# IS GRAPH MIXUP BENEFICIAL? INVESTIGATING INTERPOLATION AND EMPIRICAL PERFORMANCE OF GRAPH MIXUP METHODS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Mixup is a widely used data augmentation technique that constructs new training examples by interpolating between existing ones. While effective in domains like vision and language, applying mixup to graph data is challenging. In this paper, we analyze and empirically explore state-of-the-art graph mixup methods. We conducted an independent evaluation following established evaluation protocols for graph classification and found that none of the mixup methods yielded statistically significant improvements over the no-mixup baseline. To obtain further insights, we analyzed the graphs generated from existing mixup methods from an interpolation perspective using the graph edit distance. We found that (i) many mixup methods failed to interpolate well, (ii) that mixup methods that interpolated well often outperform methods that did not, (iii) even optimal interpolation did not lead to performance improvements. Our findings highlight the need for a more rigorous exploration and evaluation of mixup for graphs.

## 1 INTRODUCTION

Data augmentation is an essential technique for improving generalization in machine learning and is particularly useful in domains where training data is scarce. Mixup (Zhang et al., 2018), a popular data augmentation technique, creates new training examples by *interpolating* between existing ones. Originally introduced in computer vision (Zhang et al., 2018), mixup has shown strong regularization and calibration effects (Thulasidasan et al., 2019) in other domains as well; e.g., in speech recognition (Meng et al., 2021) and natural language processing (Sun et al., 2020). Mixup is appealing due to its simplicity and intuitive design, its applicability without requiring domain-specific knowledge, and since it affects both the input and label space (in contrast to augmentation techniques such as DropNode (Do et al., 2021) or DropEdge (Rong et al., 2019)). The resulting soft labels encourage linear separation between classes in the model's representation space.

In this paper, we revisit mixup for graph classification tasks. Since graphs are complex and irregular, it is not immediately obvious how mixup should be performed. In the recent years, several alternative approaches for graph mixup have been proposed to address this question, drawing from domains such as optimal transport (Villani, 2008), graph theory (Lovász and Szegedy, 2006), and graph matching networks (Li et al., 2019). However, reproduction issues have been raised for some of the methods (Omeragic and Duranović, 2023), there are no independent evaluations of graph mixup methods, and the evaluations that have been performed often focused on empirical performance and did not analyze the produced mixup graphs directly.

**Main contributions (C) and results (R). (C1)** We address these gaps by first performing an independent evaluation of graph mixup in a unified experimental setup following established evaluation protocols (Errica et al., 2019). **(R1)** We found that graph mixup provided no significant improvement over the no-mixup baseline, which questions the practical benefits of graph mixup. **(C2)** We then performed a pooled analysis in weaker statistical setups. **(R2)** Even after pooling, graph mixup provided no significant improvement over the no-mixup baseline. **(C3)** To obtain further insights, we then systematically analyzed the mixup graphs produced by existing mixup methods from an interpolation perspective. **(R3)** We found that most graph mixup methods did not interpolate well, despite that **(R4)** good interpolation properties were beneficial for empirical performance.

Table 1: Overview of graph mixup methods

| Method | Interpolating? | Inputs | Alignment | Output | Learned? |
|---|---|---|---|---|---|
| If-Mixup | ✓ | Adjacency matrices | Arbitrary | Edge-weighted graph | × |
| S-Mixup | ✓ | Adjacency matrices | Learned | Edge-weighted graph | ✓ |
| SubMix | ✓ | Graphs | Random | Graph | × |
| G-Mixup | ✓ | Graphons | Degree | Graphon | ✓ |
| FGW-Mixup | ✓ | Graphs | FGW coupling | Graph | × |
| GeoMix | ✓ | Graphs | GW coupling | Edge-weighted graph | × |
| Embedding Mixup | ✓ | Embeddings | N.A. | Embedding | × |
| GED-Mixup (Sec. 4.3) | ✓ | Graphs | Optimal | Graph | × |

## 2 Graph Mixup

Mixup was originally introduced by Zhang et al. (2018) as a data augmentation technique for supervised learning tasks, particularly in computer vision and speech recognition. In this paper, we study mixup for supervised graph classification, where the inputs are graphs (potentially including node/edge features) and the goal is to learn a classifier for unseen graphs from a set of labeled examples. Application areas include the biomedical data (Qabel et al., 2022; Wang et al., 2025; Buterez et al., 2024), bioinformatics (Jang et al., 2024; van der Weg et al., 2025; Jha et al., 2022)), cybersecurity (Bilot et al., 2024), fraud detection (Motie and Raahemi, 2024), and many more (Cao et al., 2024; Park et al., 2022; Jin et al., 2025). Training data is often scarce in these applications so that mixup is a promising approach to combat overfitting.

**Linear mixup.** Zhang et al. (2018) considered inputs and labels represented as real-valued tensors (e.g., an image and its one-hot encoded class label) and performed mixup by linear interpolation. More precisely, given two input examples $(x_1, y_1)$ and $(x_2, y_2)$, their *linear mixup* approach constructs a synthetic example $(x_M, y_M)$ by taking a convex combination of both inputs and labels

$$x_M = x_1 + \lambda(x_2 - x_1) = (1-\lambda)x_1 + \lambda x_2 \quad \text{and} \quad y_M = (1-\lambda)y_1 + \lambda y_2, \tag{1}$$

where $\lambda \in [0, 1]$ refers to a *mixup ratio*. Intuitively, $\lambda$ describes how far the result moves away from the first towards the second input. Linear mixup improved generalization and robustness in their experimental study, and mixup methods have been widely adopted and extended since then (Shamsian et al., 2024; Yun et al., 2019; Touvron et al., 2021; Liu et al., 2021; Verma et al., 2019; Ramé et al., 2021; Bao et al., 2023; Zou et al., 2023).

**Graph mixup.** Graph mixup methods formulate graph mixup—in the spirit of linear mixup—as *interpolation* between two example graphs and their labels. Conceptually, these methods interpolate the class labels of their inputs using Eq. (1), but differ in how they interpolate between the input graphs themselves. Intuitively, most methods use *alignments* to determine "common parts" between the two input graphs (Fig. 1, top right, common parts color-coded). Mixup is then performed only on the "different parts" by including nodes and edges from both graphs proportional to the desired mixup ratio (see "good mixup" in Fig. 1, $\lambda = 40\%$). This is similar to linear mixup, in which common elements of the input tensors are left unchanged and different elements are subject to mixup.

**Graph mixup methods.** Several graph-specific mixup methods have been introduced in the literature, including If-Mixup (Guo and Mao, 2023), S-Mixup (Ling et al., 2022), SubMix (Yoo et al., 2022), G-Mixup (Han et al., 2022), FGW-Mixup (Ma et al., 2023), GeoMix (Zeng et al., 2024), and Embedding Mixup (Wang et al., 2021). Key differences between these methods include: (i) the inputs to mixup, (ii) how these inputs are aligned, (iii) the output of mixup, and (iv) whether or not the mixup method itself is learned. A brief overview along these dimensions is given in Tab. 1; see also App. A.

**Inputs to mixup.** Mixup can be performed on (i) two graphs, (ii) two adjacency matrices, (iii) two graphons (each representing the set of graphs associated with a class label), or (iv) two graph embeddings produced by the downstream network (along the lines of Manifold Mixup (Verma et al., 2019)). For (ii)–(iv), interpolation is typically done using linear mixup after suitable preprocessing (e.g., reordering adjacency matrices according to an alignment and adding singleton nodes to match their sizes), whereas (i) is handled differently. In particular, SubMix (Yoo et al., 2022) adopts a strategy inspired by CutMix (Yun et al., 2019): given two input graphs, it replaces a subgraph of one input with a subgraph of the other input. FGW-Mixup (Ma et al., 2023) and GeoMix (Zeng

Table 2: Test accuracy (%) and standard error (pp) for multiple datasets. Missing entries indicate cases where FGW-Mixup could not generate mixup graphs. Statistically significant differences over the no-mixup baseline are marked bold (there are none).

| Model | Method | MUTAG | ENZYMES | IMDB-BINARY | PROTEINS | Average |
|---|---|---|---|---|---|---|
| GCN | Baseline | 78.42 ± 1.71 | 72.56 ± 1.21 | 68.80 ± 1.00 | 71.83 ± 2.94 | 72.90 ± 2.01 |
| | Emb-M. | 79.66 ± 1.45 | 70.89 ± 1.21 | 66.60 ± 1.85 | 70.09 ± 2.13 | 71.81 ± 2.78 |
| | FGW-M. | 76.82 ± 2.34 | – | – | – | – |
| | G-Mixup | 82.48 ± 1.33 | 68.28 ± 2.02 | 69.40 ± 1.13 | 74.13 ± 1.05 | 73.57 ± 3.23 |
| | GeoMix | 75.63 ± 3.43 | 74.00 ± 1.19 | 62.67 ± 2.23 | 71.65 ± 2.61 | 70.99 ± 2.89 |
| | If-Mixup | 81.43 ± 1.56 | 72.06 ± 0.98 | 68.53 ± 1.11 | 74.88 ± 1.06 | 74.22 ± 2.73 |
| | S-Mixup | 80.21 ± 1.67 | 67.29 ± 5.38 | 70.06 ± 1.61 | 72.26 ± 1.97 | 72.46 ± 2.78 |
| | SubMix | 80.03 ± 1.88 | 73.39 ± 1.42 | 68.20 ± 1.27 | 72.87 ± 2.17 | 73.62 ± 2.43 |
| | GED-M.[1] | 81.64 ± 1.81 | 72.11 ± 1.39 | 68.97 ± 1.23 | 73.44 ± 1.39 | 74.04 ± 2.70 |
| GIN | Baseline | 84.41 ± 1.39 | 70.33 ± 0.98 | 70.77 ± 0.53 | 69.91 ± 3.45 | 73.86 ± 3.52 |
| | Emb-M. | 81.89 ± 1.34 | 70.78 ± 1.02 | 67.77 ± 1.89 | 71.26 ± 2.59 | 72.92 ± 3.09 |
| | FGW-M. | 82.81 ± 1.42 | – | – | – | – |
| | G-Mixup | 80.49 ± 1.75 | 69.17 ± 1.11 | 65.90 ± 2.38 | 68.62 ± 3.40 | 71.05 ± 3.23 |
| | GeoMix | 81.78 ± 2.23 | 69.00 ± 1.31 | 70.53 ± 0.60 | 69.46 ± 2.96 | 72.69 ± 3.05 |
| | If-Mixup | 84.09 ± 1.39 | 70.06 ± 1.38 | 69.30 ± 0.72 | 70.12 ± 3.51 | 73.39 ± 3.57 |
| | S-Mixup | 80.83 ± 0.90 | 68.96 ± 1.35 | 69.29 ± 2.16 | 62.37 ± 2.24 | 70.36 ± 3.84 |
| | SubMix | 84.75 ± 1.64 | 70.72 ± 1.43 | 70.40 ± 0.45 | 71.08 ± 2.70 | 74.24 ± 3.51 |
| | GED-M.[1] | 82.84 ± 1.35 | 71.17 ± 0.95 | 70.40 ± 0.76 | 68.30 ± 3.35 | 73.18 ± 3.28 |

[1] Introduced in Sec. 4.3.

et al., 2024) rely on the (Fused-)Gromov-Wasserstein distance (Vayer et al., 2020) from the theory of optimal transport (Villani, 2008). While the former method aims to compute barycenters, the latter relies on geodesics (Peyré et al., 2016). In both cases, mixup takes place in the Gromov-Wasserstein space and is computationally expensive so that approximation algorithms are used.

**Alignments.** Most graph mixup methods (implicitly or explicitly) make use of an alignment between their inputs. Obtaining a good alignment can be challenging and computationally expensive (Chang et al., 2023). Different approaches have been explored: (i) arbitrary (i.e., determined by how the graphs happen to be provided), (ii) learned, (iii) random, (iv) degree-based ordering, (v) a coupling in the sense of optimal transport (Villani, 2008). We also explore (vi) an optimal alignment in this paper (for analysis; see Sec. 4.3). Note that Embedding Mixup (Wang et al., 2021) applies mixup on graph embeddings; here the notion of alignment is not directly applicable.

**Outputs.** Mixup methods can produce as output: (i) a mixup graph, (ii) an edge-weighted mixup graph, (iii) a graphon, and (iv) embeddings. Here (i) stays in the input space, (ii) produces graphs in which edges are labeled with "existence probabilities", (iii) can be used to sample mixup graphs, and (iv) stays in the embedding space of the downstream network.

**Learned mixup.** S-Mixup (Ling et al., 2022) and G-Mixup (Han et al., 2022) use learned mixup, i.e., they need to be trained on the training data used for the downstream task beforehand.

## 3 Empirical Analysis

We performed an independent experimental study to evaluate the empirical performance of state-of-the-art graph mixup methods for graph classification in a common setup. Our key goals were to assess to what extent graph mixup is beneficial in that it increases prediction performance.

**Experimental setup.** We independently evaluated on four representative datasets from TUDataset (Morris et al., 2020) commonly used for graph classification tasks (see App. B for dataset statistics), using GCN (Kipf and Welling, 2017) and GIN (Xu et al., 2018) as backbone models. Our methodology followed the careful choices of Errica et al. (2019), i.e., nested cross-validation for model selection and assessment, repeated runs for robustness, and significance testing via Welch's $t$-test (Welch, 1947). We report mean test accuracy with standard errors across 5-fold nested CV, with three repetitions per split. We considered the seven graph mixup variants of Tab. 1 along with GED-Mixup

Table 3: Pooled average test accuracy (%), pooled standard errors (pp) and $p$-values for GCN/GIN and the evaluation datasets under assumptions (A1)–(A3). FGW-Mixup is excluded due to missing results for some datasets. Statistical significant results over the no-mixup baseline are marked in bold.

| Method | Accuracy | A1 (standard) | | A2 (fixed dist.) | | A3 (fixed data) | |
|---|---|---|---|---|---|---|---|
| | | SE | p | SE | p | SE | p |
| Baseline | 73.38 | ±0.86 | – | ±0.67 | – | ±0.44 | – |
| Emb-M. | 72.37 | ±0.82 | 0.40 | ±0.62 | 0.27 | ±0.43 | 0.10 |
| G-Mixup | 72.31 | ±0.92 | 0.39 | ±0.68 | 0.27 | ±0.39 | 0.69 |
| GeoMix | 71.84 | ±0.97 | 0.24 | ±0.80 | 0.14 | ±0.40 | **0.01** |
| If-Mixup | 73.81 | ±0.83 | 0.72 | ±0.59 | 0.63 | ±0.36 | 0.45 |
| S-Mixup | 71.41 | ±1.12 | 0.16 | ±0.87 | 0.08 | ±0.63 | **0.01** |
| SubMix | 73.93 | ±0.83 | 0.64 | ±0.61 | 0.55 | ±0.34 | 0.32 |
| GED-Mixup (Sec. 4.3) | 73.61 | ±0.82 | 0.85 | ±0.60 | 0.80 | ±0.26 | 0.65 |

(a baseline introduced in Sec. 4.3). Mixup graphs were randomly generated in each training epoch. Hyperparameters—including model, training, and mixup settings—were tuned individually for each dataset/method via random search plus Bayesian optimization with TPE (Bergstra et al., 2011) using Optuna (Akiba et al., 2019). The experimental setup is described in detail in App. C.

**Result ①: Graph mixup provided no significant improvement over the no-mixup baseline.** Tab. 2 shows the results for all methods evaluated on GCN (Kipf and Welling, 2017) and GIN (Xu et al., 2018), respectively. Although some mixup methods appear to improve over or fall behind the no-mixup baseline, none of these differences were statistically significant. These results question whether mixup is beneficial for graph classification tasks.

**Pooled analysis.** To provide more insight, we performed a pooled analysis in weaker statistical setups. We make three assumptions of increasing strength: (A1, standard): Treat datasets and model classes as sampled from a dataset and model class distribution. This allows us to make statements about empirical performance on new datasets and model classes, which is what we are ultimately interested in. (A2, fixed distribution): Treat datasets and model class as sampled from fixed data and model distributions. This allows to make statements about empirical performance of GCN/GIN when applied to the data distribution underlying our evaluation datasets. (A3, fixed data): Treat the evaluation data as the entire population (i.e., treat the empirical distribution as the data distribution). This allows us to make statements about the particular data that is used (but not about their underlying data distribution). More details are provided in App. D.

**Result ②: Even after pooling, graph mixup provided no significant improvement over the no-mixup baseline.** Our pooled results are shown in Tab. 3. First, under (A1) and (A2), none of the results were significant even after pooling. For (A3), the strongest set of assumptions, we obtained statistical significance in that some methods performed worse than the no-mixup baseline. So even in the most generous point of view (A3), we did not obtain statistically significant results in favor of mixup. Reasons for this negative result include the suitability of graph mixup in general, potential flaws in the mixup methods, or insufficient power (e.g., due to small effect sizes).

## 4 Mixup as Interpolation

To investigate the failure of all considered graph mixup methods to produce significant performance benefits over the no-mixup baseline in our experiments, we now take a closer look at the generated mixup graphs. Recall from Sec. 2 that the goal of graph mixup is to interpolate between input graphs, according to a pre-specified mixup ratio $\lambda$. In this section, we formalize this interpolation goal and propose *interpolation error* metrics to quantify to what extent a mixup graph actually interpolates between inputs. To the best of our knowledge, such an analysis has not been done before.

In Sec. 5, we will use these results to study relationship between interpolation properties and empirical performance of graph mixup methods. We also consider an approach called GED-Mixup that interpolates optimally according to our metrics. While this method may not be practically viable in some applications due to its high computational costs, it provides a baseline result for the performance that optimal interpolation can achieve.

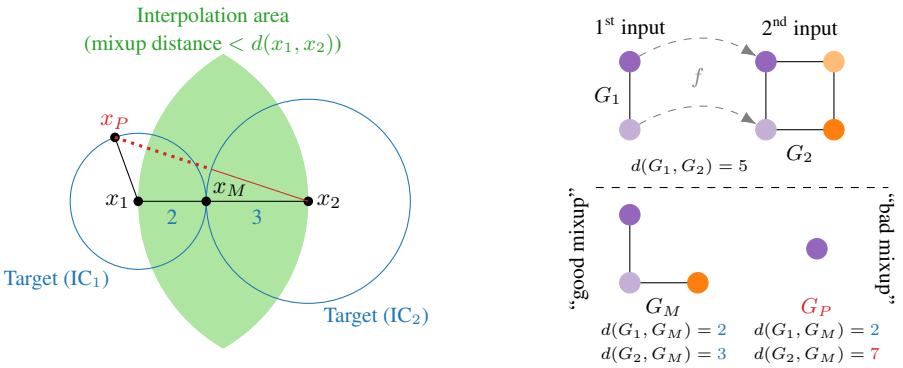



Figure 1: *Left:* $x_M$ $(d_2, 2/5)$-interpolates between inputs $x_1$ and $x_2$, whereas $x_P$ does not. The dotted line indicates $x_P$'s absolute interpolation error (AIE). *Right:* Similarly, $G_M$ $(d_{\text{GED}}, 2/5)$-interpolates between $G_1$ and $G_2$, whereas $G_P$ does not. We have $\text{AIE}(G_P) = |2-2| + |7-3| = 4$ and $\text{IE}(G_P) = 4/5$. The colors symbolize different node attributes. The mapping $f : V_1 \to V_2$ constitutes a *vertex mapping* or *alignment* between $G_1$ and $G_2$.

### 4.1 Interpolation Criteria and Interpolation Error

We first make the intuition of "interpolation" more precise. Consider two inputs $x_1$ and $x_2$, a mixup ratio $\lambda \in [0, 1]$, and a mixup result $x_M$. Given a distance metric $d(\cdot, \cdot)$ between inputs, we say that $x_M$ $(d, \lambda)$-*interpolates* between $x_1$ and $x_2$ if the following *interpolation criteria* (IC)

$$\text{IC}_1\colon d(x_M, x_1) = \lambda \cdot d(x_1, x_2) \quad \text{and} \quad \text{IC}_2\colon d(x_M, x_2) = (1-\lambda) \cdot d(x_1, x_2) \tag{2}$$

are satisfied. We refer to the right-hand sides of $\text{IC}_1$ and $\text{IC}_2$ as *interpolation targets*. If both targets are met, then (i) $d(x_1, x_2) = d(x_1, x_M) + d(x_M, x_2)$ so that $x_M$ lies on a shortest path between $x_1$ and $x_2$ (w.r.t. $d$ and by the triangle equality) and (ii) the position of $x_M$ on this shortest path precisely matches the desired relative contribution of $x_1$ (i.e., $1-\lambda$) and $x_2$ (i.e., $\lambda$).

To gain some intuition, observe that linear mixup of Eq. (1) produces the unique point that satisfies IC w.r.t. the Euclidean distance $d_2(x_1, x_2) = \|x_1 - x_2\|_2$ (and others, see Prop. 1 below), for both inputs and labels. This is visualized in Fig. 1 (left). Here $x_M$ is the result of linear mixup and $(d_2, 2/5)$-interpolates between $x_1$ and $x_2$. In contrast, $x_P$ satisfies only $\text{IC}_1$ but not $\text{IC}_2$. As this example highlights, both criteria are needed.

As we will see later, graph mixup methods often do not satisfy IC exactly but only approximately. To quantify the approximation error, we introduce the *absolute interpolation error (AIE)* given by

$$\text{AIE}_d(x_M; x_1, x_2, \lambda) \stackrel{\text{def}}{=} \big|d(x_M, x_1) - \lambda \cdot d(x_1, x_2)\big| + \big|d(x_M, x_2) - (1-\lambda) \cdot d(x_1, x_2)\big|.$$

For brevity, we often write $\text{AIE}(x_M) = \text{AIE}_d(x_M; x_1, x_2, \lambda)$ and consider the remaining quantities as arbitrary but fixed. Observe that if $x_M$ $(d, \lambda)$-interpolates between $x_1$ and $x_2$, then $\text{AIE}(x_M) = 0$ (e.g., $x_M$ in Fig. 1). If it does not, then $\text{AIE}(x_M) > 0$ (e.g., $x_P$ in Fig. 1 w.r.t. $d_2$). Intuitively, the AIE measures the distance of the mixup result to actual $(d, \lambda)$-interpolation targets (cf. Fig. 1). As this distance can be arbitrarily large, the AIE is not bounded from above.

To be able to compare interpolation errors across input pairs $(x_1, x_2)$ with different distances $d(x_1, x_2)$ in a meaningful way, we normalize AIE w.r.t. $d(x_1, x_2)$ to obtain the mixup *interpolation error (IE)*:

$$\text{IE}_d(x_M; x_1, x_2, \lambda) \stackrel{\text{def}}{=} \frac{\text{AIE}_d(x_M; x_1, x_2, \lambda)}{d(x_1, x_2)} = \left|\frac{d(x_M, x_1)}{d(x_1, x_2)} - \lambda\right| + \left|\frac{d(x_M, x_2)}{d(x_1, x_2)} - (1-\lambda)\right|. \tag{3}$$

The following proposition states that linear mixup interpolates optimally.

**Proposition 1.** *For any distance metric $d(a, b) = \|a - b\|$ induced by a norm $\|\cdot\|$ on the input/label vector space (over $\mathbb{R}$), linear mixup of Eq. (1) satisfies IC for inputs/labels and we have*

$$\text{AIE}_d(x_M) = \text{IE}_d(x_M) = 0 \quad / \quad \text{AIE}_d(y_M) = \text{IE}_d(y_M) = 0.$$

Such distances include, for example, the Manhattan distance (induced by 1-norm) and the Euclidean distance (induced by 2-norm). See App. E for a proof.

### 4.2 Alignments, Edit Sets, and the Graph Edit Distance

As discussed in Sec. 2, graph mixup methods rely on alignments to produce mixup graphs. We now formalize the notion of an alignment, describe how alignments relate to edits sets and graph mixup, and finally define an optimal alignment based on the graph edit distance.

**Alignments.** Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs where $|V_1| \leq |V_2|$ w.l.o.g. (otherwise swap $G_1$ and $G_2$). Intuitively, an alignment assigns to each node $v \in V_1$ a unique corresponding node $u \in V_2$. We formalize an alignment as an *injective vertex mapping* $f : V_1 \to V_2$; see Fig. 1 (top right) for an example. Recall that mixup methods aim to retain the "common parts" (sub-structures such as nodes, edges, or subgraphs) that exist in both input graphs and to mixup the remaining "different parts." An alignment formalizes what is meant by "common parts": for every node $v \in V_1$ its corresponding node $f(v) \in V_2$, and for every edge $(v_1, v_2) \in E_1$ its corresponding edge $(f(v_1), f(v_2)) \in E_2$ (if present). In the example of Fig. 1, the common part is given by the two purple nodes and the edge connecting these nodes.

**Edit sets.** Given an alignment, we can perform mixup by editing $G_1$ to bring it to closer towards $G_2$. To do so, we make use of *edit operations*. An edit operation is a node or edge insertion, deletion, or substitution (i.e., changing features). Let $\mathcal{F}(G_1, G_2)$ denote the set of all *edit sets*—i.e., sets of edit operations—that transform $G_1$ into $G_2$,[1] i.e.,

$$\mathcal{F}(G_1, G_2) = \big\{F = \{e_1, e_2, \ldots, e_{|F|}\} : \text{apply}(G_1, F) \cong G_2\big\},$$

where $e_i$ denotes an edit operation and $\text{apply}(G_1, F)$ denotes the result of applying all edit operations in $F$ to $G_1$. For any pair of graphs, there is an infinite number of edit sets that transform one into the other. An alignment $f$ induces a particular edit set $F_f \in \mathcal{F}(G_1, G_2)$, which only contains the edit operations for the "different parts." The edit set induced by $f$ in Fig. 1 contains five operations: two node insertion operations (for the orange nodes) and three edge insertion operations (for the edges incident to these nodes). Given $f$, edit set $F_f$ is cheap to obtain, i.e., in asymptotically linear time with respect to the number of nodes and edges (see App. A and Chang et al. (2023)).[2]

Given an edit set, we can perform mixup by applying a $\lambda$-fraction of the operations to $G_1$ (see Sec. 4.3 for details). In Fig. 1, mixup graph $G_M$ has been generated in this fashion with $\lambda = 2/5$ (and hence using 2 out of the 5 edit operations).

**Optimal alignments and graph edit distance.** An alignment is *optimal* if its induced edit set is as small as possible. Intuitively, this means that the alignment identifies a large common part. Alignment $f$ of Fig. 1 is such an optimal alignment. The size of the edit set induced by optimal alignment is given by the *graph edit distance (GED, Sanfeliu and Fu (1983), Chang et al. (2023))*:

$$d_{\text{GED}}(G_1, G_2) = \min_{F \in \mathcal{F}(G_1, G_2)} |F|.$$

In what follows, we write $d(G_1, G_2) = d_{\text{GED}}(G_1, G_2)$ for brevity. An example is given in Fig. 1.

Generally, computing the optimal alignment and/or GED is an NP-hard problem (Zeng et al., 2009). In fact, graph mixup methods typically do *not* use optimal alignments (cf. Tab. 1), and hence may produce problematic mixup graphs. For example, graph $G_P$ in Fig. 1 does not interpolate well between $G_1$ and $G_2$; graph $G_P$ has larger distance to $G_2$ than $G_1$ has to $G_2$ so that it does not interpolate at all. We further explore such questions in Sec. 5.

**Interpolation error.** The graph edit distance is the natural choice to quantify the interpolation error of mixup graphs using Eq. (3). In fact, most (all but Embedding Mixup) graph mixup methods implicitly make use of an alignment and its corresponding induced edit set. With this choice, the "good mixup" graph $G_M$ ("bad mixup" graph $G_P$) of Fig. 1 has interpolation error of 0 ($4/5$).

**Computational cost.** Even though GED computation is NP-hard, its computation can be feasible in practice. This is due to the availability of high-performance algorithms (e.g., Chang et al. (2020; 2023)) and since in our setting of graph classification, the graphs are comparably small (e.g., see dataset statistics in App. B). In our experimental study, we did not run into computational bottlenecks.

[1] Strictly speaking, $G_1$ is transformed into another graph $G_2'$ that is isomorphic to $G_2$, denoted by $G_2' \cong G_2$.

[2] For example, compare the label of each $v \in V_1$ with the label of $f(v) \in V_2$ and add a substitution operation when they differ. As another example, for each edge $(v_1, v_2) \in E_1$, check whether $(f(v_1), f(v_2)) \in E_2$ is present and add an edge insertion operation otherwise.

**Algorithm 1** GED-Mixup

**Require:** Graphs $G_1, G_2$; mixup ratio $\lambda \in [0, 1]$
**Ensure:** Mixup graph $G_M$
$F^* \leftarrow$ a minimal edit set from $G_1$ to $G_2$ (i.e., $|F^*| = d_{\text{GED}}(G_1, G_2)$)
$P \leftarrow$ a valid ordering of the edit operations in $F^*$ (e.g., chosen at random)
$P_\lambda \leftarrow$ the first $\text{round}(\lambda|P|)$ edit operations in $P$
**return** $G_M = \text{apply}(G_1, P_\lambda)$

We modified the code of `AStar-BMao` (Chang et al., 2023), a state-of-the-art algorithm for exact GED computation, such that it additionally yielded an optimal alignment $f^*$ and its induced edit set $F^*$—i.e., $|F^*| = d(G_1, G_2)$—as a by-product without any significant additional compute cost.

### 4.3 A Baseline Method: GED-Mixup

Most graph mixup methods rely on alignments when interpolating graphs. This raises the natural question of whether or not using an optimal alignment (instead of an approximate one) would benefit graph mixup. To investigate this question, we construct a simple baseline method—coined GED-Mixup—which uses the optimal alignment and serves as an analysis tool to study its effect on empirical performance. Without including GED-Mixup in our analyses, the effect of optimal interpolation on empirical performance would remain unclear. The method can be seen as a simplification of EPIC (Heo et al., 2024).[3]

The method is described briefly in Alg. 1 and in more detail in App. F. It first computes a minimal edit set, orders the edit operation in the set, and the applies a fraction of $\lambda$ of the edit operations to $G_1$. We only consider *valid orderings*, in which (i) an edge can only be inserted when its source and target node are present, (ii) a node can only be removed when it does not have an incident edge, (iii) label substitutions are only possible for nodes/edges present in the graph, and (iv) when both $G_1$ and $G_2$ are connected, so is $G_M$. This approach avoids undesirable mixup results.

The following proposition shows that GED-Mixup is optimal in that its interpolation error is as small as possible (in particular, 0 whenever interpolation targets are integer). A proof is given in App. G.

**Proposition 2.** *GED-Mixup (Alg. 1) interpolates optimally w.r.t. $d_{GED}$, and it holds*

$$\text{AIE}(G_M) = 2 \cdot |\text{round}(\lambda \cdot d_{GED}(G_1, G_2)) - \lambda \cdot d_{GED}(G_1, G_2)| \leq 1.$$

## 5 Interpolation Analysis

Equipped with the analysis tools AIE, IE, and GED-Mixup, we empirically investigate in this section how well existing mixup methods interpolate and how this related to their performance.

**Experimental setup.** We follow the experimental setup described in Sec. 3, but only considered methods that produce mixup graphs, i.e., SubMix (Yoo et al., 2022), If-Mixup (Guo and Mao, 2022), S-Mixup (Ling et al., 2022), GeoMix (Zeng et al., 2024), FGW-Mixup (Ma et al., 2023), and GED-Mixup.[4] During training, we collected all generated mixup graphs as well as their inputs and corresponding value of $\lambda$.[5] We used this approach because it allows us to analyze the mixup graphs actually used during training, and because some mixup methods are learned based on training data. Given a set $T = \{ (G_1, G_2, G_M, \lambda)_i \}_{i=1}^n$ of mixup graphs, we report the *mean interpolation error*

[3] EPIC uses learned cost models and GED approximations whereas our approach simply uses unit edit costs and exact GED. We did not consider EPIC in our experimental study as there is no source code available.

[4] Methods that produce weighted mixup graphs, in which each edge is annotated with an "existence probability", are marked with $*$. To treat these methods appropriately when computing mIE, we account for edge weights by sampling edges according to their probability. Corresponding mIE scores can hence be interpreted as an "expected mIE". Accounting for edge weights in this fashion always decreased the corresponding mIE scores substantially.

[5] As $\lambda$-values are sampled from a Beta distribution during training or were out of our control (in SubMix), we allow for a small tolerance of $\varepsilon = 0.005$.

Table 4: Comparison of mean interpolation error (mIE) obtained by various methods and datasets (lower is better and $\geq 2$ is particularly bad). We were unable to generate graphs with FGW-Mixup on some datasets (denoted with –). Emb-Mixup and G-Mixup do not appear here as we only considered methods that perform pairwise mixup of two examples (required by mIE). More details can be found in App. H.

| Method | MUTAG | ENZYMES | IMDB-BINARY | PROTEINS | Average |
|---|---|---|---|---|---|
| GED-Mixup | 0.05 | 0.01 | 0.01 | 0.01 | 0.02 |
| SubMix | 0.45 | 0.28 | 0.49 | 0.30 | 0.38 |
| If-Mixup* | 1.57 | 0.69 | 0.86 | 1.00 | 1.03 |
| S-Mixup* | 3.56 | 0.95 | 1.32 | 0.72 | 1.64 |
| GeoMix* | 4.31 | 1.32 | 0.66 | 1.60 | 1.97 |
| FGW-Mixup | 2.76 | – | – | – | 2.76 |

*(mIE)* given by

$$\mathrm{mIE}(T) = \frac{1}{n} \sum_{(G_1,G_2,G_M,\lambda)\in T} \mathrm{IE}_{d_{\mathrm{GED}}}(G_M; G_1, G_2, \lambda).$$

An mIE value of 0 indicates that all mixup graphs perfectly interpolate between their inputs; larger values indicate larger errors. Values greater than 2 generally indicate bad interpolation properties. To see this, observe that simply setting $G_M = G_1$ for an input pair leads to IE $\leq 2$ (independently of $\lambda$); hence this clearly flawed approach already leads to an mIE $\leq 2$ when used throughout training.

**Result ③: Most graph mixup methods did not interpolate well.** We sampled 500 mixup graphs for each combination of method, dataset, and choice of $\lambda \in [0.5 \pm \varepsilon], [0.8 \pm \varepsilon], [0.9 \pm \varepsilon]$ for $\varepsilon = 0.005$. Our results are summarized in Tab. 4; more detailed results are given in App. H. As can be seen, only GED-Mixup and SubMix generally produced graphs that interpolated well. This is expected for GED-Mixup, since it interpolates optimally by design. SubMix uses random alignments, but uses a more coherent CutMix approach (i.e., swap entire subgraphs) and thus is less impacted by sub-optimal alignments. S-Mixup, Geo-Mix, and FGW-Mixup did not produce mixup graphs that interpolated between their inputs, and If-Mixup fell in between. This suggests that the (approximate) alignments being used by the latter methods are far from optimal; we provide more detail below.

**Result ④: Good interpolation properties were beneficial for empirical performance.** We now study to what extent interpolation properties correlate with empirical performance. Fig. 2 summarizes our results using (A3). As can be seen, all methods with high interpolation error as well as Emb-Mixup and G-Mixup (which do not perform pairwise interpolation of graphs) provided clearly inferior results compared to methods that interpolated better (GED-Mixup, SubMix, If-Mixup). This statement is statistically significant under all of our assumptions (A1)–(A3) ($p = 6.97 \times 10^{-03}, 3.52 \times 10^{-04}, 1.04 \times 10^{-09}$ resp.). These findings provide evidence that bad interpolation properties are detrimental for empirical performance. Nevertheless, even optimal interpolation did not lead to statistically significant improvements (see also Tab. 2 and Tab. 3), i.e., good interpolation is not sufficient.

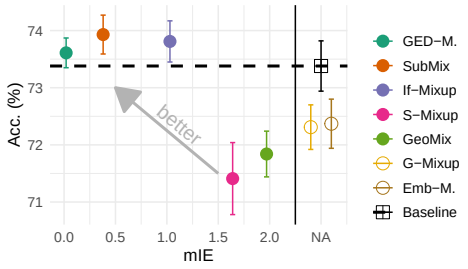
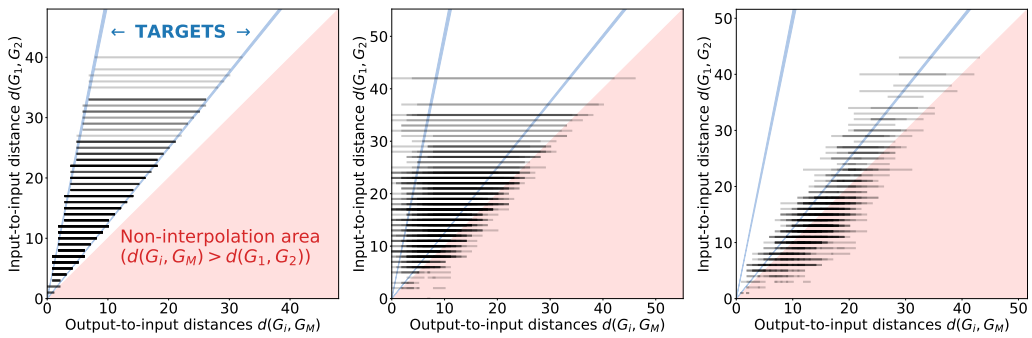


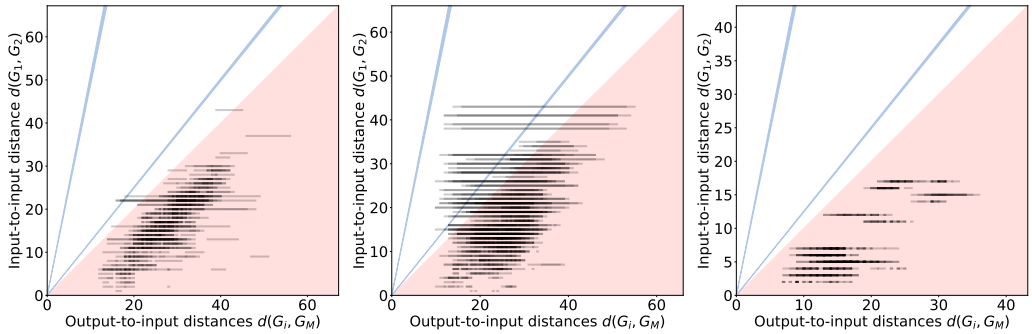

Figure 2: Mean interpolation error (mIE) and resulting test accuracy (%) along with standard errors under (A3, fixed data).

**Detailed analysis.** To shed some light into the graphs produced by each of the methods, we visualize properties of the mixup graphs on the MUTAG dataset[6] for a choice of $\lambda = 0.8 \pm \varepsilon$ in Fig. 3, including the resulting mIE score. The plot represents each augmented pair $(G_1, G_2, G_M, \lambda)$ by a (slightly transparent) horizontal line, where the height correspond to the distance between the input graphs ($y = d(G_1, G_2)$) and the start- and endpoint to the distance of the mixup graph to each input (from $x_1 = d(G_1, G_M)$ to $x_2 = d(G_2, G_M)$. The blue targets indicate the points where $d(G_1, G_M) = \lambda d(G_1, G_2)$ and $d(G_2, G_M) = (1-\lambda)d(G_1, G_2)$, i.e., $\mathrm{IC}_1$ and $\mathrm{IC}_2$, resp., are satisfied. Ideally, all lines start and end at their target. The area marked in red shows particularly troublesome

[6]The conclusions for the MUTAG dataset are representative for the other datasets as well; see App. I.




(a) GED-Mixup (mIE = 0.04) (b) SubMix (mIE = 0.62) (c) If-Mixup* (mIE = 1.31)




(d) FGW-Mixup (mIE = 2.51) (e) S-Mixup* (mIE = 3.37) (f) GeoMix* (mIE = 4.25)

Figure 3: Visualization of mixup graphs produced on the MUTAG dataset. Each horizontal line corresponds to an input pair and its mixup graph and should ideally start and end at the blue targets.

cases: if an endpoint falls into this area, the corresponding mixup graph has a larger distance to one of the inputs than the distance between the inputs themselves. In addition to not hitting the interpolation targets, such graphs cannot even be interpreted as an interpolation between their inputs (corresponds to the non-green area in Fig. 1).

As expected, GED-Mixup interpolated well between inputs. For SubMix, which also interpolated well, we can see that the reason it slightly fell behind GED-Mixup interpolation is that it typically over- or undershot interpolation targets. If-Mixup, which fell behind considerably, produced results that roughly satisfied one of the targets ($IC_2$) but not the other one ($IC_1$). The mixup graphs produced by all other methods could generally not be treated as interpolations (red area is almost always touched).

**Related work.** Note that some prior work also performed structural analysis of mixup graphs to some extent. In particular, Zeng et al. (2024) evaluate structural plausibility of their GeoMix method using the Gromov-Wasserstein distance, i.e., within the Gromov-Wasserstein space rather than the input space. Their conclusions heavily depend on the suitability of that space, which our results call into question. Moreover, Ling et al. (2022) analyze their S-Mixup method using a variant of GED (which ignores distance between inputs, for example); our result indicate that S-Mixup does not exhibit good interpolation properties even though it optimizes that variant. Both works are limited in that they analyzed their respective proposed methods only, whereas our work provides a more holistic view.

## 6 Conclusion

We performed an independent evaluation of in a unified experimental setup following established evaluation protocols and systematically analyzed the mixup graphs produced by existing mixup methods from an interpolation perspective. While we do believe that mixup can be beneficial for graph classification tasks, our experimental study did not provide evidence for its efficacy. However, it also did not provide evidence to the contrary. We did find evidence that high interpolation errors lead to inferior results, though, which indicates that interpolation properties should be taken into account in subsequent works.

## Reproducibility Statement

All of our work is reproducible. For this, we provided pseudo code for our analysis tool, GED-Mixup, both in the main text in Sec. 4.3 and in App. F with more details. Our experimental design for the interpolation analysis is described in Sec. 5, the experimental design for the empirical performance is summarized in Sec. 3 in the main text and described in detail in App. C. The source code required to reproduce our experiments is provided alongside this submission.

## REFERENCES


Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. Mixup: Beyond Empirical Risk Minimization. In *International Conference on Learning Representations*, 2018.

Sunil Thulasidasan, Gopinath Chennupati, Jeff A Bilmes, Tanmoy Bhattacharya, and Sarah Michalak. On Mixup Training: Improved Calibration and Predictive Uncertainty for Deep Neural Networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Linghui Meng, Jin Xu, Xu Tan, Jindong Wang, Tao Qin, and Bo Xu. Mixspeech: Data augmentation for low-resource automatic speech recognition. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7008–7012. IEEE, 2021.

Lichao Sun, Congying Xia, Wenpeng Yin, Tingting Liang, Philip Yu, and Lifang He. Mixup-Transformer: Dynamic Data Augmentation for NLP Tasks. In Donia Scott, Nuria Bel, and Chengqing Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3436–3440, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.305.

Tien Huu Do, Duc Minh Nguyen, Giannis Bekoulis, Adrian Munteanu, and Nikos Deligiannis. Graph convolutional neural networks with node transition probability-based message passing and DropNode regularization. *Expert Systems with Applications*, 174:114711, July 2021. ISSN 0957-4174. doi: 10.1016/j.eswa.2021.114711.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations*, September 2019.

Cédric Villani. *Optimal Transport: Old and New*, volume 338. Springer, 2008.

László Lovász and Balázs Szegedy. Limits of dense graph sequences. *Journal of Combinatorial Theory, Series B*, 96(6):933–957, November 2006. ISSN 0095-8956. doi: 10.1016/j.jctb.2006.05.002.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph Matching Networks for Learning the Similarity of Graph Structured Objects. In *Proceedings of the 36th International Conference on Machine Learning*, pages 3835–3845. PMLR, May 2019.

Ermin Omeragic and Vuk Duranović. [Re] G-Mixup: Graph Data Augmentation for Graph Classification. In *ML Reproducibility Challenge 2022*, 2023.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A Fair Comparison of Graph Neural Networks for Graph Classification. In *International Conference on Learning Representations*, September 2019.

Aymen Qabel, Sofiane Ennadir, Giannis Nikolentzos, Johannes Lutzeyer, Michail Chatzianastasis, Henrik Boström, and Michalis Vazirgiannis. Structure-Aware Antibiotic Resistance Classification Using Graph Neural Networks, October 2022.

Conghao Wang, Gaurav Asok Kumar, and Jagath C. Rajapakse. Drug discovery and mechanism prediction with explainable graph neural networks. *Scientific Reports*, 15(1):179, January 2025. ISSN 2045-2322. doi: 10.1038/s41598-024-83090-3.

David Buterez, Jon Paul Janet, Steven J. Kiddle, Dino Oglic, and Pietro Lió. Transfer learning with graph neural networks for improved molecular property prediction in the multi-fidelity setting. *Nature Communications*, 15(1):1517, February 2024. ISSN 2041-1723. doi: 10.1038/s41467-024-45566-8.

Yaan J. Jang, Qi-Qi Qin, Si-Yu Huang, Arun T. John Peter, Xue-Ming Ding, and Benoît Kornmann. Accurate prediction of protein function using statistics-informed graph networks. *Nature Communications*, 15(1):6601, August 2024. ISSN 2041-1723. doi: 10.1038/s41467-024-50955-0.

Karel van der Weg, Erinc Merdivan, Marie Piraud, and Holger Gohlke. TopEC: Prediction of Enzyme Commission classes by 3D graph neural networks and localized 3D protein descriptor. *Nature Communications*, 16(1):2737, March 2025. ISSN 2041-1723. doi: 10.1038/s41467-025-57324-5.

Kanchan Jha, Sriparna Saha, and Hiteshi Singh. Prediction of protein–protein interaction using graph neural networks. *Scientific Reports*, 12(1):8360, May 2022. ISSN 2045-2322. doi: 10.1038/s41598-022-12201-9.

Tristan Bilot, Nour El Madhoun, Khaldoun Al Agha, and Anis Zouaoui. A Survey on Malware Detection with Graph Representation Learning. *ACM Comput. Surv.*, 56(11):278:1–278:36, June 2024. ISSN 0360-0300. doi: 10.1145/3664649.

Soroor Motie and Bijan Raahemi. Financial fraud detection using graph neural networks: A systematic review. *Expert Systems with Applications*, 240:122156, April 2024. ISSN 0957-4174. doi: 10.1016/j.eswa.2023.122156.

Chengtai Cao, Fan Zhou, Yurou Dai, Jianping Wang, and Kunpeng Zhang. A Survey of Mix-based Data Augmentation: Taxonomy, Methods, Applications, and Explainability. *ACM Comput. Surv.*, 57(2):37:1–37:38, October 2024. ISSN 0360-0300. doi: 10.1145/3696206.

Chanwoo Park, Sangdoo Yun, and Sanghyuk Chun. A Unified Analysis of Mixed Sample Data Augmentation: A Loss Function Perspective. In *Advances in Neural Information Processing Systems*, October 2022.

Xin Jin, Hongyu Zhu, Siyuan Li, Zedong Wang, Zicheng Liu, Juanxi Tian, Chang Yu, Huafeng Qin, and Stan Z. Li. A Survey on Mixup Augmentations and Beyond, April 2025.

Aviv Shamsian, Aviv Navon, David W. Zhang, Yan Zhang, Ethan Fetaya, Gal Chechik, and Haggai Maron. Improved Generalization of Weight Space Networks via Augmentations. In *Proceedings of the 41st International Conference on Machine Learning*, pages 44378–44393. PMLR, July 2024.

Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. CutMix: Regularization Strategy to Train Strong Classifiers With Localizable Features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6023–6032, 2019.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In *Proceedings of the 38th International Conference on Machine Learning*, pages 10347–10357. PMLR, July 2021.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning*, pages 6438–6447. PMLR, 2019.

Alexandre Ramé, Rémy Sun, and Matthieu Cord. MixMo: Mixing Multiple Inputs for Multiple Outputs via Deep Subnetworks. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 803–813, October 2021. doi: 10.1109/ICCV48922.2021.00086.

Wenxuan Bao, Francesco Pittaluga, Vijay Kumar B G, and Vincent Bindschaedler. DP-Mix: Mixup-based Data Augmentation for Differentially Private Learning. *Advances in Neural Information Processing Systems*, 36:12154–12170, December 2023.

Difan Zou, Yuan Cao, Yuanzhi Li, and Quanquan Gu. The Benefits of Mixup for Feature Learning. In *Proceedings of the 40th International Conference on Machine Learning*, pages 43423–43479. PMLR, July 2023.

Hongyu Guo and Yongyi Mao. Interpolating graph pair to regularize graph classification. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, volume 37 of *AAAI'23/IAAI'23/EAAI'23*, pages 7766–7774. AAAI Press, February 2023. ISBN 978-1-57735-880-0. doi: 10.1609/aaai.v37i6.25941.

Hongyi Ling, Zhimeng Jiang, Meng Liu, Shuiwang Ji, and Na Zou. Graph Mixup with Soft Alignments. In *Proceedings of the 40th International Conference on Machine Learning*, September 2022.

Jaemin Yoo, Sooyeon Shim, and U Kang. Model-Agnostic Augmentation for Accurate Graph Classification. In *Proceedings of the ACM Web Conference 2022*, WWW '22, pages 1281–1291, New York, NY, USA, April 2022. Association for Computing Machinery. ISBN 978-1-4503-9096-5. doi: 10.1145/3485447.3512175.

Xiaotian Han, Zhimeng Jiang, Ninghao Liu, and Xia Hu. G-Mixup: Graph Data Augmentation for Graph Classification. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 8230–8248. PMLR, 2022.

Xinyu Ma, Xu Chu, Yasha Wang, Yang Lin, Junfeng Zhao, Liantao Ma, and Wenwu Zhu. Fused Gromov-Wasserstein Graph Mixup for Graph-level Classifications. *Advances in Neural Information Processing Systems*, 36:15252–15276, December 2023.

Zhichen Zeng, Ruizhong Qiu, Zhe Xu, Zhining Liu, Yuchen Yan, Tianxin Wei, Lei Ying, Jingrui He, and Hanghang Tong. Graph Mixup on Approximate Gromov–Wasserstein Geodesics. In *Forty-First International Conference on Machine Learning*, June 2024.

Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. Mixup for Node and Graph Classification. In *Proceedings of the Web Conference 2021*, WWW '21, pages 3663–3674, New York, NY, USA, June 2021. Association for Computing Machinery. ISBN 978-1-4503-8312-7. doi: 10.1145/3442381.3449796.

Titouan Vayer, Laetitia Chapel, Remi Flamary, Romain Tavenard, and Nicolas Courty. Fused Gromov-Wasserstein Distance for Structured Objects. *Algorithms*, 13(9):212, September 2020. ISSN 1999-4893. doi: 10.3390/a13090212.

Gabriel Peyré, Marco Cuturi, and Justin Solomon. Gromov-Wasserstein Averaging of Kernel and Distance Matrices. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2664–2672. PMLR, June 2016.

Lijun Chang, Xing Feng, Kai Yao, Lu Qin, and Wenjie Zhang. Accelerating Graph Similarity Search via Efficient GED Computation. *IEEE Transactions on Knowledge and Data Engineering*, 35(5): 4485–4498, May 2023. ISSN 1558-2191. doi: 10.1109/TKDE.2022.3153523.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A collection of benchmark datasets for learning with graphs, July 2020.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, February 2017.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations*, 2018.

Bernard L. Welch. The generalization of 'STUDENT'S'problem when several different population varlances are involved. *Biometrika*, 34(1-2):28–35, 1947.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 2623–2631, New York, NY, USA, July 2019. Association for Computing Machinery. ISBN 978-1-4503-6201-6. doi: 10.1145/3292500.3330701.

Alberto Sanfeliu and King-Sun Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(3):353–362, May 1983. ISSN 2168-2909. doi: 10.1109/TSMC.1983.6313167.

Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, August 2009. ISSN 2150-8097. doi: 10.14778/1687627.1687631.

Lijun Chang, Xing Feng, Xuemin Lin, Lu Qin, Wenjie Zhang, and Dian Ouyang. Speeding Up GED Verification for Graph Similarity Search. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 793–804, April 2020. doi: 10.1109/ICDE48307.2020.00074.

Jaeseung Heo, Seungbeom Lee, Sungsoo Ahn, and Dongwoo Kim. EPIC: Graph augmentation with edit path interpolation via learnable cost. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24, pages 4116–4126, Jeju, Korea, August 2024. ISBN 978-1-956792-04-1. doi: 10.24963/ijcai.2024/455.

Hongyu Guo and Yongyi Mao. ifMixup: Interpolating Graph Pair to Regularize Graph Classification, November 2022.

David B. Blumenthal, Nicolas Boria, Johann Gamper, Sébastien Bougleux, and Luc Brun. Comparing heuristics for graph edit distance computation. *The VLDB Journal*, 29(1):419–458, January 2020. ISSN 0949-877X. doi: 10.1007/s00778-019-00544-1.

Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 271–279, New York, NY, USA, May 2003. Association for Computing Machinery. ISBN 978-1-58113-680-7. doi: 10.1145/775152.775191.

A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, February 1991. ISSN 0022-2623. doi: 10.1021/jm00106a046.

Nils Kriege and Petra Mutzel. Subgraph matching kernels for attributed graphs. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 291–298, Madison, WI, USA, June 2012. Omnipress. ISBN 978-1-4503-1285-1.

Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. BRENDA, the enzyme database: Updates and major new developments. *Nucleic Acids Research*, 32(Database issue):D431–433, January 2004. ISSN 1362-4962. doi: 10.1093/nar/gkh081.

Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics (Oxford, England)*, 21 Suppl 1:i47–56, June 2005. ISSN 1367-4803. doi: 10.1093/bioinformatics/bti1007.

Pinar Yanardag and S.V.N. Vishwanathan. Deep Graph Kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1365–1374, New York, NY, USA, August 2015. Association for Computing Machinery. ISBN 978-1-4503-3664-2. doi: 10.1145/2783258.2783417.

Paul D. Dobson and Andrew J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, July 2003. ISSN 0022-2836. doi: 10.1016/s0022-2836(03)00628-4.

Matthias Fey and Jan Eric Lenssen. Fast Graph Representation Learning with PyTorch Geometric, April 2019.

Gavin C. Cawley and Nicola LC Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *The Journal of Machine Learning Research*, 11:2079–2107, 2010.

Yoshua Bengio. Practical Recommendations for Gradient-Based Training of Deep Architectures. In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade: Second Edition*, pages 437–478. Springer, Berlin, Heidelberg, 2012. ISBN 978-3-642-35289-8. doi: 10.1007/978-3-642-35289-8_26.

Yoshua Bengio and Yves Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of machine learning research*, 5(Sep):1089–1105, 2004.

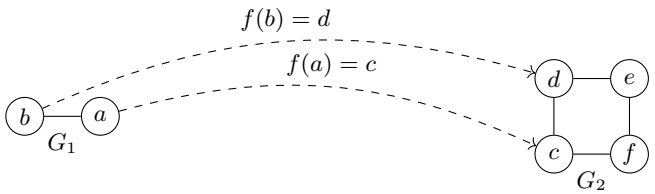
Jiaxuan You, Rex Ying, and Jure Leskovec. Design space for graph neural networks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, pages 17009–17021, Red Hook, NY, USA, December 2020. Curran Associates Inc. ISBN 978-1-7138-2954-6.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, September 1951. ISSN 0003-4851, 2168-8990. doi: 10.1214/aoms/1177729586.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1): 1929–1958, January 2014. ISSN 1532-4435.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456, Lille, France, July 2015. JMLR.org.




$$A_1 = \begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{array} \qquad A_2^{\text{unaligned}} = \begin{array}{c} \\ c \\ e \\ d \\ f \end{array} \begin{array}{c} \begin{array}{cccc} c & e & d & f \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \end{array} \qquad A_2^{\text{aligned}} = \begin{array}{c} \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccc} c & d & e & f \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Figure 4: The vertex mapping $f : V_1 \to V_2$ (dashed lines) maps the vertices $V_1$ of $G_1$ to the vertices $V_2$ of $G_2$. The vertex mapping $f$ is a full vertex mapping because $f(v_1)$ is well-defined for all $v_1 \in V_1$. The adjacency matrix $A_1$ corresponds to $G_1$, and both $A_2^{\text{unaligned}}$ and $A_2^{\text{aligned}}$ correspond to $G_2$. The matrix $A_1$ directly appears as a substructure in $A_2^{\text{aligned}}$ at the top left of the adjacency matrix, but not in $A_2^{\text{unaligned}}$.

# A Related Work

We first discuss vertex mappings and related concepts, and subsequently summarize existing mixup methods by relating them to vertex mappings.

## A.1 Preliminaries: Vertex Mappings, Edit Sets, Graph Edit Distance, Adjacency Matrices

Consider two input graphs $G_1$ and $G_2$, with vertex sets $V_1$ and $V_2$ as well as edge sets $E_1$ and $E_2$, respectively. Without loss of generality, assume that $|V_1| \leq |V_2|$. Two example graphs are shown in Fig. 4 (top). Note that the letters shown in the figure do not correspond to vertex labels (the graphs are unlabeled); we use them for expository reasons only.

A *vertex mapping* $f : V_1 \to V_2$ is an injective mapping from the vertices of $G_1$ to the vertices of $G_2$. Vertex mappings formalize the notion of an alignment of Sec. 2. One such mapping is shown in Fig. 4 (top).

Every vertex mapping $f$ induces an edit set, which is obtained by performing the edits required to obtain $G_2$ from $G_1$ by "transforming" each vertex $v_1 \in V_1$ (along with its neighborhood) to $f(v_1) \in V_2$ (likewise). For example, if $(v_1, u_1) \notin E_1$ but $(f(v_1), f(u_1)) \in E_2$, we include an insertion operation of edge $(v_1, u_1)$ to the edit set. The edit set induced by the vertex mapping shown in Fig. 4 is given by

$$\{\,\text{insert vertex } e, \text{insert vertex } f, \text{insert edge } (b, e), \text{insert edge } (a, f), \text{insert edge } (e, f)\,\}\,.$$

The induced edit set can be computed in asymptotically linear time with respect to the number of nodes and edges (cf. Alg. 1 in (Chang et al., 2020)).

A vertex mapping is *optimal* if the size of its induced edit set is as small as possible. The GED $d(G_1, G_2)$ is given by the size of the edit sets of optimal vertex mappings. State-of-the-art algorithms for exact GED computation such as (Chang et al., 2023; 2020; Blumenthal et al., 2020) (sometimes implicitly) produce an optimal vertex mapping and hence a corresponding edit set as a by-product. We exploit this fact in our implementation of GED-Mixup.

For example, the computational framework used by Chang et al. (2020) and its improved version (Chang et al., 2023) starts by constructing a prefix-shared search tree, where each path from the root to a leaf represents a *full vertex mapping* $f$, and each path from the root to a non-leaf represents a *partial vertex mapping* $f_p$, in which $f_p(v_1)$ is undefined for some $v_1 \in V_1$. Each node is associated with a *cost*, which constitutes a lower bound on (or, for leaves, the exact value of) the sizes of the induced edit sets obtained by all possible completions of the node's partial vertex mapping. The goal is to find a minimal-cost leaf, e.g., by using $A^*$ search. In order to obtain an efficient algorithm, lower bounds should (i) be as tight as possible so that pruning is effective and (ii) be efficiently computable.

We can alternatively represent vertex mappings by *vertex orderings*, i.e., an ordering of the vertices in $V_1$ and $V_2$, as well as by the corresponding *adjacency matrices*. For example, Fig. 4 shows adjacency matrix $A_1$ of $G_1$ for vertex ordering $(a, b)$. Likewise, $A_2^{\text{unaligned}}$ is the adjacency matrix of $G_2$ for vertex ordering $(c, e, d, f)$. The corresponding vertex mapping $f^{\text{unaligned}}$ is given by $f^{\text{unaligned}}(a) = c$ and $f^{\text{unaligned}}(b) = e$, i.e., the position of every vertex $v_1$ the ordering of $V_1$ matches the position of $f^{\text{unaligned}}(v_1)$ in the ordering of $V_2$. Vice versa, we can *align* the adjacency matrix of $G_2$ to the adjacency matrix $A_1$ with respect to a vertex mapping $f$. The corresponding adjacency matrix is shown as $A_2^{\text{aligned}}$.

Given two adjacency matrices $A_1$ and $A_2$, we can obtain an edit set by transforming $A_1$ to $A_2$ using (i) vertex insertions (inserting a zero row/column at the bottom/right of $A_1$), (ii) edge insertions (flip a 0 to a 1), (iii) edge deletions (flip a 1 to a 0), and (iv) vertex and edge relabelings (not shown).

We are now ready to describe existing graph mixup methods.

### A.2 Methods for Graph Mixup

We continue to use the setup and notation established in the previous section.

**If-Mixup (Guo and Mao, 2023).** If-Mixup uses adjacency matrices $A_1$ and $A_2$ obtained by a "default" vertex ordering of $V_1$ and $V_2$ present in the data. In more detail, it first pads $A_1$ (i.e., the smaller adjacency matrix) with zero rows and columns to obtain $A_1'$ (now of same size as $A_2$), and then performs linear mixup. The entries of the resulting adjacency matrix lie in $[0, 1]$ and can be interpreted as edge weights or edge existence probabilities (which are then fed into the GNN model as additional features). The result is heavily influenced by the vertex ordering present in the input data; e.g., for Fig. 4, it is less suitable when $A_2 = A_2^{\text{unaligned}}$ and more suitable when $A_2 = A_2^{\text{aligned}}$.

**S-Mixup (Ling et al., 2022).** S-Mixup can be seen as a variant of If-Mixup that aims to obtain a better alignment between the two adjacency matrices. Observe that the reordering operation involved in aligning an adjacency matrix $A_2$ to $A_1$ w.r.t. to vertex mapping $f$ can be expressed as $\boldsymbol{A}_2^{\text{aligned}} = \boldsymbol{P}\boldsymbol{A}_2\boldsymbol{P}^\top$, where $\boldsymbol{P}$ is a corresponding *alignment matrix* (which is a permutation matrix). S-Mixup uses a *soft alignment matrix* obtained from a Graph Matching Network (Li et al., 2019), which is trained on the available data. This soft alignment may not be optimal, but the hope is that it induces smaller edit sets than the "arbitrary" vertex mapping used by If-Mixup. Another difference is that S-Mixup, depending on the order of the two inputs, either produces a mixup graph with as many vertices as $G_1$ or a mixup graph with as many vertices as $G_2$.

**SubMix (Yoo et al., 2022).** SubMix is a mixup method inspired by CutMix (Yun et al., 2019). In CutMix, patches of images are cut and pasted between training examples, and the examples labels are mixed proportionally to the size of the patches. SubMix first samples a partial vertex ordering $V_1'$ of $V_1$ with a diffusion process (personalized page rank (Jeh and Widom, 2003)), likewise $V_2'$ of $V_2$. It then replaces the subgraph of $G_2$ induced by $V_2$ by the subgraph of $G_1$ induced by $V_1'$. Since the partial vertex orderings are obtained from a random diffusion process, the quality of the obtained mixup graph depends on chance.

**G-Mixup (Han et al., 2022).** G-Mixup follows a different approach. It first computes graphons (Lovász and Szegedy, 2006), each corresponding to a class and summarizing all graphs of that class present in the training data. In more detail, a graphon is a symmetric function $W : [0, 1]^2 \to [0, 1]$ and can be interpreted as a "probabilistic adjacency matrix" of graphs of arbitrary sizes. Each probabilistic graph is represented by a set $V$ of vertex positions (each in $[0, 1]$), and the edge existence probability between vertices $u, v \in V$ is given by $W(u, v)$. To obtain a graphon, G-Mixup orders the vertices of the training graphs by degree, i.e., it implicitly uses a custom vertex ordering. To perform mixup, G-Mixup interpolates the graphons of two classes using linear mixup, and then samples graphs from the resulting mixed graphon. Note that (Omeragic and Duranović, 2023) reported reproducibility problems for this method.

**FGW-Mixup (Ma et al., 2023) and GeoMix (Zeng et al., 2024).** Both FGW-Mixup and GeoMix rely on the Gromov-Wasserstein distance (Vayer et al., 2020) from the theory of optimal transport (Vil-

lani, 2008). FGW-Mixup uses the Fused-Gromov-Wasserstein distance, which incorporates graph structure and features and aims to compute barycenters, wheres GeoMix uses the plain Gromov-Wasserstein distance and relies on geodesics Peyré et al. (2016). In both cases, mixup takes place in the Gromov-Wasserstein space. The optimal coupling between the input graphs obtained in the minimization of the (Fused-)Gromov-Wasserstein distance corresponds to a vertex mapping, i.e., these methods also aim to find a suitable mapping. Both FGW-Mixup and GeoMix are computationally expensive and hence make use of approximation algorithms.

**Embedding Mixup.** Mixup can also be performed in the embedding space of the GNN model (Wang et al., 2021), along the lines of manifold mixup (Verma et al., 2019). To do so, the current GNN model is used to compute embeddings of the two input graphs, and linear mixup is performed subsequently. Note that in contrast to the methods discussed before, embedding mixup does not generate a mixup graph. It also does not use an underlying vertex mapping, as the embeddings being interpolated are neural representations of entire input graphs.

**EPIC (Heo et al., 2024).** Edit Path Interpolation via Learnable Cost (EPIC) is based on the GED and associated edit paths between graphs, akin to our GED-Mixup baseline of Sec. 4.3. EPIC learns a cost model that aims to quantify the "importance" of specific edit operations from training data. GEDs are computed using this cost model and approximation algorithms. GED-Mixup is a simplified variant of EPIC in that it (i) uses unit edit costs (and thus does not require learning) and (ii) uses exact GED. We did not consider EPIC in our experimental study as there was no source code available.

## B Dataset Statistics

All datasets are obtained through TUDataset (Morris et al., 2020). Key statistics are shown in Tab. 5.

Table 5: Dataset Statistics

| | **MUTAG** | **ENZYMES** | **IMDB-BINARY** | **PROTEINS** |
|---|---|---|---|---|
| Domain | Molecules | Bioinformatics | Social Networks | Bioinformatics |
| Graphs | 188 | 600 | 1000 | 1113 |
| Classes | 2 | 6 | 2 | 2 |
| Avg. Nodes | 17.93 | 32.63 | 19.77 | 39.06 |
| Avg. Edges | 19.79 | 62.14 | 96.53 | 72.82 |
| Node Labels | ✓ | ✓ | – | ✓ |
| Edge Labels | ✓ | – | – | – |
| Node Attr. | – | ✓ (18) | – | ✓ (1) |
| Geometry | – | – | – | – |
| Edge Attr. | – | – | – | – |
| References | (1991; 2012) | (2004; 2005) | (2015) | (2005; 2003) |

## C Experimental Details

This section outlines the details of the experimental setup that are summarized in Sec. 3 in the main text.

**Datasets.** We considered the datasets discussed in App. B.

**Models.** We considered the GCN (Kipf and Welling, 2017) and GIN (Xu et al., 2018) models in our structure. Both are simple, commonly used models and have sufficient representational capacity for the tasks we consider here. I.e., they did overfit in our experiments, a problem that mixup aims to alleviate.

**Mixup methods.** We considered all methods discussed in Sec. 2 for which open-source implementations were available, i.e., If-Mixup (Guo and Mao, 2023), S-Mixup (Ling et al., 2022), SubMix (Yoo et al., 2022), G-Mixup (Han et al., 2022), FGW-Mixup (Ma et al., 2023), GeoMix (Zeng et al., 2024), and Emb-Mixup (Wang et al., 2021).

**Training.** Given a labeled training set and a choice of hyperparameters, we trained each model in a common training pipeline based on PyTorch Geometric (Fey and Lenssen, 2019) and Optuna (Akiba et al., 2019), using cross entropy loss. To perform mixup, we used the original implementations from prior work as well as our implementation of GED-Mixup. In each epoch, we generated a fixed number of mixup graphs (a hyperparameter) and added them to the training data for this epoch. We followed the common approach of producing mixup graphs by mixing two randomly chosen graphs/labels from the training data, using mixup ratio $\lambda \sim \mathrm{Beta}(\alpha, \alpha)$ (for hyperparameter $\alpha \in \mathbb{R}^+$).

**Methodology.** We followed Errica et al. (2019), who describe key criteria for evaluating methods for graph classification tasks. This includes (i) nested cross-validation (CV) for model selection (inner CV for hyperparameter search) and model assessment (outer CV for test), (ii) repeat model assessment multiple times to account for training randomness (e.g., model initialization), and (iii) publish source code and ensure reproducibility. These criteria are well-established in machine learning (Cawley and Talbot, 2010; Bengio, 2012; Bengio and Grandvalet, 2004) and particularly important for graph classification tasks due to small dataset sizes and a lack of predefined train-test splits.

We emphasize these points because we found that prior studies often did not fully adhere to such evaluation standards. For instance, some studies (Guo and Mao, 2023; Yoo et al., 2022; Wang et al., 2021) followed the evaluation protocol of Xu et al. (2018), despite its problematic use of validation rather than test performance (Errica et al., 2019). Other studies (Ling et al., 2022; Han et al., 2022) used holdout validation instead of a cross-validation or use cross-validation only for model selection but not for model assessment (Ma et al., 2023). While most methods provide source code of the proposed method (except for (Guo and Mao, 2023; Heo et al., 2024)), source code for other key aspects such as model selection is missing. Adding to these points, statistical significance was rarely assessed so that it is not clear whether reported improvements are real. As we will see, our study answers this negatively, i.e., most methods failed to produce statistically significant improvements under a rigorous evaluation.

**Hyperparameters.** We used training hyperparameters (such as the learning rate or optimizer), model architecture hyperparameters (such as the number of layers, hidden dimensionalities, or dropout probability), and mixup hyperparameters (such as the number of added mixup graphs or sampling distribution of the mixup ratio $\lambda$). Tab. 6 summarizes our hyperparameters and search space. For GCN/GIN, we followed You et al. (2020). For specific mixup methods, we used the hyperparameter values or search spaces suggested in the code or the publication. Descriptions and rationales are given below the table.

**Tuning.** Given a training and a validation split, we sampled 10 hyperparameter configurations randomly and subsequently $n$ configurations using Bayesian optimization with TPE (Bergstra et al., 2011). Model selection was performed by mean validation accuracy (over the inner CV folds). We first tuned strong baseline models without mixup in this fashion (using $n = 90$), and subsequently tuned just the mixup hyperparameters (separately for each mixup method, using $n = 10$) in the same way. This ensures fairness, as all mixup methods used the same, well-performing model architecture.

**Metrics.** We used 5-fold nested cross-validation throughout, three repeated training runs for model assessment, and report mean classification accuracy on the test splits as a metric. We also report standard errors and statistical significance compared to the no-mixup baseline using Welch's two-sided $t$-test (Welch, 1947) with a significance level of $0.05$.

**Hardware and computational cost.** We required approximately 144 GPU hours per model and dataset to determine the GNN hyperparameters and performance. We required approximately 96 GPU hours to determine hyperparameters and performance for each mixup method and dataset. We used NVIDIA RTX 2080Ti and NVIDIA RTX A6000 GPUs supported by various generations of either Intel Xeon CPUs (such as E2640 v2, E5-2640 v3, E5-2698 v4, and Silver 4114) or various generations of AMD EPYC CPUs (such as 7413, 9474F, and 7713P).

**Software.** Our experimental pipeline is implemented with PyTorch Geometric (Fey and Lenssen, 2019) (MIT License) and Optuna (Akiba et al., 2019) (MIT License). We used the original mixup implementations whenever available:

- Emb-Mixup: (own implementation)
- FGW-Mixup: https://github.com/ArthurLeoM/FGWMixup (no license)
- GED-Mixup: (own implementation)

Table 6: Hyperparameter search space

| | **Hyperparameter** | **Search space** |
|---|---|---|
| Training | Optimizer | Adam (2015), SGD (1951) |
| | Learning rate | $10^{-5}$ to $10^{-1}$ (log scale) |
| | Batch size | $\{8, 16, 32, 64, 128, 256\}$ |
| | Dropout probability (2014) | $[0, 0.5]$ |
| | Early stopping[1] | $\{1, \ldots, 1000\}$ |
| GCN/GIN | No. pre-processing layers | $1, 2, 3$ |
| | No. convolutional layers | $2, 4, \ldots, 8$ |
| | No. post-processing layers | $1, 2, 3$ |
| | Embedding size | 32 to 256 (log scale) |
| | Readout | Mean, Max, Sum |
| | Normalization | None, Batch norm (2015) |
| Mixup* | Augmentation ratio[2] | $[0.2, 2.0]$ |
| | Keep original graphs?[3] | Yes or no |
| | Mixup ratio parameter $\alpha$[4] | $\{0.1, 0.3, 0.5, 1.0, 5.0\}$ |
| FGW-Mixup | FGW-alpha[5] | $\{0.05, 0.5, 0.95\}$ |
| | $\rho$[6] | $\{0.1, 1, 10\}$ |
| S-Mixup | GMNet batch size[7] | $\{8, 64, 128\}$ |
| | No. GMNet layers[7] | $\{4, 5, 6\}$ |
| SubMix | Subgraph size[8] | $\{0.2, 0.4, 0.6\}$ |

* Used by all mixup methods unless stated otherwise.

[1] We determine the number of "early-stopping" epochs that leads to the best validation result, and use it when we retrain on the entire four folds for testing on the remaining fold. We do this so that test data is not used for early stopping and hence not leaked.

[2] Fraction of generated mixup graphs per epoch w.r.t. the size of the training data. Here we force each mixup method to add a non-trivial fraction (20%) of mixup graphs during training. We do this since we are primarily interested in whether mixup is effective and not in hyperparameter choices that do not actually add mixup graphs.

[3] Whether to include non-mixup graphs during training.

[4] Mixup ratio $\lambda$ is sampled from $\mathrm{Beta}(\alpha, \alpha)$. Does not apply to SubMix.

[5] Trade-off parameter between Gromov-Wasserstein cost on graph structure and Wasserstein cost on node features (cf. (Ma et al., 2023)).

[6] Step size hyperparameter in mirror descent (cf. (Ma et al., 2023)).

[7] Hyperparameter of the graph matching network (cf. (Li et al., 2019)).

[8] Relative size of the selected subgraphs (cf. (Yoo et al., 2022)).

- GeoMix: https://github.com/zhichenz98/GeoMix-ICML24 (no license)
- G-Mixup: https://github.com/ahxt/g-mixup (no license)
- If-Mixup: (own implementation)
- SubMix: https://github.com/snudatalab/GraphAug (custom license available under the specified URL)
- S-Mixup: https://github.com/divelab/DIG (GPL-3.0 license)
- GED and edit set computation: AStar-BMao (Chang et al., 2023) available from GitHub https://github.com/LijunChang/Graph_Edit_Distance (MIT License). We modified the code to additionally provide vertex mappings.

## D Pooled Analysis

We estimate standard errors under assumptions (A1)–(A3) discussed in Sec. 3. We start with (A1) and subsequently discuss (A2)–(A3).

**Observations.** Consider an arbitrary but fixed mixup method. In our evaluation, we considered

- Two model architectures $\mathcal{M} = \{\,\text{GCN}, \text{GIN}\,\}$,
- Four data distributions $\mathcal{D} = \{\,\text{MUTAG}, \text{ENZYMES}, \text{IMDB-BINARY}, \text{PROTEINS}\,\}$,
- Five folds $\mathcal{F}_d = \{\, f_1^d, \ldots, f_5^d \,\}$ per data distribution $d \in \mathcal{D}$, each consisting of a training and a test split, and
- Three runs $\mathcal{R}_{mdf} = \{\, r_1^{mdf}, r_2^{mdf}, r_3^{mdf} \,\}$ per model architecture $m \in \mathcal{M}$, data distribution $d \in \mathcal{D}$, and fold $f \in \mathcal{F}_d$.

We then observe the accuracies $A_{mdfr} \in [0, 1]$ of run $r$ on fold $f$ for data distribution $d$ and model architecture $m$.

**Assumption (A1, standard).** In what follows, we use capital letters to refer to random variables (e.g., $D$ for a random data distribution) and small as well as calligraphic letters to refer to concrete realizations (e.g., $d$ for a concrete realization of $D$ and $\mathcal{D}$ for multiple such realizations). Under (A1), we assume that

(A1-M) model architectures are drawn from a distribution $p(M)$ of "real-world" model architectures,

(A1-D) data distributions are drawn from a distribution $p(D)$ of "real-world" data distributions,

(A1-F) folds $F$ are drawn from a fold distribution $p(F_D)$ obtained by sampling each element of each of the training/test splits independently from data distribution $D$,

(A1-R) runs $R$ are drawn from a run distribution $p(R_{MDF})$ determined by the random choices made during training (such as randomness in initialization, batch construction, or mixup).

These assumptions allow us to estimate the impact of mixup beyond the concrete datasets and model architectures used in this study.

Note that we make a key simplifying assumption here: we treat each observation $A_{mdfr}$ as an independent realization of $A_{MDFR}$. By doing so, we ignore that in our implementation, (i) all 3 runs use the same fold and hyperparameters, and (ii) all 5 folds are obtained by cross-valuation and from a single dataset. This may lead to an underestimation of variance Bengio and Grandvalet (2004). We proceed this way to keep the cost of the experimental study controlled, and we need this simplifying assumption to make analysis feasible.

**Estimators.** We estimate the expected accuracy $A_{mdf} = \mathbb{E}_{R\sim p(R_{mdf})}[A_{mdfR}]$ of model architecture $m$ on fold $f$ for data distribution $d$ by the sample mean, i.e.,

$$\hat{A}_{mdf} = \frac{1}{|\mathcal{R}_{mdf}|} \sum_{r \in \mathcal{R}_{mdf}} A_{mdfr}.$$

Likewise, the expected accuracy $A_{md} = \mathbb{E}_{F\sim p(F_d)}[A_{mdF}]$ of model architecture $m$ on data distribution $d$ is estimated as

$$\hat{A}_{md} = \frac{1}{|\mathcal{F}_d|} \sum_{f\in\mathcal{F}_d} \hat{A}_{mdf}.$$

This estimate is shown in Tab. 2 for various mixup methods. Finally, the estimates for the expected performance $A_m \sim \mathbb{E}_{D\sim p(D)}[A_{mD}]$ of model architecture $m$ and expected overall performance $A = \mathbb{E}_{M\sim p(M)}[A_M]$ are obtained similarly, i.e.,

$$\hat{A}_m = \frac{1}{|\mathcal{D}|} \sum_{d\in\mathcal{D}} \hat{A}_{md}$$

$$\hat{A} = \frac{1}{|\mathcal{M}|} \sum_{m\in\mathcal{M}} \hat{A}_m.$$

The latter estimate is shown in Tab. 3 for various mixup methods.

**Standard errors.** We now derive the standard errors of each of the above estimators under (A1). First, we estimate the variance $\sigma^2_{mdf} = \mathrm{Var}_{R\sim p(R_{mdf})}[A_{mdfR}]$ by the sample variance, i.e.,

$$\hat{\sigma}^2_{mdf} = \frac{1}{|\mathcal{R}_{mdf}| - 1} \sum_{r\in\mathcal{R}_{mdf}} (A_{mdfr} - \hat{A}_{mdf})^2.$$

For the variance $\sigma^2_{md} = \mathrm{Var}_{F\sim p(F_d)}[A_{mdF}]$, we use the law of total variance to obtain

$$\sigma^2_{md} = \underbrace{\mathbb{E}_{F\sim p(F_d)}[\mathrm{Var}(A_{mdF}|F)]}_{\text{“within-fold”}} + \underbrace{\mathrm{Var}_{F\sim p(F_d)}(\mathbb{E}[A_{mdF}|F])}_{\text{“between-folds”}}.$$

We estimate the first part by the mean within-fold variance estimate and the second part by the sample variance of estimates across folds:

$$\hat{\sigma}^2_{md} = \underbrace{\frac{1}{|\mathcal{F}|} \sum_{f\in\mathcal{F}_d} \hat{\sigma}^2_{mdf}}_{\text{“within-fold”}} + \underbrace{\frac{1}{|\mathcal{F}| - 1} \sum_{f\in\mathcal{F}_d} (\hat{A}_{mdf} - \hat{A}_{md})^2}_{\text{“between-folds”}}.$$

Using the simplifying assumption of independent observations discussed above, we estimate the standard error of $\hat{A}_{md}$ by

$$\widehat{\mathrm{SE}}(\hat{A}_{md}) = \sqrt{\frac{\hat{\sigma}^2_{md}}{n_{md}}},$$

where $n_{md} = \sum_{f\in\mathcal{F}_d} \sum_{r\in\mathcal{R}_{mdf}} 1 = 15$. This standard error is shown in Tab. 2.

Using the same approach, we obtain an estimate of the variance $\sigma^2_m = \mathrm{Var}_{D\sim p(D)}[A_{mD}]$ as

$$\hat{\sigma}^2_m = \underbrace{\frac{1}{|\mathcal{D}|} \sum_{d\in\mathcal{D}} \hat{\sigma}^2_{md}}_{\text{“within-dataset”}} + \underbrace{\frac{1}{|\mathcal{D}| - 1} \sum_{d\in\mathcal{D}} (\hat{A}_{md} - \hat{A}_m)^2}_{\text{“between-datasets”}}$$

and of variance $\sigma^2 = \mathrm{Var}_{M\sim p(M)}[A_M]$ as

$$\hat{\sigma}^2 = \underbrace{\frac{1}{|\mathcal{M}|} \sum_{m\in\mathcal{M}} \hat{\sigma}^2_m}_{\text{“within-model”}} + \underbrace{\frac{1}{|\mathcal{M}| - 1} \sum_{m\in\mathcal{M}} (\hat{A}_m - \hat{A})^2}_{\text{“between-models”}}.$$

We estimate the standard error of $\hat{A}$ by

$$\widehat{\mathrm{SE}}(\hat{A}) = \sqrt{\frac{\hat{\sigma}^2}{n}},$$

where $n = \sum_{m\in\mathcal{M}} \sum_{d\in\mathcal{D}} \sum_{f\in\mathcal{F}_d} \sum_{r\in\mathcal{R}_{mdf}} 1 = 120$. This standard error is shown in the (A1) column of Tab. 3.

**Assumption (A2, fixed distribution).** For (A2), we make the stronger assumption that $\mathcal{M}$ and $\mathcal{D}$ are not samples (from a distribution of model architectures and a distribution of data distributions, respectively) but the entire population of interest. Then $A_m = \frac{1}{|\mathcal{D}|}\sum_d A_{md}$ and $A = \frac{1}{|\mathcal{M}|}\sum_{m\in\mathcal{M}} A_m$ become means (instead of being expected values). As a consequence, the "between-datasets" and the "between-models" terms in our estimate of $\hat{\sigma}_m^2$ and $\hat{\sigma}^2$, respectively, vanish. The corresponding standard errors are shown in the (A2) column of Tab. 3.

**Assumption (A3, fixed data).** For (A3), we make assumption (A2) plus the additional assumption that the folds in $\mathcal{F}_d$ are not samples (from the data distribution $d$) but the entire population of interest. Then $A_{md} = \frac{1}{|\mathcal{F}_d|}\sum_{f\in\mathcal{F}_d} A_{mdf}$ becomes a mean (instead of being an expected value). As a consequence, the "between-folds" term in our estimate of $\hat{\sigma}_{md}^2$ vanishes as well. The corresponding standard errors are shown in the (A3) column of Tab. 3.

## E Proof of Prop. 1

**Proposition 1.** *For any distance metric $d(a,b) = \|a-b\|$ induced by a norm $\|\cdot\|$ on the input/label vector space (over $\mathbb{R}$), linear mixup of Eq. (1) satisfies IC for inputs/labels and we have*

$$\mathrm{AIE}_d(x_M) = \mathrm{IE}_d(x_M) = 0 \quad / \quad \mathrm{AIE}_d(y_M) = \mathrm{IE}_d(y_M) = 0.$$

*Proof.* Consider inputs $x_1$ and $x_2$, any $\lambda \in [0,1]$, and any norm $\|\cdot\|$ on the input space. Linear mixup produces the result

$$x_M = x_1 + \lambda(x_2 - x_1).$$

We have

$$\begin{aligned} d(x_1, x_M) &= \|x_1 - x_M\| \\ &= \|x_1 - (x_1 + \lambda(x_2 - x_1))\| \\ &= \|-\lambda(x_2 - x_1)\| \\ &= \lambda\,\|(x_2 - x_1)\| \\ &= \lambda \cdot d(x_1, x_2), \end{aligned}$$

which proves $\mathrm{IC}_1$. The same arguments can be made for $\mathrm{IC}_2$ and as well as $y_M$ so that all interpolation errors are zero as claimed. □

## F Details of GED-Mixup

Alg. 2 shows a slightly more detailed version of Alg. 1. In our implementation, we use AStar-BMao (Chang et al., 2023), an algorithm for GED computation, and modified it to also output a corresponding edit set $F$.

In contrast to Alg. 1, Alg. 2 does not first sample an edit path and subsequently generate a mixup graph. Instead, it generates $G_M$ directly.

Akin to the discussion in Sec. 4.3, we consider an edit path as *valid* if its operations (when applied in order) each satisfy: (i) an edge can only be inserted when its source and target vertex are present, (ii) a vertex can only be removed when it does not have an incident edge, (iii) label substitutions are only possible for vertices/edges present in the graph, and (iv) when both $G_1$ and $G_2$ are connected, so is $G_M$. We check for conditions (i)—(iii) as we go: AdmissibleMixup repeatedly samples a not-yet-used and *admissible* operation—i.e., an operation satisfying (i)—(iii)—from $F$ and returns the resulting mixup graph. If the mixup graph also satisfies (iv), we output it, otherwise we repeat the sampling process.[7]

[7] Assume $G_1$ and $G_2$ are connected. For some edit sets, every admissible edit path of length $\mathrm{round}(\lambda|F|)$ produces a mixup graph that is not connected. This problem can be fixed by allowing one more or one less edit operation. In our implementation, we use a simpler approach and abort generation if we do not obtain a valid mixup graph after 10 repetitions. This is not a problem in practice, as we can simply sample a new value for $\lambda$ or a new set of graphs to mixup.

**Algorithm 2** GED-Mixup of Alg. 1 in more detail

**Require:** Graphs $G_1, G_2$; mixup ratio $\lambda \in [0, 1]$
**Ensure:** Mixup graph $G_M$

$F^* \leftarrow$ a minimal edit set between $G_1$ and $G_2$, obtained by AStar-BMao (Chang et al., 2023)
**repeat**
  $G_M \leftarrow \textsc{AdmissibleMixup}(G_1, F^*, \text{round}(\lambda|F|))$
**until** $G_M$ is connected **or** $G_1$ is not connected **or** $G_2$ is not connected
**return** $G_M$

**function** $\textsc{AdmissibleMixup}(G_1, F, n)$
  $G_M \leftarrow G_1$
  **for** $i \leftarrow 1, \ldots, n$ **do**
    $f \leftarrow$ sample an edit operation from $F$ that is admissible on $G_M$ ▷ App. F
    $G_M \leftarrow \text{apply}(G_M, f)$
    $F \leftarrow F \setminus \{ f \}$
  **end for**
  **return** $G_M$
**end function**

Note that our approach to obtain an edit paths is rather naive: The obtained path is "random" and largely ignores locality of edit operations. Since we view GED-Mixup as a baseline, however, we did not explore this further.

## G Proof of Prop. 2

**Proposition 2.** *GED-Mixup (Alg. 1) interpolates optimally w.r.t. $d_{GED}$, and it holds*

$$\text{AIE}(G_M) = 2 \cdot |\text{round}(\lambda \cdot d_{GED}(G_1, G_2)) - \lambda \cdot d_{GED}(G_1, G_2)| \leq 1.$$

*Proof.* Consider inputs $G_1$ and $G_2$, let $P$ be the edit path chosen by Alg. 1. By construction, we have

$$\text{apply}(G_1, P) \cong G_2 \quad \text{and} \quad d_{12} \stackrel{\text{def}}{=} d_{\text{GED}}(G_1, G_2) = |P|.$$

Denote by

$$d_{M1} = \lambda \cdot |P| = \lambda \cdot d_{12}$$

the interpolation target ($\text{IC}_1$) and by $P_\lambda$ the first $\text{round}(d_{M1})$ edit operations in $P$. Alg. 1 produces

$$G_M = \text{apply}(G_1, P_\lambda)$$

and it holds that

$$d(G_M, G_1) = \text{round}(d_{M1})$$

To see this, first observe that $P_\lambda$ contains $\text{round}(d_{M1})$ edit operations so that $d(G_1, G_M) \leq \text{round}(d_{M1})$. Now suppose for contradiction that $d(G_1, G_M) < \text{round}(d_{M1})$ and let $P'_\lambda$ be a corresponding edit path. By replacing $P_\lambda$ by $P'_\lambda$ in $P$, we obtain an edit path $P'$ with $|P'| < |P|$ and $\text{apply}(\text{G}_1, \text{P'}) \cong G_2$, contradicting $P$'s property of being a shortest edit path from $G_1$ to $G_2$.

Denote by

$$d_{M2} = (1 - \lambda) \cdot |P| = (1 - \lambda) \cdot d_{12}$$

the interpolation target ($\text{IC}_2$). Using a similar argument as above, we obtain

$$d(G_M, G_2) = \text{round}(d_{M2})$$

Optimality now follows since GEDs are always integer-valued and the GEDs of $\text{round}(d_{M1})$ and $\text{round}(d_{M2})$ obtained by Alg. 1 are the integer values closest to their targets $d_{M1}$ and $d_{M2}$, respectively.

Using the facts that $0 \leq d_{M1} \leq d_{12}$ and $d_{M2} = d_{12} - d_{M1}$, we obtain

$$\begin{aligned}
\text{AIE}(G_M) &= |d_{\text{GED}}(G_M, G_1) - \lambda \cdot d_{\text{GED}}(G_1, G_2)| + |d_{\text{GED}}(G_M, G_2) - (1-\lambda) \cdot d_{\text{GED}}(G_1, G_2)| \\
&= |\text{round}(d_{M1}) - d_{M1}| + |\text{round}(d_{M2}) - d_{M2}| \\
&= |\text{round}(d_{M1}) - d_{M1}| + |\text{round}(d_{12} - d_{M1}) - (d_{12} - d_{M1})| \\
&= |\text{round}(d_{M1}) - d_{M1}| + |d_{12} - \text{round}(-d_{M1}) - d_{12} + d_{M1}| \\
&= |\text{round}(d_{M1}) - d_{M1}| + |\text{round}(-d_{M1}) + d_{M1}| \\
&= 2 \cdot |\text{round}(d_{M1}) - d_{M1}| \\
&\leq 1,
\end{aligned}$$

where the last equality is obtained by considering the two cases (a) $d_{M1}$ is rounded down and (b) $d_{M1}$ is rounded up separately. This proves the claimed interpolation error. □

## H Additional Results for Mean Interpolation Error

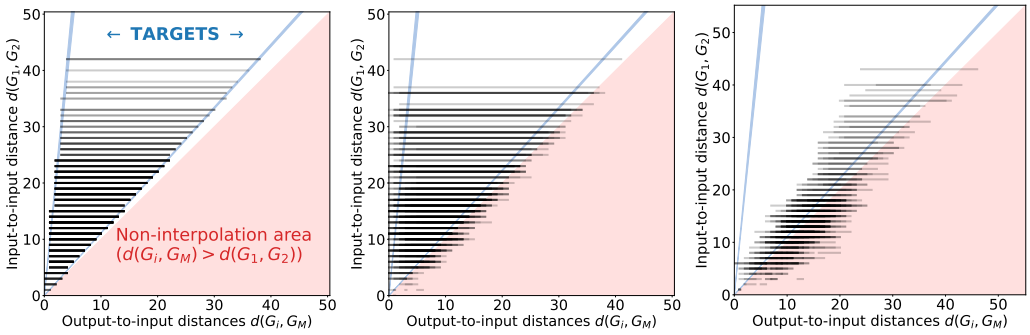
Tab. 7 contains results for multiple mixup ratios (in comparison to Tab. 4 from the main section which only includes a summary). We found that for GED-Mixup, the mIE was stable across different mixup ratios, whereas for other methods (such as If-Mixup, SubMix or S-Mixup), the mIE decreased with increasing mixup ratio.

Table 7: Detailed comparison of mean interpolation error (mIE) obtained by various methods and datasets (lower is better and $\geq 2$ is particularly bad). We were unable to generate graphs with FGW-Mixup on some datasets (denoted with –).

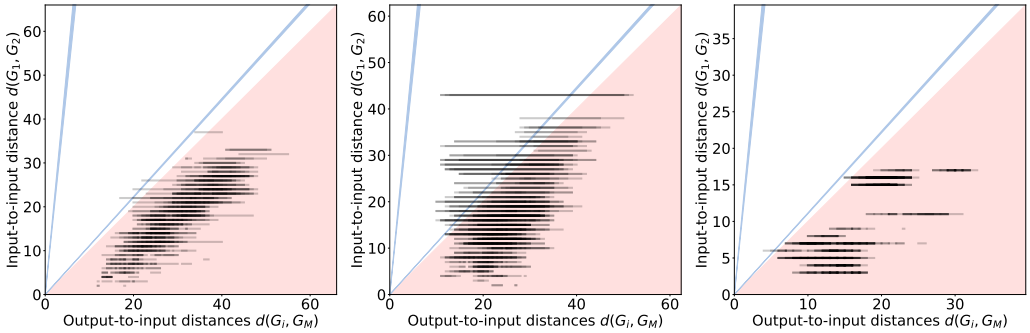
| **Method** | $\lambda$ | **MUTAG** | **ENZYMES** | **IMDB-BINARY** | **PROTEINS** | **Average** |
|---|---|---|---|---|---|---|
| GED-Mixup | Avg. | 0.05 | 0.01 | 0.01 | 0.01 | 0.02 |
| | 0.5 | 0.06 | 0.01 | 0.02 | 0.02 | 0.03 |
| | 0.8 | 0.04 | 0.01 | 0.01 | 0.01 | 0.02 |
| | 0.9 | 0.04 | 0.01 | 0.01 | 0.01 | 0.02 |
| SubMix | Avg. | 0.45 | 0.28 | 0.49 | 0.30 | 0.38 |
| | 0.5 | – | 0.53 | 0.94 | 0.57 | 0.68 |
| | 0.8 | 0.62 | 0.21 | 0.35 | 0.22 | 0.35 |
| | 0.9 | 0.28 | 0.11 | 0.18 | 0.10 | 0.17 |
| If-Mixup* | Avg. | 1.57 | 0.69 | 0.86 | 1.00 | 1.03 |
| | 0.5 | 1.91 | 0.74 | 1.64 | 1.26 | 1.39 |
| | 0.8 | 1.31 | 0.72 | 0.61 | 0.91 | 0.89 |
| | 0.9 | 1.49 | 0.61 | 0.33 | 0.82 | 0.81 |
| S-Mixup* | Avg. | 3.56 | 0.95 | 1.32 | 0.72 | 1.64 |
| | 0.5 | 4.03 | 1.17 | 1.52 | 1.16 | 1.97 |
| | 0.8 | 3.37 | 0.97 | 1.10 | 0.56 | 1.50 |
| | 0.9 | 3.28 | 0.69 | 1.34 | 0.44 | 1.44 |
| GeoMix* | Avg. | 4.31 | 1.32 | 0.66 | 1.60 | 1.97 |
| | 0.5 | 4.85 | 1.54 | 0.81 | 1.67 | 2.22 |
| | 0.8 | 4.25 | 1.24 | 0.55 | 1.80 | 1.96 |
| | 0.9 | 3.85 | 1.19 | 0.61 | 1.34 | 1.75 |
| FGW-Mixup | Avg. | 2.76 | – | – | – | 2.76 |
| | 0.5 | 1.95 | – | – | – | 1.95 |
| | 0.8 | 2.51 | – | – | – | 2.51 |
| | 0.9 | 3.82 | – | – | – | 3.82 |

# I Additional Examples of Interpolation Errors

This section contains plots that visualize the interpolation error mIE for further datasets and mixup ratios $\lambda$. In line with the discussion of Fig. 3 in the main paper, GED-Mixup interpolates well between inputs. The second best results are often obtained by SubMix, If-Mixup, or S-Mixup (except on MUTAG). Tab. 4 in the main section and Tab. 7 provide summaries beyond the example mixup graphs shown in the plots.

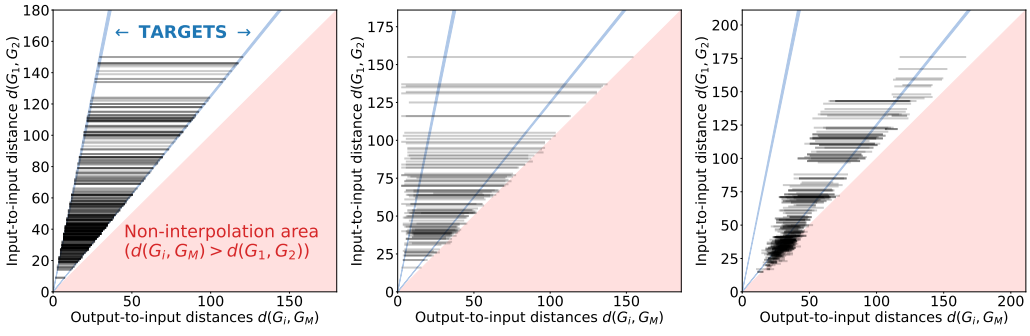



(a) GED-Mixup (mIE = 0.04) (b) SubMix (mIE = 0.28) (c) If-Mixup* (mIE = 1.49)



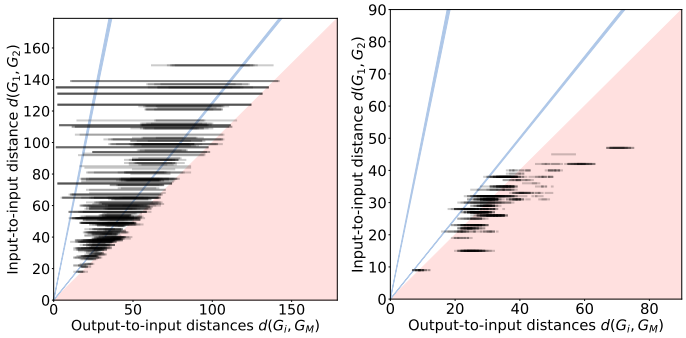

(d) FGW-Mixup (mIE = 3.82) (e) S-Mixup* (mIE = 3.28) (f) GeoMix* (mIE = 3.85)

Figure 5: Visualization of mixup graphs produced on the MUTAG dataset ($\lambda = 0.9 \pm \varepsilon$).



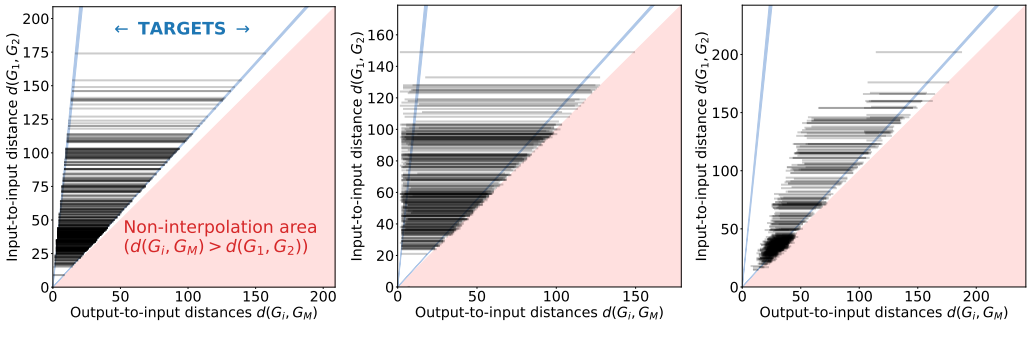

(a) GED-Mixup ($\mathrm{mIE} = 0.01$) (b) SubMix ($\mathrm{mIE} = 0.21$) (c) If-Mixup$^*$ ($\mathrm{mIE} = 0.72$)

N/A



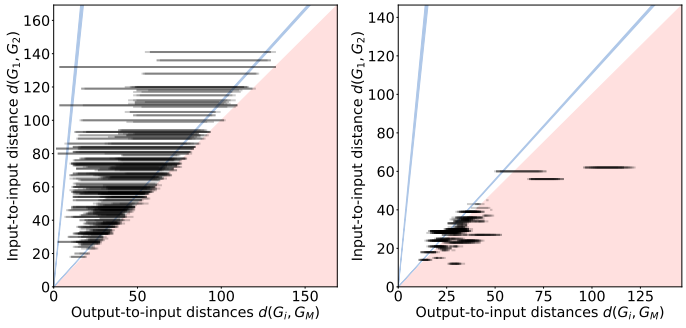

(d) FGW-Mixup (e) S-Mixup$^*$ ($\mathrm{mIE} = 0.97$) (f) GeoMix$^*$ ($\mathrm{mIE} = 1.24$)

Figure 6: Visualization of mixup graphs produced on the ENZYMES dataset ($\lambda = 0.8 \pm \varepsilon$). Results for FGW-Mixup are missing since we were not able to generate graphs for ENZYMES.



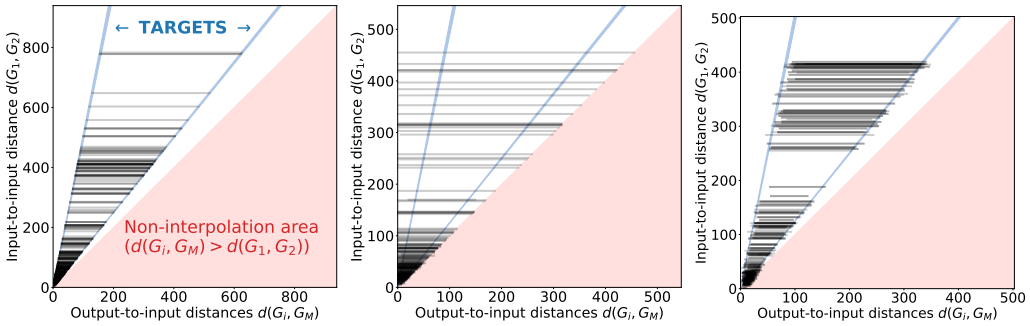

(a) GED-Mixup ($\mathrm{mIE} = 0.01$) (b) SubMix ($\mathrm{mIE} = 0.11$) (c) If-Mixup$^*$ ($\mathrm{mIE} = 0.61$)

N/A

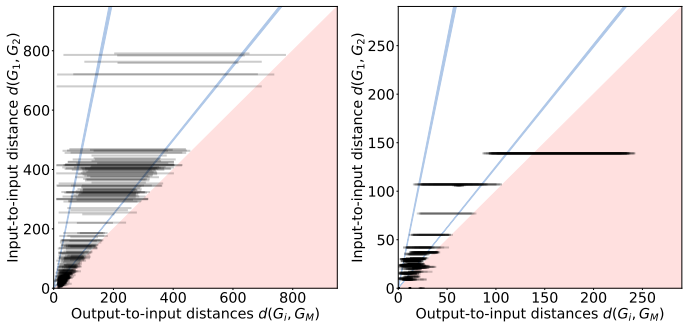



(d) FGW-Mixup (e) S-Mixup$^*$ ($\mathrm{mIE} = 0.69$) (f) GeoMix$^*$ ($\mathrm{mIE} = 1.19$)

Figure 7: Visualization of mixup graphs produced on the ENZYMES dataset ($\lambda = 0.9 \pm \varepsilon$). Results for FGW-Mixup are missing since we were not able to generate graphs for ENZYMES.

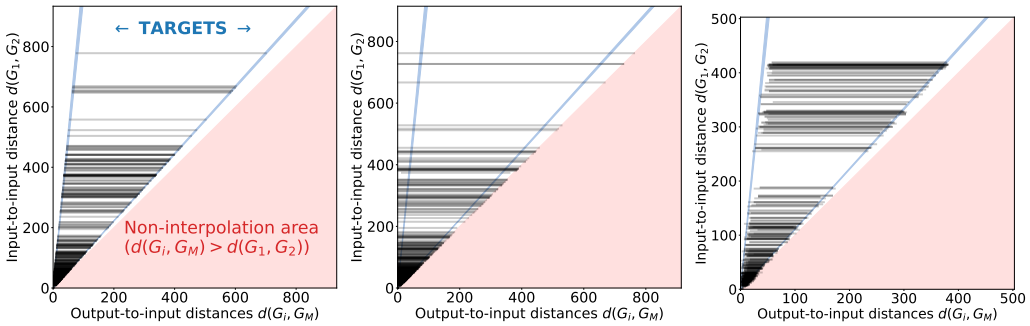



(a) GED-Mixup ($\mathrm{mIE} = 0.01$) (b) SubMix ($\mathrm{mIE} = 0.35$) (c) If-Mixup* ($\mathrm{mIE} = 0.61$)

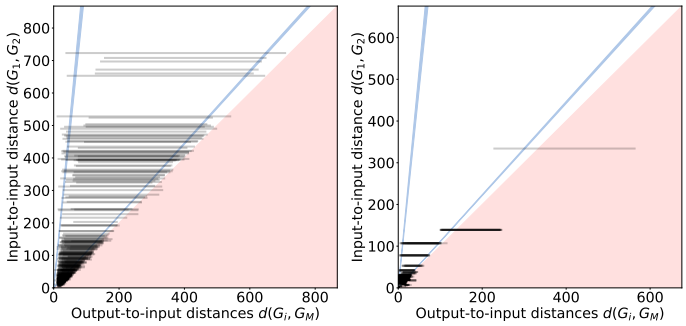



N/A (d) FGW-Mixup (e) S-Mixup* ($\mathrm{mIE} = 1.10$) (f) GeoMix* ($\mathrm{mIE} = 0.55$)

Figure 8: Visualization of mixup graphs produced on the IMDB-BINARY dataset ($\lambda = 0.8 \pm \varepsilon$). Results for FGW-Mixup are missing since we were not able to generate graphs for IMDB-BINARY.

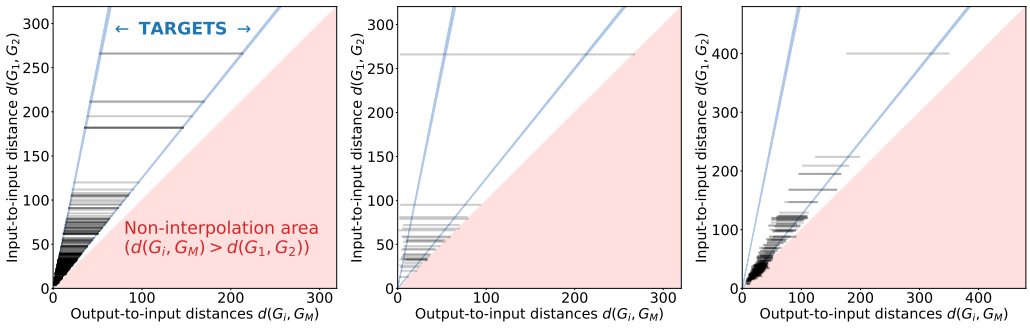



(a) GED-Mixup ($\mathrm{mIE} = 0.01$) (b) SubMix ($\mathrm{mIE} = 0.18$) (c) If-Mixup* ($\mathrm{mIE} = 0.33$)



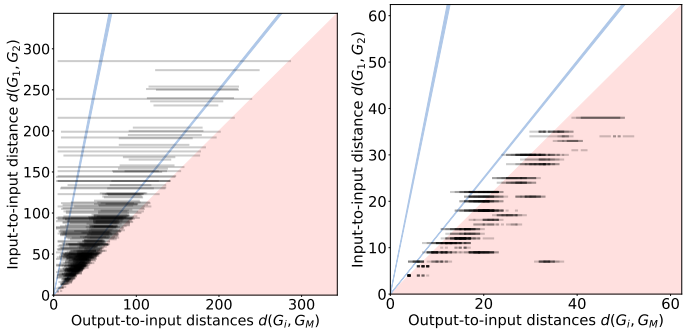

N/A (d) FGW-Mixup (e) S-Mixup* ($\mathrm{mIE} = 1.34$) (f) GeoMix* ($\mathrm{mIE} = 0.61$)

Figure 9: Visualization of mixup graphs produced on the IMDB-BINARY dataset ($\lambda = 0.9 \pm \varepsilon$). Results for FGW-Mixup are missing since we were not able to generate graphs for IMDB-BINARY.



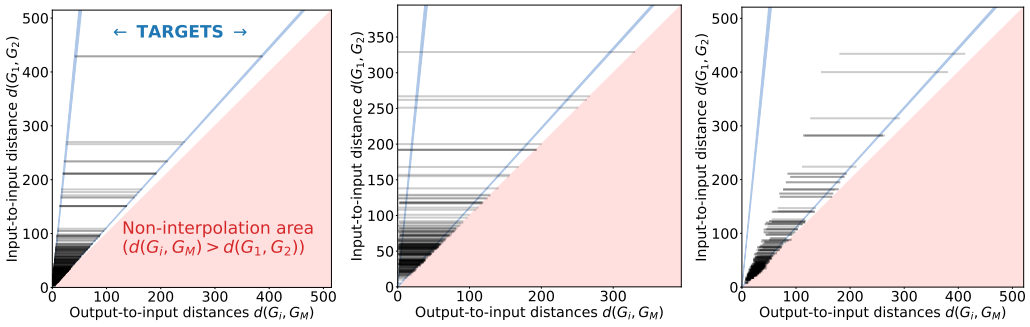

(a) GED-Mixup ($\mathrm{mIE} = 0.01$) (b) SubMix ($\mathrm{mIE} = 0.22$) (c) If-Mixup* ($\mathrm{mIE} = 0.91$)

N/A



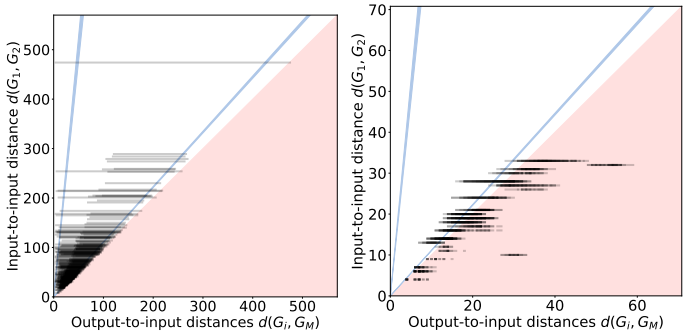

(d) FGW-Mixup (e) S-Mixup* ($\mathrm{mIE} = 0.56$) (f) GeoMix* ($\mathrm{mIE} = 1.80$)

Figure 10: Visualization of mixup graphs produced on the PROTEINS dataset ($\lambda = 0.8 \pm \varepsilon$). Results for FGW-Mixup are missing since we were not able to generate graphs for PROTEINS.




(a) GED-Mixup ($\mathrm{mIE} = 0.01$) (b) SubMix ($\mathrm{mIE} = 0.10$) (c) If-Mixup* ($\mathrm{mIE} = 0.82$)

N/A




(d) FGW-Mixup (e) S-Mixup* ($\mathrm{mIE} = 0.44$) (f) GeoMix* ($\mathrm{mIE} = 1.34$)

Figure 11: Visualization of mixup graphs produced on the PROTEINS dataset ($\lambda = 0.9 \pm \varepsilon$). Results for FGW-Mixup are missing since we were not able to generate graphs for PROTEINS.