, Dimitris Papailiopoulos, and Stephen Wright. Atomo: Communication-efficient learning via atomic sparsification. *Advances in neural information processing systems*, 31, 2018.

Hongyi Wang, Saurabh Agarwal, Yoshiki Tanaka, Eric Xing, Dimitris Papailiopoulos, et al. Cuttlefish: Low-rank model training without all the tuning. *Proceedings of Machine Learning and Systems*, 5:578–605, 2023.

Shizheng Wen, Arsh Kumbhat, Levi Lingsch, Sepehr Mousavi, Yizhou Zhao, Praveen Chandrashekar, and Siddhartha Mishra. Geometry aware operator transformer as an efficient and accurate neural surrogate for pdes on arbitrary domains. *Advances in Neural Information Processing Systems*, 38:155423–155501, 2026.

Max A Woodbury. *Inverting modified matrices*. Department of Statistics, Princeton University, 1950.

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.

Mei Yang, Junyong Liu, Gao Qiu, Tingjian Liu, and Zhiyuan Tang. Ac optimal power flows: Combining lagrangian dual and physics-guided graph neutral networks. 2022.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023.

Zhen Zhang, Meihan Liu, Anhui Wang, Hongyang Chen, Zhao Li, Jiajun Bu, and Bingsheng He. Collaborate to adapt: Source-free graph domain adaptation via bi-directional adaptation. In *Proceedings of the ACM Web Conference 2024*, pp. 664–675, 2024.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024.

Wentao Zhu, Mingyu Han, Jovica V Milanović, and Peter Crossley. Methodology for reliability assessment of smart grid considering risk of failure of communication architecture. *IEEE Transactions on Smart Grid*, 11(5):4358–4365, 2020.

Ray Daniel Zimmerman, Carlos Edmundo Murillo-Sánchez, and Robert John Thomas. MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education. *IEEE Transactions on Power Systems*, 26(1):12–19, 2010.

APPENDIX

A EXTENDED REVIEW OF RELATED WORK

This section provides an extended review of related works from two perspectives. We first review learning-based approaches for handling topology changes in neural solvers, and then discuss existing methods that exploit low-rank structure and clarify how their use of low rank differs from RecastNet.

A.1 LEARNING-BASED APPROACHES TO TOPOLOGY CHANGES

Learning neural solvers on graphs. A central challenge in neural solvers for networked systems is that the solution depends not only on operating conditions but also on the network topology. A common approach is therefore to explicitly encode graph connectivity into neural solvers, allowing the learned solution mapping to reflect interactions between connected components.

Graph neural networks (GNNs) encode network connectivity through message passing and have been widely used for learning physical interactions and solving networked systems, including Deep Statistical Solvers for networked system equations (Donon et al., 2020c) and MeshGraphNets for learning dynamics over mesh connectivity (Pfaff et al., 2021). Physics-informed graph models further incorporate physical relations into GNNs. Physics-informed GNNs have been developed for solving PDEs by combining graph representations with physical equations and numerical discretizations Yang et al. (2022); Ramakrishna & Scaglione (2021). In power systems, Kirchhoff's laws have been used as physical constraints for training GNN-based neural solvers (Donon et al., 2020a); PowerFlowNet combines network-based message passing with power flow representations through its encoder and PowerFlowConv layers (Lin et al., 2024). Graph transformers provide more expressive architectures for learning neural solvers over graphs. Spectral attention, for example, incorporates graph spectral information into transformer architectures (Kreuzer et al., 2021); broader graph transformer approaches are reviewed in (Shehzad et al., 2026).

Another line of neural solvers is based on operator learning, which learns mappings between function spaces rather than finite-dimensional parameter-to-solution mappings. Recent neural operators have been extended to varying domains and geometries. For example, domain-agnostic Fourier neural operators enable operator learning across different domain representations (Liu et al., 2023a); geometry-aware operator transformers incorporate geometric information to learn solution operators on arbitrary domains (Wen et al., 2026).

Topology changes can also be introduced explicitly into the learned representation. LEAP (Donon et al., 2020b) uses structural variables describing topology changes to modulate latent representations, allowing improved generalization to rare or unseen topologies.

Neural solvers under topology shifts. In practical applications, the graph topology distribution at test time may differ from that during training. Graph domain adaptation methods address such graph distribution shifts by adapting learned graph representations across different distributions. StruRW (Liu et al., 2023b) reweights the influence of neighboring nodes to align structural information between source and target graphs. Pair-Align (Liu et al., 2024) further considers both structural and label shifts through pairwise alignment, where edge weights are used to recalibrate neighborhood aggregation and adjust the learning objective.

More recent studies consider test-time adaptation, where the model is transferred to a target domain at test time without re-accessing the source domain. GTrans (Jin et al., 2022) optimizes node attributes and graph connectivity at inference time while keeping the pretrained model fixed. GraphCTA (Zhang et al., 2024) performs model adaptation and graph adaptation alternately, updating the model and target graph through contrastive learning. Matcha (Bao et al., 2025) adapts hop-aggregation parameters at test time to restore node representations degraded by graph structure shifts. To account for uncertainty, TSA (Hsu et al., 2025) aligns neighborhood structures at test time through uncertainty-aware neighborhood weighting and adaptive balancing between self-node and aggregated representations.

How RecastNet differs. Despite their different mechanisms, the above approaches handle topology changes through learning: graph-based neural solvers incorporate topology information into the

learned solution mapping, while graph adaptation methods further adapt learned representations, graph structures, or model parameters under topology shifts. In contrast, RecastNet separates neural solver training from topology handling. It trains the neural solver only on a single base topology and keeps it frozen thereafter, while topology changes are represented through changes in the network Laplacian and incorporated analytically through recasting. This enables zero-shot adaptation to unseen topology changes without cross-topology training, retraining, or fine-tuning.

A.2 Low-Rank Adaptation in Machine Learning

One important mechanism in RecastNet is to represent topology changes as low-rank updates to the network Laplacian and use the SMW formula to analytically update its inverse. This use of low-rank updates may appear related to low-rank adaptation methods widely used in machine learning. We therefore clarify their differences below.

Low-rank structure has been widely exploited in machine learning, particularly for parameter-efficient neural network adaptation. Low-Rank Adaptation (LoRA) (Hu et al., 2021) represents parameter updates with a low-rank decomposition and uses this to constrain the fine-tuning process, thereby substantially reducing the number of trainable parameters during fine-tuning. Subsequent variants extend this principle in different ways. AdaLoRA (Zhang et al., 2023) and DoRA (Mao et al., 2024) allocate adaptive parameter budgets through importance-based selection of low-rank components. DyLoRA (Valipour et al., 2023) trains adapters to operate across a range of ranks. VeRA (Kopiczko et al., 2024) further reduces trainable parameters by a single pair of low-rank matrices shared across layers.

Low-rank structure has also been exploited to improve neural network training. CUTTLEFISH (Wang et al., 2023) adaptively determines low-rank parameterizations during training, while ReLoRA (Lialin et al., 2024) repeatedly applies low-rank updates to train full-rank neural network models. In distributed training, ATOMO (Wang et al., 2018) and PowerSGD (Vogels et al., 2019) reduce communication costs by compressing gradient matrices using low-rank representations. Targeting at memory efficiency, GaLore (Zhao et al., 2024) projects gradients into low-rank subspaces, and LORO (Mo et al., 2025) performs optimization directly over low-rank parameter manifolds.

How RecastNet differs. The above methods exploit or impose low-rank structure on neural network quantities, such as parameter updates, weight matrices, or gradients, primarily to improve the efficiency of model training, fine-tuning, or adaptation. In contrast, the low-rank structure in RecastNet arises from *topology changes in the underlying physical network* rather than from the neural network itself. Specifically, modifying a small number of links induces a low-rank update to the governing network Laplacian:

$$L_{\mathcal{G}} = L_{\mathcal{G}_0} + B_\Delta \Delta B_\Delta^\top, \qquad \operatorname{rank}(B_\Delta \Delta B_\Delta^\top) \ll n \tag{11}$$

which RecastNet exploits through the SMW formula while keeping the neural solver unchanged.

This reflects a fundamentally different view of model adaptation. Low-rank parameter-efficient methods such as LoRA learn a compact neural network update when the target task or configuration changes. In contrast, RecastNet focuses on physical topology changes; it does not learn how the neural network should adapt to this change, but analytically propagates its effect to the pretrained and frozen base-topology neural solver. In this way, topology adaptation is moved from neural network training to analytical recasting.

B Low-Rank Topology Changes Can Induce Large Solution Changes

This appendix clarifies an important distinction: a low-rank topology change means that only a small part of the network structure is modified, but does not imply that the resulting solution change is small. In fact, even a rank-one topology change can substantially alter the system solution.

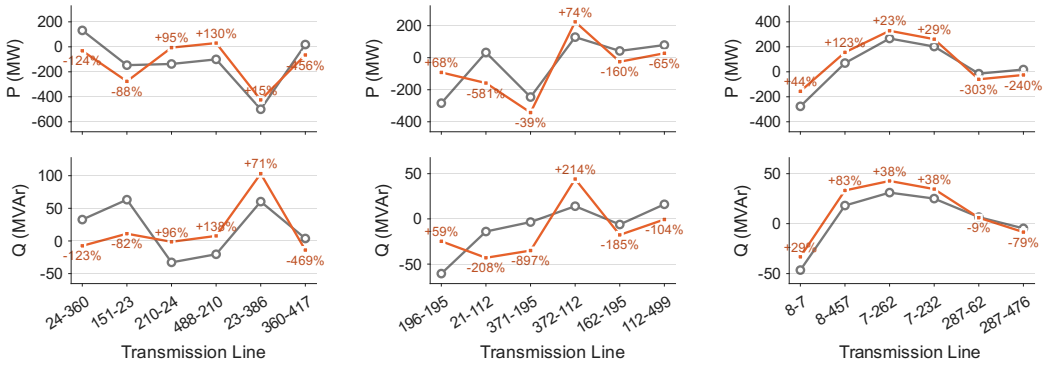
To see this directly, consider a linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ and a rank-one topology change $\boldsymbol{A}' = \boldsymbol{A} + \boldsymbol{u}\delta a\boldsymbol{v}^\top$. The Sherman–Morrison formula gives:

$$\begin{aligned}\boldsymbol{x}' &= (\boldsymbol{A} + \boldsymbol{u}\delta a\boldsymbol{v}^\top)^{-1}\boldsymbol{b} = \left(\boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{u}\left((\delta a)^{-1} + \boldsymbol{v}^\top\boldsymbol{A}^{-1}\boldsymbol{u}\right)^{-1}\boldsymbol{v}^\top\boldsymbol{A}^{-1}\right)\boldsymbol{b} \\ &= \boldsymbol{x} - \frac{\delta a\,\boldsymbol{v}^\top\boldsymbol{x}}{1 + \delta a\,\boldsymbol{v}^\top\boldsymbol{A}^{-1}\boldsymbol{u}}\boldsymbol{A}^{-1}\boldsymbol{u}\end{aligned} \tag{12}$$

Hence, the topology-induced solution change is exactly:

$$\Delta\boldsymbol{x} = \boldsymbol{x}' - \boldsymbol{x} = -\frac{\delta a\,\boldsymbol{v}^\top\boldsymbol{x}}{1 + \delta a\,\boldsymbol{v}^\top\boldsymbol{A}^{-1}\boldsymbol{u}}\boldsymbol{A}^{-1}\boldsymbol{u} \tag{13}$$

Eq. (13) directly shows that the rank of the topology change does not bound the magnitude of the solution change $\Delta\boldsymbol{x}$. For example, if $1 + \delta a\,\boldsymbol{v}^\top\boldsymbol{A}^{-1}\boldsymbol{u}$ approaches zero, $\|\Delta\boldsymbol{x}\|$ can become arbitrarily large, even though the topology change remains rank one. Figure 7 illustrates this effect in the South Carolina 500-bus AC power flow, where a single-line outage can cause substantial power flow redistribution, with several line flows changing by more than 100%.



(a) Outage of line 24–23. (b) Outage of line 21–196. (c) Outage of line 8–287.

Figure 7: **Impact of single-line outages on AC power flow** in the South Carolina 500-bus system. Results show active (P) and reactive (Q) power flows before (gray) and after (orange) the outage, with percentages representing the relative changes.

C Proofs and Other Technical Details

This appendix provides proofs and further technical discussions supporting the methodology introduced in Section 3.

C.1 Discussions for Solution-to-Inverse Recasting

We begin with the proof of the solution-to-inverse recasting introduced in Section 3.1.

C.1.1 Proof of Theorem 1

Proof. Recall the base-topology linear network equation $\boldsymbol{L}_{\mathcal{G}_0}(\boldsymbol{\xi})\boldsymbol{x}_0(\boldsymbol{\xi}) = \boldsymbol{b}(\boldsymbol{\xi})$. Differentiating this network equation with respect to ξ_j gives:

$$\frac{\partial\boldsymbol{L}_{\mathcal{G}_0}}{\partial\xi_j}\boldsymbol{x}_0 + \boldsymbol{L}_{\mathcal{G}_0}\frac{\partial\boldsymbol{x}_0}{\partial\xi_j} = \frac{\partial\boldsymbol{b}}{\partial\xi_j} \tag{14}$$

Since the grounded weighted graph Laplacian $\boldsymbol{L}_{\mathcal{G}_0}$ is nonsingular, Eq. (14) yields:

$$\frac{\partial\boldsymbol{x}_0}{\partial\xi_j} = \boldsymbol{L}_{\mathcal{G}_0}^{-1}\left(\frac{\partial\boldsymbol{b}}{\partial\xi_j} - \frac{\partial\boldsymbol{L}_{\mathcal{G}_0}}{\partial\xi_j}\boldsymbol{x}_0\right) := \boldsymbol{L}_{\mathcal{G}_0}^{-1}\boldsymbol{s}_j^* \tag{15}$$

Define $\boldsymbol{S}_0^* := [\boldsymbol{s}_1^*, \ldots, \boldsymbol{s}_p^*]$. Stacking the identities in Eq. (15) column-wise for $j = 1, \ldots, p$ yields:

$$\frac{\partial \boldsymbol{x}_0}{\partial \boldsymbol{\xi}} = \boldsymbol{L}_{\mathcal{G}_0}^{-1} \boldsymbol{S}_0^* \tag{16}$$

If $\boldsymbol{S}_0^*$ has full row rank, right-multiplying Eq. (16) by the pseudoinverse $(\boldsymbol{S}_0^*)^\dagger$ gives the inverse of the network Laplacian:

$$\frac{\partial \boldsymbol{x}_0}{\partial \boldsymbol{\xi}} (\boldsymbol{S}_0^*)^\dagger = \boldsymbol{L}_{\mathcal{G}_0}^{-1} \boldsymbol{S}_0^* (\boldsymbol{S}_0^*)^\dagger = \boldsymbol{L}_{\mathcal{G}_0}^{-1} \boldsymbol{I} = \boldsymbol{L}_{\mathcal{G}_0}^{-1} \tag{17}$$

In practice, a neural solver approximates the true solution, i.e., $\mathcal{N}_0(\boldsymbol{\xi}; \boldsymbol{\theta}) \approx \boldsymbol{x}_0(\boldsymbol{\xi})$. Accordingly, the input–output Jacobian $\boldsymbol{J}_{\mathcal{N}_0} = \partial \mathcal{N}_0 / \partial \boldsymbol{\xi}$ and the sensitivity matrix $\boldsymbol{S}_{\mathcal{N}_0}$ serve as the neural counterparts of $\partial \boldsymbol{x}_0 / \partial \boldsymbol{\xi}$ and $\boldsymbol{S}_0^*$, respectively. This yields the recast inverse as stated in Theorem 1:

$$\widehat{\boldsymbol{L}}_{\mathcal{G}_0}^{-1} = \boldsymbol{J}_{\mathcal{N}_0} \boldsymbol{S}_{\mathcal{N}_0}^\dagger \approx \boldsymbol{L}_{\mathcal{G}_0}^{-1} \tag{18}$$

The approximation error of this recast inverse is characterized in Appendix C.1.3. □

C.1.2 Rank Condition and Recasting under Rank Deficiency

We discuss the rank condition in Theorem 1. In the networked system equation of Eq. (2), $\boldsymbol{\xi}$ collects three types of system conditions: nodal, link, and shared system conditions. In particular, the nodal conditions ξ_i provide independent variations of the nodal injections across the network, making $\boldsymbol{S}_{\mathcal{N}_0}$ typically full row rank for this class of network systems. Yet, for theoretical completeness, we also discuss the rank-deficient case below.

When $\boldsymbol{S}_{\mathcal{N}_0}$ does not have full row rank, we generally cannot recover the complete inverse $\boldsymbol{L}_{\mathcal{G}_0}^{-1}$ from the available Jacobian relations. However, this does not necessarily prevent topology recasting, since only the inverse actions associated with the topology change are needed.

Consider a rank-deficient case with $\mathrm{rank}(\boldsymbol{S}_0^*) = q < n$, where $\boldsymbol{S}_0^*$ is defined in the Appendix C.1.1. Denote $\boldsymbol{P}_S = \boldsymbol{S}_0^* (\boldsymbol{S}_0^*)^\dagger$ as the orthogonal projector onto the column space of $\boldsymbol{S}_0^*$. From Eq. (16), we obtain the projected inverse:

$$\frac{\partial \boldsymbol{x}_0}{\partial \boldsymbol{\xi}} (\boldsymbol{S}_0^*)^\dagger = \boldsymbol{L}_{\mathcal{G}_0}^{-1} \boldsymbol{S}_0^* (\boldsymbol{S}_0^*)^\dagger = \boldsymbol{L}_{\mathcal{G}_0}^{-1} \boldsymbol{P}_S \tag{19}$$

Although the complete inverse is not recovered, its action is preserved on the column space of $\boldsymbol{S}_0^*$.

Next, consider a low-rank topology change $\boldsymbol{L}_{\mathcal{G}} = \boldsymbol{L}_{\mathcal{G}_0} + \boldsymbol{B}_\Delta \boldsymbol{\Delta} \boldsymbol{B}_\Delta^\top$. The SMW update requires the recast inverse along the topology-change directions, i.e., $\boldsymbol{L}_{\mathcal{G}_0}^{-1} \boldsymbol{B}_\Delta$ as shown in Eq. (3). Denote $\mathrm{col}(\cdot)$ as the column space of a matrix. If $\mathrm{col}(\boldsymbol{B}_\Delta) \subseteq \mathrm{col}(\boldsymbol{S}_0^*)$, then:

$$\boldsymbol{P}_S \boldsymbol{B}_\Delta = \boldsymbol{B}_\Delta \quad \Longrightarrow \quad \boldsymbol{L}_{\mathcal{G}_0}^{-1} \boldsymbol{B}_\Delta = \boldsymbol{L}_{\mathcal{G}_0}^{-1} \boldsymbol{P}_S \boldsymbol{B}_\Delta = \frac{\partial \boldsymbol{x}_0}{\partial \boldsymbol{\xi}} (\boldsymbol{S}_0^*)^\dagger \boldsymbol{B}_\Delta \tag{20}$$

Hence, even in the rank-deficient case, the inverse action required by a particular topology change can still be recovered exactly if its change directions lie in $\mathrm{col}(\boldsymbol{S}_0^*)$. This proof directly connects to solution recasting in Sections 3.2 and 3.3.

C.1.3 Characterizing the Inverse-Recasting Error

We next quantify how the base-topology prediction error affects the recast inverse. Define $\boldsymbol{\varepsilon}_0(\boldsymbol{\xi}) := \mathcal{N}_0(\boldsymbol{\xi}; \boldsymbol{\theta}) - \boldsymbol{x}_0(\boldsymbol{\xi})$ as the neural solver prediction error, and define $\boldsymbol{E}_{\mathcal{G}_0}(\boldsymbol{\varepsilon}_0) := [(\partial \boldsymbol{L}_{\mathcal{G}_0} / \partial \xi_j) \boldsymbol{\varepsilon}_0]_{j=1}^p$. We prove that the inverse-recasting error is bounded by:

$$\left\| \widehat{\boldsymbol{L}}_{\mathcal{G}_0}^{-1} - \boldsymbol{L}_{\mathcal{G}_0}^{-1} \right\|_2 \leq \frac{\|\partial \boldsymbol{\varepsilon}_0 / \partial \boldsymbol{\xi}\|_2 + \left\| \boldsymbol{L}_{\mathcal{G}_0}^{-1} \right\|_2 \|\boldsymbol{E}_{\mathcal{G}_0}(\boldsymbol{\varepsilon}_0)\|_2}{\sigma_{\min}(\boldsymbol{S}_{\mathcal{N}_0})} \tag{21}$$

Proof. Using the defined neural solver error $\boldsymbol{\varepsilon}_0(\boldsymbol{\xi}) := \mathcal{N}_0(\boldsymbol{\xi}; \boldsymbol{\theta}) - \boldsymbol{x}_0(\boldsymbol{\xi})$, the input–output Jacobian of the neural solver can be written as:

$$\boldsymbol{J}_{\mathcal{N}_0} = \frac{\partial \boldsymbol{x}_0}{\partial \boldsymbol{\xi}} + \frac{\partial \boldsymbol{\varepsilon}_0}{\partial \boldsymbol{\xi}} \tag{22}$$

According to Eq. (16), the first term $\partial \boldsymbol{x}_0/\partial \boldsymbol{\xi} = \boldsymbol{L}_{\mathcal{G}_0}^{-1}\boldsymbol{S}_0^*$. Substituting $\boldsymbol{x}_0 = \mathcal{N}_0 - \boldsymbol{\varepsilon}_0$ into the definition of $\boldsymbol{S}_0^*$ in Eq. (15) yields:

$$\boldsymbol{S}_0^* = [\frac{\partial \boldsymbol{b}}{\partial \xi_j} - \frac{\partial \boldsymbol{L}_{\mathcal{G}_0}}{\partial \xi_j}\boldsymbol{x}_0]_{j=1}^p = [\frac{\partial \boldsymbol{b}}{\partial \xi_j} - \frac{\partial \boldsymbol{L}_{\mathcal{G}_0}}{\partial \xi_j}(\mathcal{N}_0 - \boldsymbol{\varepsilon}_0)]_{j=1}^p = \boldsymbol{S}_{\mathcal{N}_0} + \boldsymbol{E}_{\mathcal{G}_0}(\boldsymbol{\varepsilon}_0) \tag{23}$$

Substituting this into Eq. (22) and right-multiplying by $\boldsymbol{S}_{\mathcal{N}_0}^\dagger$ gives:

$$\begin{aligned} \boldsymbol{J}_{\mathcal{N}_0}\boldsymbol{S}_{\mathcal{N}_0}^\dagger &= \boldsymbol{L}_{\mathcal{G}_0}^{-1} + \left(\frac{\partial \boldsymbol{\varepsilon}_0}{\partial \boldsymbol{\xi}} + \boldsymbol{L}_{\mathcal{G}_0}^{-1}\boldsymbol{E}_{\mathcal{G}_0}(\boldsymbol{\varepsilon}_0)\right)\boldsymbol{S}_{\mathcal{N}_0}^\dagger \\ \xRightarrow{\text{Eq. (18)}} \quad \widehat{\boldsymbol{L}}_{\mathcal{G}_0}^{-1} - \boldsymbol{L}_{\mathcal{G}_0}^{-1} &= \left(\frac{\partial \boldsymbol{\varepsilon}_0}{\partial \boldsymbol{\xi}} + \boldsymbol{L}_{\mathcal{G}_0}^{-1}\boldsymbol{E}_{\mathcal{G}_0}(\boldsymbol{\varepsilon}_0)\right)\boldsymbol{S}_{\mathcal{N}_0}^\dagger \end{aligned} \tag{24}$$

Taking the spectral norm of Eq. (24) gives:

$$\begin{aligned} \left\|\widehat{\boldsymbol{L}}_{\mathcal{G}_0}^{-1} - \boldsymbol{L}_{\mathcal{G}_0}^{-1}\right\|_2 &\le \left(\|\partial \boldsymbol{\varepsilon}_0/\partial \boldsymbol{\xi}\|_2 + \left\|\boldsymbol{L}_{\mathcal{G}_0}^{-1}\right\|_2 \|\boldsymbol{E}_{\mathcal{G}_0}(\boldsymbol{\varepsilon}_0)\|_2\right)\left\|\boldsymbol{S}_{\mathcal{N}_0}^\dagger\right\|_2 \\ &= \frac{\|\partial \boldsymbol{\varepsilon}_0/\partial \boldsymbol{\xi}\|_2 + \left\|\boldsymbol{L}_{\mathcal{G}_0}^{-1}\right\|_2 \|\boldsymbol{E}_{\mathcal{G}_0}(\boldsymbol{\varepsilon}_0)\|_2}{\sigma_{\min}(\boldsymbol{S}_{\mathcal{N}_0})} \end{aligned} \tag{25}$$

This concludes the proof. □

Interpretation of the error bound. The error bound shows that the inverse-recasting error depends on both the prediction accuracy and the input–output Jacobian accuracy of the base-topology neural solver. The sensitivity matrix $\boldsymbol{S}_{\mathcal{N}_0}$ governs how these errors are amplified. As $\boldsymbol{\varepsilon}_0$ and $\partial \boldsymbol{\varepsilon}_0/\partial \boldsymbol{\xi}$ approach zero, the recast inverse converges to $\boldsymbol{L}_{\mathcal{G}_0}^{-1}$.

C.2 Discussions for Linear Solution Recasting

This appendix provides the derivation and additional technical details for the linear solution recasting in Section 3.2.

C.2.1 Proof of Proposition 2

Proof. We first establish the structure of the topology-specific sensitivity matrix $\boldsymbol{S}_{\mathcal{N}_0,\Delta}$ used in Proposition 2, and then prove statements (i) and (ii).

Topology-specific sensitivity structure. Recall from Theorem 1 that, for each condition ξ_j, the corresponding sensitivity column is:

$$\boldsymbol{s}_j = \frac{\partial \boldsymbol{b}}{\partial \xi_j} - \frac{\partial \boldsymbol{L}_{\mathcal{G}_0}}{\partial \xi_j}\mathcal{N}_0 \tag{26}$$

Following the partition $\boldsymbol{\xi} = (\boldsymbol{\xi}_{\text{loc}}, \boldsymbol{\xi}_{\text{sh}}, \boldsymbol{\xi}_L)$, we consider the corresponding blocks $\boldsymbol{S}_{\text{loc}}$, $\boldsymbol{S}_{\text{sh}}$, and $\boldsymbol{S}_L$ separately.

For the local-condition block $\boldsymbol{S}_{\text{loc}}$, $\boldsymbol{L}_{\mathcal{G}_0}$ is independent of ξ_i, while $\partial \boldsymbol{b}/\partial \xi_i$ has a nonzero entry only at node i. Therefore, we have:

$$\boldsymbol{S}_{\text{loc}} = \text{diag}(s_1, \ldots, s_n), \qquad s_i = \frac{\partial b_i}{\partial \xi_i} \tag{27}$$

For the shared-condition block $\boldsymbol{S}_{\text{sh}}$, the number of shared system conditions is typically much smaller than the number of nodes (i.e., $p_{\text{sh}} \ll n$). Thus, $\boldsymbol{S}_{\text{sh}}$ is a skinny matrix.

We finally consider the link-condition block $\boldsymbol{S}_L$. For a link e involved in the topology change: denote its incidence vector as $\boldsymbol{B}_{:,e}$ and its weight as w_e, where w_e is parameterized by the link condition $\xi_{L,e}$. The link condition $\xi_{L,e}$ affects the linear system through the Laplacian $\boldsymbol{L}_{\mathcal{G}_0} = \boldsymbol{B}\boldsymbol{W}\boldsymbol{B}^\top$. Differentiating the Laplacian with respect to $\xi_{L,e}$ yields the corresponding sensitivity column:

$$\frac{\partial \boldsymbol{L}_{\mathcal{G}_0}}{\partial \xi_{L,e}} = \frac{\partial w_e}{\partial \xi_{L,e}}\boldsymbol{B}_{:,e}\boldsymbol{B}_{:,e}^\top \Rightarrow \boldsymbol{s}_{L,e} = -\frac{\partial w_e}{\partial \xi_{L,e}}\left(\boldsymbol{B}_{:,e}^\top \mathcal{N}_0\right)\boldsymbol{B}_{:,e} \tag{28}$$

Thus, the link-sensitivity column $\boldsymbol{s}_{L,e}$ is aligned with the incidence direction $\boldsymbol{B}_{:,e}$ of link e. This motivates us to retain only the sensitivity columns associated with the affected links. Consider a rank-r topology change where $r \ll n$. Collecting the r link-sensitivity columns associated with these affected links, we construct $\boldsymbol{S}_{L,\Delta} \in \mathbb{R}^{n\times r}$, which is skinny.

We then combine the shared-condition block and retained link-condition block as $\boldsymbol{S}_{\mathrm{sh},\Delta} := [\boldsymbol{S}_{\mathrm{sh}},\, \boldsymbol{S}_{L,\Delta}] \in \mathbb{R}^{n\times(p_{\mathrm{sh}}+r)}$. The topology-specific sensitivity matrix is finally formed as:

$$\boldsymbol{S}_{\mathcal{N}_0,\Delta} = [\boldsymbol{S}_{\mathrm{loc}},\, \boldsymbol{S}_{\mathrm{sh},\Delta}] \tag{29}$$

which consists of a nonsingular diagonal block and a skinny block.

Proof of statement (i)**.** Eq. (6) shows that the base-topology Laplacian inverse enters the new-topology solution through $\boldsymbol{H}_\Delta = \boldsymbol{L}_{\mathcal{G}_0}^{-1}\boldsymbol{B}_\Delta$. Applying the solution-to-inverse recasting in Eq. (4) to the topology-specific sensitivities yields:

$$\widehat{\boldsymbol{H}}_\Delta := \boldsymbol{J}_{\mathcal{N}_0,\Delta}\boldsymbol{S}_{\mathcal{N}_0,\Delta}^{\dagger}\boldsymbol{B}_\Delta \tag{30}$$

Substituting $\widehat{\boldsymbol{H}}_\Delta$ for $\boldsymbol{H}_\Delta$ in Eq. (6) gives:

$$\widehat{\boldsymbol{x}}_{\mathcal{G}} = \mathcal{N}_0(\boldsymbol{\xi};\boldsymbol{\theta}) - \widehat{\boldsymbol{H}}_\Delta\left(\boldsymbol{\Delta}^{-1} + \boldsymbol{B}_\Delta^\top\widehat{\boldsymbol{H}}_\Delta\right)^{-1}\boldsymbol{B}_\Delta^\top\mathcal{N}_0(\boldsymbol{\xi};\boldsymbol{\theta}) \tag{31}$$

which proves statement (i).

Proof of statement (ii)**.** Evaluating $\widehat{\boldsymbol{H}}_\Delta$ in Eq. (30) appears to require the pseudoinverse $\boldsymbol{S}_{\mathcal{N}_0,\Delta}^{\dagger}$. We show that only its action on $\boldsymbol{B}_\Delta$ is needed. This action can be evaluated through a low-dimensional SMW update.

Since $\boldsymbol{S}_{\mathrm{loc}}$ is nonsingular and $\boldsymbol{S}_{\mathcal{N}_0,\Delta}$ has full row rank, the pseudoinverse required in Eq. (30) can be written as:

$$\boldsymbol{S}_{\mathcal{N}_0,\Delta}^{\dagger}\boldsymbol{B}_\Delta = \boldsymbol{S}_{\mathcal{N}_0,\Delta}^\top\left(\boldsymbol{S}_{\mathcal{N}_0,\Delta}\boldsymbol{S}_{\mathcal{N}_0,\Delta}^\top\right)^{-1}\boldsymbol{B}_\Delta \tag{32}$$

From Eq. (29), the matrix to be inverted in Eq. (32) has the structured form

$$\boldsymbol{S}_{\mathcal{N}_0,\Delta}\boldsymbol{S}_{\mathcal{N}_0,\Delta}^\top = \underbrace{\boldsymbol{S}_{\mathrm{loc}}\boldsymbol{S}_{\mathrm{loc}}^\top}_{\text{Diagonal}} + \underbrace{\boldsymbol{S}_{\mathrm{sh},\Delta}\boldsymbol{S}_{\mathrm{sh},\Delta}^\top}_{\mathrm{rank}\le p_{\mathrm{sh}}+r} \tag{33}$$

Denote $\boldsymbol{D} = \boldsymbol{S}_{\mathrm{loc}}\boldsymbol{S}_{\mathrm{loc}}^\top$, which is diagonal and nonsingular. Then, applying SMW to Eq. (33) yields:

$$\left(\boldsymbol{S}_{\mathcal{N}_0,\Delta}\boldsymbol{S}_{\mathcal{N}_0,\Delta}^\top\right)^{-1} = \boldsymbol{D}^{-1} - \boldsymbol{D}^{-1}\boldsymbol{S}_{\mathrm{sh},\Delta}\left(\boldsymbol{I} + \boldsymbol{S}_{\mathrm{sh},\Delta}^\top\boldsymbol{D}^{-1}\boldsymbol{S}_{\mathrm{sh},\Delta}\right)^{-1}\boldsymbol{S}_{\mathrm{sh},\Delta}^\top\boldsymbol{D}^{-1} \tag{34}$$

The only dense inversion in Eq. (34) is $(\boldsymbol{I} + \boldsymbol{S}_{\mathrm{sh},\Delta}^\top\boldsymbol{D}^{-1}\boldsymbol{S}_{\mathrm{sh},\Delta})^{-1}$, whose dimension is $p_{\mathrm{sh}} + r$. Substituting Eq. (34) into Eq. (32) and Eq. (30), we finally obtain $\widehat{\boldsymbol{H}}_\Delta$ without explicitly constructing either $\boldsymbol{S}_{\mathcal{N}_0,\Delta}^{\dagger}$ or a dense $n \times n$ inverse.

The subsequent solution update in Eq. (6) requires only the additional $r \times r$ SMW inversion. Hence, the dense matrix inversions required by the complete cross-topology solution recasting have dimensions $(p_{\mathrm{sh}} + r) \times (p_{\mathrm{sh}} + r)$ and $r \times r$, neither of them scales with n. This proves statement (ii). □

C.2.2 Batched Implementation

Although the recast solution in Eq. (6) is not a conventional neural network forward pass, its constituent operations admit the same tensorized execution used in standard neural network inference.

Consider a batch of B problem instances indexed by b, with system conditions $\boldsymbol{\xi}^{(b)}$ and topology changes $(\boldsymbol{B}_\Delta^{(b)}, \boldsymbol{\Delta}^{(b)})$. The base predictions $\mathcal{N}_0(\boldsymbol{\xi}^{(b)};\boldsymbol{\theta})$ are evaluated in a single batched forward pass, while the required input–output Jacobian actions are obtained through batched automatic differentiation without explicitly forming a full Jacobian for each instance.

For each instance, the structured evaluation in Eq. (34) involves only diagonal operations and a linear system of dimension $p_{\mathrm{sh}} + r^{(b)}$. Likewise, the final topology correction in Eq. (6) requires a linear

system of dimension $r^{(b)}$. When the instances share the same topology-change rank, these matrices can be stacked and evaluated directly using batched linear-algebra kernels. For mixed ranks, instances can be grouped by rank or padded to the largest rank within a batch. The computational workflow can be summarized as

$$\{\boldsymbol{\xi}^{(b)}, \boldsymbol{B}_\Delta^{(b)}, \boldsymbol{\Delta}^{(b)}\}_{k=1}^B \;\longrightarrow\; \{\mathcal{N}_0^{(b)}, \widehat{\boldsymbol{H}}_\Delta^{(b)}\}_{k=1}^B \;\longrightarrow\; \{\widehat{\boldsymbol{x}}_\mathcal{G}^{(b)}\}_{k=1}^B \tag{35}$$

This is the batched inference setting used in the runtime experiments of Section 4.

C.2.3 Proof of Corollary 3

Proof. Recall that Corollary 3 defines the base-topology prediction error $\boldsymbol{\varepsilon}_0$ and directional inverse-action error $\boldsymbol{E}_{H,\Delta}$, where $\boldsymbol{H}_\Delta = \boldsymbol{L}_{\mathcal{G}_0}^{-1}\boldsymbol{B}_\Delta$. For a rank-$r$ topology change, we denote:

$$\boldsymbol{K}_\Delta := \boldsymbol{\Delta}^{-1} + \boldsymbol{B}_\Delta^\top \boldsymbol{H}_\Delta, \qquad \widehat{\boldsymbol{K}}_\Delta := \boldsymbol{\Delta}^{-1} + \boldsymbol{B}_\Delta^\top \widehat{\boldsymbol{H}}_\Delta, \qquad \widehat{\boldsymbol{K}}_\Delta - \boldsymbol{K}_\Delta = \boldsymbol{B}_\Delta^\top \boldsymbol{E}_{H,\Delta} \tag{36}$$

Using the SMW formula and Eq. (6), the true and recast solutions under the changed topology can be written as:

$$\boldsymbol{x}_\mathcal{G} = \left(\boldsymbol{I} - \boldsymbol{H}_\Delta \boldsymbol{K}_\Delta^{-1} \boldsymbol{B}_\Delta^\top\right) \boldsymbol{x}_0, \qquad \widehat{\boldsymbol{x}}_\mathcal{G} = \left(\boldsymbol{I} - \widehat{\boldsymbol{H}}_\Delta \widehat{\boldsymbol{K}}_\Delta^{-1} \boldsymbol{B}_\Delta^\top\right) (\boldsymbol{x}_0 + \boldsymbol{\varepsilon}_0) \tag{37}$$

We now evaluate the recast-solution error using Eq. (37):

$$\begin{aligned} \widehat{\boldsymbol{x}}_\mathcal{G} - \boldsymbol{x}_\mathcal{G} &= \left(\boldsymbol{I} - \widehat{\boldsymbol{H}}_\Delta \widehat{\boldsymbol{K}}_\Delta^{-1} \boldsymbol{B}_\Delta^\top\right) (\boldsymbol{x}_0 + \boldsymbol{\varepsilon}_0) - \left(\boldsymbol{I} - \boldsymbol{H}_\Delta \boldsymbol{K}_\Delta^{-1} \boldsymbol{B}_\Delta^\top\right) \boldsymbol{x}_0 \\ &= \left(\boldsymbol{I} - \widehat{\boldsymbol{H}}_\Delta \widehat{\boldsymbol{K}}_\Delta^{-1} \boldsymbol{B}_\Delta^\top\right) \boldsymbol{\varepsilon}_0 + \left(\boldsymbol{H}_\Delta \boldsymbol{K}_\Delta^{-1} - \widehat{\boldsymbol{H}}_\Delta \widehat{\boldsymbol{K}}_\Delta^{-1}\right) \boldsymbol{B}_\Delta^\top \boldsymbol{x}_0 \end{aligned} \tag{38}$$

We can further derive $\boldsymbol{H}_\Delta \boldsymbol{K}_\Delta^{-1} - \widehat{\boldsymbol{H}}_\Delta \widehat{\boldsymbol{K}}_\Delta^{-1} = \left(\boldsymbol{H}_\Delta \boldsymbol{K}_\Delta^{-1} \boldsymbol{B}_\Delta^\top - I\right) \boldsymbol{E}_{H,\Delta} \widehat{\boldsymbol{K}}_\Delta^{-1}$. Substituting this relation into Eq. (38) yields the bound of the recast-solution error:

$$\begin{aligned} \|\widehat{\boldsymbol{x}}_\mathcal{G} - \boldsymbol{x}_\mathcal{G}\|_2 &= \| \left(\boldsymbol{I} - \widehat{\boldsymbol{H}}_\Delta \widehat{\boldsymbol{K}}_\Delta^{-1} \boldsymbol{B}_\Delta^\top\right) \boldsymbol{\varepsilon}_0 - \left(\boldsymbol{I} - \boldsymbol{H}_\Delta \boldsymbol{K}_\Delta^{-1} \boldsymbol{B}_\Delta^\top\right) \boldsymbol{E}_{H,\Delta} \widehat{\boldsymbol{K}}_\Delta^{-1} \boldsymbol{B}_\Delta^\top \boldsymbol{x}_0 \|_2 \\ &\le c_{0,\Delta} \|\boldsymbol{\varepsilon}_0\|_2 + c_{H,\Delta} \|\boldsymbol{E}_{H,\Delta}\|_2 := \mathcal{E}_\Delta \end{aligned} \tag{39}$$

Here, $c_{0,\Delta}$ and $c_{H,\Delta}$ are the topology-dependent error-propagation coefficients defined in Corollary 3, whose upper bounds are given as:

$$c_{0,\Delta} \le \left\| \boldsymbol{I} - \widehat{\boldsymbol{H}}_\Delta \widehat{\boldsymbol{K}}_\Delta^{-1} \boldsymbol{B}_\Delta^\top \right\|_2, \quad c_{H,\Delta} \le \left\| \boldsymbol{I} - \boldsymbol{H}_\Delta \boldsymbol{K}_\Delta^{-1} \boldsymbol{B}_\Delta^\top \right\|_2 \left\| \widehat{\boldsymbol{K}}_\Delta^{-1} \boldsymbol{B}_\Delta^\top \boldsymbol{x}_0 \right\|_2 \tag{40}$$

Consequently, $\mathcal{E}_\Delta \le \epsilon \|\boldsymbol{x}_\mathcal{G}\|_2$ directly implies $\|\widehat{\boldsymbol{x}}_\mathcal{G} - \boldsymbol{x}_\mathcal{G}\|_2 / \|\boldsymbol{x}_\mathcal{G}\|_2 \le \epsilon$, which is the ϵ-recastability condition in Eq. (7). □

C.3 Discussions for Nonlinear Solution Recasting

This appendix provides the derivation and additional technical details for the nonlinear solution recasting in Section 3.3.

C.3.1 Derivation of Dual-RecastNet

Topology-induced solution change. Recall the nonlinear system under the base and changed topologies:

$$\boldsymbol{L}_{\mathcal{G}_0} \boldsymbol{x}_0 = \boldsymbol{f}(\boldsymbol{x}_0, \boldsymbol{\xi}), \qquad \boldsymbol{L}_\mathcal{G} \boldsymbol{x}_\mathcal{G} = \boldsymbol{f}(\boldsymbol{x}_\mathcal{G}, \boldsymbol{\xi}) \tag{41}$$

where $\boldsymbol{L}_\mathcal{G} = \boldsymbol{L}_{\mathcal{G}_0} + \Delta \boldsymbol{L}$. Denote $\Delta \boldsymbol{x}_\mathcal{G} := \boldsymbol{x}_\mathcal{G} - \boldsymbol{x}_0$. A first-order expansion of the nonlinear relation around $\boldsymbol{x}_0$ gives:

$$\boldsymbol{f}(\boldsymbol{x}_\mathcal{G}, \boldsymbol{\xi}) \approx \boldsymbol{f}(\boldsymbol{x}_0, \boldsymbol{\xi}) + \boldsymbol{D}_0 \Delta \boldsymbol{x}_\mathcal{G} \tag{42}$$

where $\boldsymbol{D}_0 := \partial \boldsymbol{f} / \partial \boldsymbol{x}|_{\boldsymbol{x}=\boldsymbol{x}_0}$. Since each local relation f_i depends only on its corresponding state x_i, $\boldsymbol{D}_0$ is diagonal. Substituting $\boldsymbol{x}_\mathcal{G} = \boldsymbol{x}_0 + \Delta \boldsymbol{x}_\mathcal{G}$ and Eq. (42) into Eq. (41) yields

$$\boldsymbol{L}_\mathcal{G} \Delta \boldsymbol{x}_\mathcal{G} \approx -\Delta \boldsymbol{L}\, \boldsymbol{x}_0 + \boldsymbol{D}_0 \Delta \boldsymbol{x}_\mathcal{G} \quad \Longrightarrow \quad \Delta \boldsymbol{x}_\mathcal{G} \approx \boldsymbol{L}_\mathcal{G}^{-1} \left(-\Delta \boldsymbol{L}\, \boldsymbol{x}_0 + \boldsymbol{D}_0 \Delta \boldsymbol{x}_\mathcal{G}\right) \tag{43}$$

which gives Eq. (9).

Auxiliary neural solver. Eq. (43) requires the network inverse $\boldsymbol{L}_{\mathcal{G}}^{-1}$, which can be obtained from $\boldsymbol{L}_{\mathcal{G}_0}^{-1}$ through the SMW formula in Eq. (3). However, unlike the linear case in Section 3.2, the base-topology neural solver $\mathcal{N}_0^{\mathrm{sol}}$ learns the nonlinear solution mapping $\boldsymbol{\xi} \mapsto \boldsymbol{x}_0$ in Eq. (2b), from which $\boldsymbol{L}_{\mathcal{G}_0}^{-1}$ cannot be directly recast. We therefore introduce an auxiliary neural solver for the fixed relation $\boldsymbol{L}_{\mathcal{G}_0}\delta\boldsymbol{x} = \boldsymbol{b}$. Importantly, its training pairs $(\boldsymbol{b}, \delta\boldsymbol{x})$ are obtained from intermediate quantities already generated when solving the nonlinear training instances for $\mathcal{N}_0^{\mathrm{sol}}$.

Specifically, consider the first Newton step for the base-topology nonlinear system. Given the initial state $\boldsymbol{x}^{(0)}$, the first correction $\delta\boldsymbol{x}^{(1)}$ satisfies:

$$\left(\boldsymbol{L}_{\mathcal{G}_0} - \boldsymbol{D}_0^{\mathrm{NR}}\right)\delta\boldsymbol{x}^{(1)} = -\boldsymbol{F}_0^{\mathrm{NR}} \tag{44}$$

where $\boldsymbol{D}_0^{\mathrm{NR}} := \partial\boldsymbol{f}/\partial\boldsymbol{x}|_{\boldsymbol{x}=\boldsymbol{x}^{(0)}}$ and $\boldsymbol{F}_0^{\mathrm{NR}} := \boldsymbol{L}_{\mathcal{G}_0}\boldsymbol{x}^{(0)} - \boldsymbol{f}(\boldsymbol{x}^{(0)}, \boldsymbol{\xi})$. Rearranging Eq. (44) yields

$$\boldsymbol{L}_{\mathcal{G}_0}\delta\boldsymbol{x}^{(1)} = \boldsymbol{D}_0^{\mathrm{NR}}\delta\boldsymbol{x}^{(1)} - \boldsymbol{F}_0^{\mathrm{NR}} := \boldsymbol{b}^{(1)} \tag{45}$$

Thus, each nonlinear training instance already provides an auxiliary pair $(\boldsymbol{b}^{(1)}, \delta\boldsymbol{x}^{(1)})$ satisfying the fixed linear relation $\boldsymbol{L}_{\mathcal{G}_0}\delta\boldsymbol{x} = \boldsymbol{b}$. Additional Newton steps, when available, provide more $(\boldsymbol{b}, \delta\boldsymbol{x})$ pairs satisfying the same linear relation. We use these pairs to train the auxiliary neural solver $\mathcal{N}_0^{\mathrm{aux}} : \boldsymbol{b} \mapsto \delta\boldsymbol{x}$, without requiring additional system solutions.

Nonlinear solution recasting. The auxiliary neural solver takes $\boldsymbol{b}$ as input. According to Theorem 1, the sensitivity matrix $\boldsymbol{S}_{\mathrm{aux}}$ reduces to $\boldsymbol{I}$, yielding:

$$\widehat{\boldsymbol{L}}_{\mathcal{G}_0}^{-1} := \boldsymbol{J}_{\mathcal{N}_0^{\mathrm{aux}}} = \frac{\partial\mathcal{N}_0^{\mathrm{aux}}}{\partial\boldsymbol{b}} \approx \boldsymbol{L}_{\mathcal{G}_0}^{-1} \tag{46}$$

For the low-rank change $\boldsymbol{L}_{\mathcal{G}} = \boldsymbol{L}_{\mathcal{G}_0} + \boldsymbol{B}_{\Delta}\boldsymbol{\Delta}\boldsymbol{B}_{\Delta}^{\top}$, Proposition 2 recasts $\widehat{\boldsymbol{L}}_{\mathcal{G}_0}^{-1}$ to yield $\widehat{\boldsymbol{L}}_{\mathcal{G}}^{-1}$. Together with the nonlinear solution neural solver, we obtain:

$$\widehat{\boldsymbol{x}}_{\mathcal{G}}^{(k+1)} = \widehat{\boldsymbol{x}}_{\mathcal{G}}^{(k)} + \widehat{\boldsymbol{L}}_{\mathcal{G}}^{-1}\left(\boldsymbol{f}(\widehat{\boldsymbol{x}}_{\mathcal{G}}^{(k)}, \boldsymbol{\xi}) - \boldsymbol{L}_{\mathcal{G}}\widehat{\boldsymbol{x}}_{\mathcal{G}}^{(k)}\right) \tag{47}$$

which gives Eq. (10).

C.3.2 Nonlinear Recastability

We next derive the sufficient condition for nonlinear recastability in Corollary 4. Define the recasting map as $\boldsymbol{T}_{\mathcal{G}}(\boldsymbol{x}) := \boldsymbol{x} + \widehat{\boldsymbol{L}}_{\mathcal{G}}^{-1}\big(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\xi}) - \boldsymbol{L}_{\mathcal{G}}\boldsymbol{x}\big)$, and write the recast inverse as $\widehat{\boldsymbol{L}}_{\mathcal{G}}^{-1} = \boldsymbol{L}_{\mathcal{G}}^{-1} + \boldsymbol{E}_{\mathcal{G}}$, where $\boldsymbol{E}_{\mathcal{G}}$ denotes the inverse-recasting error. Differentiating the recasting map yields:

$$\frac{\partial\boldsymbol{T}_{\mathcal{G}}}{\partial\boldsymbol{x}} = \boldsymbol{I} + \widehat{\boldsymbol{L}}_{\mathcal{G}}^{-1}\left(\boldsymbol{D}(\boldsymbol{x}) - \boldsymbol{L}_{\mathcal{G}}\right) = \boldsymbol{L}_{\mathcal{G}}^{-1}\boldsymbol{D}(\boldsymbol{x}) + \boldsymbol{E}_{\mathcal{G}}\left(\boldsymbol{D}(\boldsymbol{x}) - \boldsymbol{L}_{\mathcal{G}}\right) \tag{48}$$

where $\boldsymbol{D}(\boldsymbol{x}) := \partial\boldsymbol{f}/\partial\boldsymbol{x}$. Hence, if there exists a neighborhood $\mathcal{U}$ of $\boldsymbol{x}_{\mathcal{G}}$ such that

$$q := \sup_{\boldsymbol{x}\in\mathcal{U}} \left\|\boldsymbol{L}_{\mathcal{G}}^{-1}\boldsymbol{D}(\boldsymbol{x}) + \boldsymbol{E}_{\mathcal{G}}\left(\boldsymbol{D}(\boldsymbol{x}) - \boldsymbol{L}_{\mathcal{G}}\right)\right\|_2 < 1 \tag{49}$$

then $\boldsymbol{T}_{\mathcal{G}}$ is a contraction on $\mathcal{U}$, which gives the sufficient condition in Corollary 4. For an initialization within the corresponding contraction region, the recast solution converges locally to $\boldsymbol{x}_{\mathcal{G}}$ at a linear rate bounded by $\|\widehat{\boldsymbol{x}}_{\mathcal{G}}^{(k)} - \boldsymbol{x}_{\mathcal{G}}\|_2 \le q^k\|\widehat{\boldsymbol{x}}_{\mathcal{G}}^{(0)} - \boldsymbol{x}_{\mathcal{G}}\|_2$.

The approximation error in the auxiliary neural solver affects the convergence condition and rate, but not the converged solution: if the contraction condition above is satisfied, the recasting steps always converge to the true changed-topology solution $\boldsymbol{x}_{\mathcal{G}}$.

D Experimental Details

D.1 Models and Baselines

Base MLP. We use residual MLPs as the base-topology neural solvers throughout the experiments. Each model takes the system conditions $\boldsymbol{\xi}$ as input and predicts the corresponding base-topology solution $\widehat{\boldsymbol{x}}_0$. The MLP consists of an input layer, two residual hidden layers with SiLU activations, and a linear output layer. The model size is adjusted with the system size to provide sufficient capacity for different networks. Inputs and outputs are standardized using statistics from the training set. The same trained base model is kept frozen during recasting and is also evaluated directly as the Base MLP baseline.

Topology-Aware MLP. The Topology-Aware MLP uses the same residual-MLP backbone but additionally takes the topology status of the network as input through edge-status indicators. It is trained jointly on the base topology and multiple changed topologies, using four times as many training samples as the Base MLP. For the fine-tuning experiments, the trained model is further updated using samples from each target topology.

Graph Convolutional Network. We use a graph convolutional network (GCN) as the graph-learning baseline. The GCN takes link parameters, MLP prediction, and graph node information to learn the topology-induced solution correction. Node features are first encoded into 64-dimensional representations, followed by four residual graph-convolution layers and a linear output layer.

PowerFlowNet. For AC power flow, we compare against PowerFlowNet (Lin et al., 2024), a message-passing graph neural solver designed for power flow prediction. We use the implementation provided with the PFΔ benchmark (Rivera Him et al., 2026) and train it on a balanced mixture of N, $N-1$, and $N-2$ samples.

D.2 Experimental Systems

Synthetic networks. We use three representative synthetic network families (Adam-Day et al., 2024): Erdős–Rényi network (ER), Stochastic Block Model (SBM), and Barabási–Albert (BA). ER networks are used for the main scaling experiments, while SBM and BA networks are additionally considered to evaluate RecastNet across different graph structures. We consider network sizes $N \in \{50, 100, 250, 500, 1000\}$. For each network, we generate a connected base topology $\mathcal{G}_0$, assign link weights, and sample the system conditions $\boldsymbol{\xi} = (\boldsymbol{\xi}_{\mathrm{loc}}, \boldsymbol{\xi}_{\mathrm{sh}}, \boldsymbol{\xi}_{\mathrm{L}})$. Topology changes are generated by removing links from $\mathcal{G}_0$ while preserving network connectivity. Depending on the experiment, we consider either a specified number of removed links or topology changes affecting up to 20% of the network edges.

Nodal relations. To evaluate RecastNet across various nodal component models, we consider four nodal relations in the synthetic experiments: linear, quadratic, trigonometric, and mixed. Denote ξ_i as the local condition at node i and $\boldsymbol{\xi}_{\mathrm{sh}}$ the shared system conditions. The nodal injection b_i is defined as:

$$\begin{aligned}
\text{Linear:} \quad & b_i = a_i \xi_i + \boldsymbol{c}_i^\top \boldsymbol{\xi}_{\mathrm{sh}} + d_i \\
\text{Quadratic:} \quad & b_i = a_i \xi_i + \boldsymbol{c}_i^\top \boldsymbol{\xi}_{\mathrm{sh}} + d_i + q_i \left(\xi_i^2 - \frac{1}{3} \right) \\
\text{Trigonometric:} \quad & b_i = a_i \xi_i + \boldsymbol{c}_i^\top \boldsymbol{\xi}_{\mathrm{sh}} + d_i + u_i \sin(\omega_i \xi_i + \phi_i) + v_i \cos(\omega_i \xi_i - \phi_i) \\
\text{Mixed:} \quad & b_i = a_i \xi_i + \boldsymbol{c}_i^\top \boldsymbol{\xi}_{\mathrm{sh}} + d_i + q_i \left(\xi_i^2 - \frac{1}{3} \right) + u_i \sin(\omega_i \xi_i + \phi_i) + v_i \cos(\omega_i \xi_i - \phi_i)
\end{aligned}$$

The coefficients $(a_i, \boldsymbol{c}_i, d_i, q_i, u_i, v_i, \omega_i, \phi_i)$ are randomly generated once for each network and then fixed across all operating conditions and topology changes.

Real-world power networks. We use four benchmark power systems from MATPOWER (Zimmerman et al., 2010): IEEE 57-bus, IEEE 118-bus, South Carolina 500-bus (ACTIVSg500), and European 1354-bus (case1354pegase). Together, these cases cover different grid sizes and connectivity patterns. Each MATPOWER case has its own bus, branch, and generator information, which can be constructed into a graph system. In particular, we consider the bus as the node and the transmission line as the link. We consider both DC and AC power flow, with $N-k$ transmission-line outages or grid reconfiguration representing topology changes.

D.3 Evaluation Metrics

We evaluate RecastNet using the following metrics.

Relative Frobenius error. This metric measures the accuracy of solution-to-inverse recasting:

$$e_{\mathrm{inv}}(\%) = \frac{\| \widehat{\boldsymbol{L}}_{\mathcal{G}_0}^{-1} - \boldsymbol{L}_{\mathcal{G}_0}^{-1} \|_{\mathrm{F}}}{\| \boldsymbol{L}_{\mathcal{G}_0}^{-1} \|_{\mathrm{F}}} \times 100 \tag{50}$$

Here, $\widehat{\boldsymbol{L}}_{\mathcal{G}_0}^{-1}$ and $\boldsymbol{L}_{\mathcal{G}_0}^{-1}$ denote the recast and exact base-topology Laplacian inverses, respectively. This metric is used in Figure 2.

Relative solution error. This metric measures the accuracy of the predicted solution under a changed topology $\mathcal{G}$:

$$e_{\mathrm{sol}}(\%) = \frac{\|\widehat{\boldsymbol{x}}_{\mathcal{G}} - \boldsymbol{x}_{\mathcal{G}}\|_2}{\|\boldsymbol{x}_{\mathcal{G}}\|_2} \times 100 \tag{51}$$

Here, $\widehat{\boldsymbol{x}}_{\mathcal{G}}$ and $\boldsymbol{x}_{\mathcal{G}}$ denote the predicted and exact solutions under $\mathcal{G}$, respectively. We report the median and 95th percentile (P95) of this metric across experiments in Figures 3, 4, 5c and Tables 1 and 2.

Gain of recasting. This metric measures how much of the additional prediction error caused by a topology change is removed by recasting. Denote E_{MLP} as the relative solution error obtained by directly applying the frozen Base MLP to the changed topology, E_{Recast} as the error after recasting, and E_{Base} as the Base MLP error on the original topology. We define:

$$\mathrm{Gain}(\%) = \frac{E_{\mathrm{MLP}} - E_{\mathrm{Recast}}}{E_{\mathrm{MLP}} - E_{\mathrm{Base}}} \times 100 \tag{52}$$

A Gain of 0% indicates no improvement over the frozen Base MLP, while 100% indicates that recasting reduces the solution error back to the base-topology error level. This metric is reported in Table 1.

Recovery of topology-induced solution changes. This metric evaluates how well RecastNet captures the change in solution caused by a topology change. Denote the true solutions on the base and changed topologies as $\boldsymbol{x}_0$ and $\boldsymbol{x}_{\mathcal{G}}$, respectively, such that the true topology-induced solution change is $\Delta\boldsymbol{x} = \boldsymbol{x}_{\mathcal{G}} - \boldsymbol{x}_0$. Similarly, the change predicted by RecastNet is $\widehat{\Delta\boldsymbol{x}} = \widehat{\boldsymbol{x}}_{\mathcal{G}} - \widehat{\boldsymbol{x}}_0$. We measure the recovery of these solution changes across test instances as:

$$\mathrm{Recovery}(\%) = \left(1 - \frac{\left\|\widehat{\Delta\boldsymbol{x}} - \Delta\boldsymbol{x}\right\|_2^2}{\|\Delta\boldsymbol{x}\|_2^2}\right) \times 100 \tag{53}$$

A Recovery of 100% means that RecastNet exactly reproduces the solution change caused by the topology modification. A Recovery of 0% corresponds to predicting no solution change, i.e., directly reusing the base-model prediction. This metric is used in Figure 3c.