From Ricci Curvature to Metric Matching: A Simplified Approach to Geometric Transfer Learning

Sung Moon Ko [*1] **Jaewan Lee** [*1] **Sumin Lee** [*1] **Soorin Yim** [*1] **Sehui Han** [1]

Abstract

Riemannian geometry offers a principled framework for modeling the smooth, curved latent spaces induced by deep learning models, and has shown particular promise in transfer learning. Recent approaches improve prediction performance by aligning the Ricci scalar curvature between target and source domains, thereby matching their intrinsic geometric structures. However, Ricci curvature computation is mathematically complex and computationally expensive, limiting its reproducibility. In this work, we propose a simplified metric matching approach that reduces both the algorithmic complexity and computational cost. Our method leverages an exact diffeomorphism between target and source spaces, enabling coordinate frame alignment such that, by definition of the curvature tensor, the curvature remains unchanged. Extensive experiments demonstrate that our approach improves training speed by 91.1% and reduces memory usage by 62.6% compared to GEAR, a Ricci curvature–based method, while maintaining comparable predictive performance.

1. Introduction

Geometrical interpretations of deep learning models have expanded significantly in recent years. In particular, for non-Euclidean settings, differential geometry has become a key mathematical tool for uncovering and analyzing the underlying structure of learned representations Bronstein et al. (2017). Since deep learning models aim to approximate complex non-linear mappings, the latent spaces they induce are often curved rather than flat. Riemannian differential geometry, which studies smooth, differentiable manifolds equipped with a well-defined metric tensor, provides a principled framework to interpret and extract meaningful information from such latent manifolds. Recent work in geometric deep learning has extended these ideas across multiple domains, including classification Pegios et al. (2024); Lee et al. (2022), clustering Hu et al. (2024); Yang et al. (2018), and generative modeling Park et al. (2023); Grattarola et al. (2019). Furthermore, Riemannian metric learning has been shown to enhance both model performance and interpretability by encouraging the network to operate in geometrically plausible directions Li et al. (2023); Sun et al. (2024).

Riemannian geometry has also proven effective in transfer learning. Transfer learning aims to overcome the limitations of scarce training data by leveraging knowledge from other tasks Zhuang et al. (2011); Long et al.; Zhuang et al. (2013; 2014); Pan et al. (2020); Quattoni et al. (2008); Kulis et al. (2011); Raghu et al. (2019); Yu et al. (2022). However, much of the existing research in this area has focused on classification models Radhakrishnan et al. (2023); Basu et al. (2023); Wenzel et al. (2022). In contrast, many scientific applications—such as molecular property prediction—require regression models. Despite this need, relatively few studies have explored regression-based transfer learning Scarselli et al. (2009); Bruna et al. (2013); Duvenaud et al. (2015); Defferrard et al. (2016); Jin et al. (2018); Coley et al. (2019); Ko et al. (2023a;b; 2024); Yim et al. (2024); Lee et al. (2024); Ko et al. (2025). Since many real-world problems are inherently regression-based, we focus on advancing regression transfer learning through a geometric interpretation of latent space alignment in this work.

The most notable recent work on geometric alignment for regression-based transfer learning is presented in Ko et al. (2025). This study extends the latent space alignment framework of Ko et al. (2023b) to full Ricci scalar curvature matching. By incorporating curvature alignment, the authors achieved global latent space matching, greater flexibility in choosing embedding structures, and demonstrated its effectiveness through extensive experiments on molecular property prediction tasks, yielding performance gains of 14.4% on random splits and 8.3% on scaffold splits. However, despite these advantages, the method incurs substantial computational overhead and structural complexity, which may hinder reproducibility and scalability. In this work, we build directly upon this approach and further extend the algorithm by introducing a metric matching architec-

[1]LG AI Research, Seoul, Korea. Correspondence to: Sung Moon Ko <sungmoon.ko@lgresearch.ai>, Sehui Han <hansse.han@lgresearch.ai>.

Accepted by ICML 2026 AI for Science Workshop

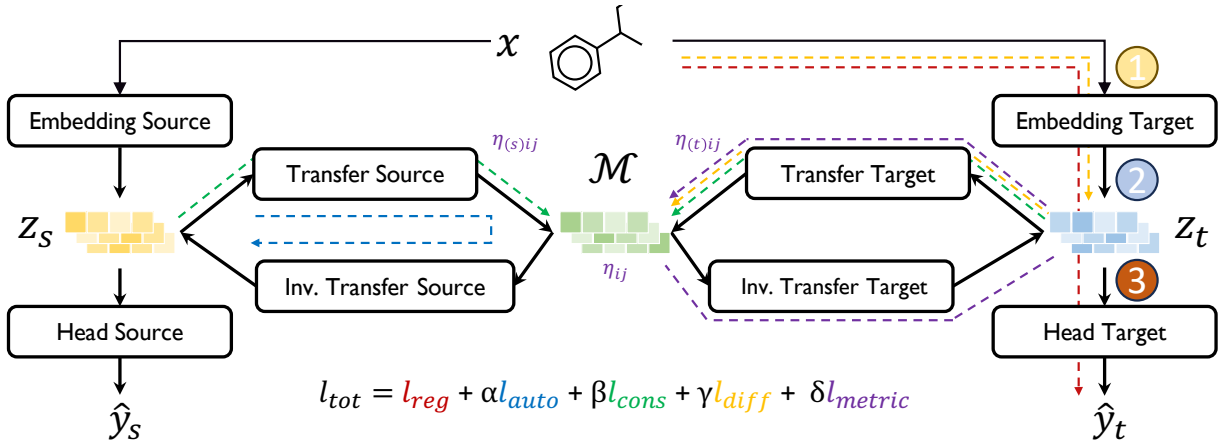

Figure 1. Overview of the simplified GEAR architecture. The framework consists of three main modules: (1) the embedding module, which encodes input data into latent vectors; (2) the transfer module, which aligns latent spaces across domains via diffeomorphic mappings; and (3) the task-specific heads for prediction.

ture that preserves the geometric alignment benefits while significantly reducing both complexity and computational cost.

A common approach to characterizing the shape of a manifold in differential geometry is through curvature computation. Prior work has leveraged this principle by aligning curvatures to improve latent space matching and, consequently, model performance. However, the Ricci scalar curvature is derived from the Riemann curvature tensor, which itself is constructed from the metric tensor and its derivatives up to second order. Therefore, if the metric tensors of two manifolds are matched and expressed in the same coordinate basis, their curvatures will, by definition, be identical. This observation implies that computing the full curvature tensor is not strictly necessary; by introducing metric matching combined with diffeomorphism alignment, one can ensure curvature equivalence directly. This approach substantially reduces structural complexity, and by avoiding the second-order derivatives of the metric tensor, it also eliminates much of the associated computational overhead.

Hence, we propose a novel metric matching algorithm that inherently extends the prior work GEAR Ko et al. (2025), addressing both its computational overhead and structural complexity. This improvement enhances the scalability and reproducibility of the model. We validate these advantages through extensive experiments on molecular property regression tasks, demonstrating that the proposed method is significantly faster than GEAR while achieving comparable performance.

Our main contributions are summarized as follows:

- We introduce a novel **metric matching** strategy combined with a **diffeomorphism loss** to achieve precise geometric alignment between latent spaces.
- We significantly reduce the structural complexity of the model, improving scalability and reproducibility.
- We achieve a training speed improvement of **91.1%** and a reduction in memory usage of **62.6%** compared to GEAR, while maintaining comparable predictive performance.

2. Methods

The latent space of a deep learning model is a nonlinearly transformed embedding of the input data points, where the network organizes them into a configuration that simplifies the task for downstream prediction heads. When multiple datasets are correlated and share overlapping regions in their distributions, this latent arrangement allows for geometric alignment between domains, enabling effective knowledge transfer.

Our first key assumption is that the latent space induced by the network is Riemannian. A Riemannian manifold is smooth and differentiable, equipped with a well-defined metric tensor. It is a classical result in differential geometry that any smooth manifold admits a Riemannian metric. Therefore, the main question is whether the latent space is smooth. Modern deep learning models are typically constructed from two components: (1) linear layers, which are trivially smooth, and (2) nonlinear activation functions. If these activations are smooth and differentiable—such as logistic, sinusoidal, or SiLU functions—then the resulting latent space is also smooth. Consequently, any network equipped with smooth activation functions induces a Riemannian latent space.

Given this structure, Riemannian geometry provides a principled framework for comparing and aligning the exact shapes of these latent manifolds—not merely their topological fea-

tures—thereby enabling more effective transfer learning across correlated tasks.

2.1. Riemann Curvature

To maintain an abstract and general formulation, we adopt the Einstein summation convention with index notation. The Riemann curvature tensor is the most general mathematical object describing the exact shape of a manifold across its regions. Its value depends on both the location of measurement and the choice of basis. This rank-4 tensor can always be expressed in terms of the Riemannian metric tensor. Constructing the Riemann curvature tensor requires first defining vector fields on curved manifolds and introducing the machinery to compare them at different points—namely, parallel transport and covariant differentiation.

The definition of a vector is tied to its behavior under coordinate transformations. A quantity is defined as a vector if it satisfies the vector transformation rule:

$$X = x^i dx_i = x'^i dx'_i = x^i \frac{dx'^j}{dx^i} \frac{dx^k}{dx'^j} dx_k = X' \quad (1)$$

Here, the transformation map $\frac{dx'^j}{dx^i}$ represents the Jacobian of the vector field. The real complication arises when differentiating a vector field. In flat (Euclidean) space, the derivative of a vector field is trivially another tensor. However, in a curved space, the ordinary derivative is not a well-defined vector transformation operator.

Geometrically, differentiation corresponds to an infinitesimal change along a path. In flat space, the basis vectors remain constant under coordinate transformations, but in a curved space, the basis itself changes from point to point. This variation in the basis requires the introduction of covariant derivatives, which incorporate correction terms (Christoffel symbols) to ensure that the derivative transforms as a tensor under coordinate changes.

$$\nabla_j T^i = \partial_j T^i + \Gamma^i{}_{jl} T^l \quad (2)$$

The Riemann curvature tensor measures how much a vector changes after being parallel transported along different paths on a manifold. Intuitively, it captures the failure of second covariant derivatives to commute.

For example, consider a vector lying on the equator of a 2D sphere. First, transport the vector along a path from the equator to the north pole, then down to the equator again, arriving at a point 90° east of the starting position. Finally, return along the equator to the original point. Upon returning, the vector will be rotated by 90° relative to its initial orientation.

In flat space, parallel transport is path-independent, so the vector would return unchanged regardless of the chosen path. In curved space, however, the difference between the transported vectors along two different paths is generally non-trivial. The Riemann curvature tensor formalizes this difference, encoding the intrinsic curvature of the manifold at each point and in each direction.

$$R^i{}_{ljk} T^l = [\nabla_j, \nabla_k] T^i \quad (3)$$

Furthermore, the indices of the Riemann curvature tensor can be contracted using the curved-space metric tensor, which itself can be obtained from a locally flat (Euclidean) metric through a transformation by the appropriate Jacobian matrices. This transformation encodes how local coordinates are embedded in the curved manifold, ensuring that all contractions and index manipulations remain consistent with the manifold's intrinsic geometry.

$$g_{ij} = \frac{dx'^m}{dx^i} \frac{dx'^n}{dx^j} \eta_{mn} = \frac{dx'^m}{dx^i} \frac{dx'_m}{dx^j} \quad (4)$$

Because the Riemann curvature tensor is antisymmetric in both its first two indices and its last two indices, there are two distinct non-trivial ways to contract it. Contracting over one such index pair yields the Ricci curvature tensor, which summarizes curvature information by reducing the tensor's rank from four to two. A further contraction of the Ricci tensor over its remaining indices produces the Ricci scalar curvature, a single scalar quantity that encapsulates the manifold's intrinsic curvature at each point.

$$R = g^{ij} g^{lk} R_{iljk} \quad (5)$$

As its name suggests, the Ricci scalar curvature is a scalar quantity and therefore remains invariant under any diffeomorphic coordinate transformation. This invariance makes it an especially convenient and powerful tool for characterizing the shape of the underlying manifold at a given point, providing a coordinate-independent measure of its intrinsic geometry. Building on this property, the previous work Ko et al. (2025) adopts the scalar curvature as the core element of its architecture.

However, the Ricci scalar curvature is derived from the metric tensor and its derivatives up to second order, involving a sequence of intricate contractions and nontrivial operations. This inherent complexity can significantly hinder both the reproducibility and computational efficiency of the algorithm. A closer examination of its formulation reveals that the scalar curvature will be identical if and only if the metric tensor and its coordinate basis are the same. Therefore, by defining a loss function that constrains both the diffeomorphism and the metric tensor, one can preserve curvature equivalence while drastically simplifying the architecture.

2.2. Main Architecture

The main architecture of our model builds upon the prior work of Ko et al. (2025). Based on the reasoning outlined

above, we replace the curvature loss with a diffeomorphism loss to ensure that mapped vectors lie in the same coordinate frame. By definition, this guarantees curvature equivalence, resulting in a substantially simplified GEAR architecture. Moreover, by eliminating the need to compute second-order derivatives of the metric tensor, our modification significantly increases training speed.

Our model consists of three main components: an embedding module, a transfer module, and task-specific head networks. Given an input data point x, it is first passed through the embedding layer to produce a latent vector:

$$z = \mathrm{Embed}(x) \tag{6}$$

We denote the latent vector from the target domain as z_t and from the source domain as z_s throughout the rest of this work. These latent vectors will be transformed to one another by the transfer module.

$$z' = \mathrm{Transfer}(z), \qquad \hat{z} = \mathrm{Transfer}^{-1}(z') \tag{7}$$

Here, z' denotes a latent vector expressed in a locally flat frame, and $\hat{z}$ represents its pullback to the original coordinate patch. These transfer modules are implemented in pairs: one network pushes forward a latent vector to a locally flat frame, while the other pulls it back to the original coordinate frame via an inverse transformation. Directly computing matrix inverses, however, can be numerically unstable. To address this, we adopt an autoencoder-style architecture, which provides a stable mechanism for learning the forward and inverse mappings while maintaining reliable gradient flow during backpropagation. Accordingly, we define an autoencoder loss for this process.

$$l_{\mathrm{auto}} = \mathrm{MSE}(z, \hat{z}) \tag{8}$$

To be more specific, we define four distinct transformations that can be computed from the source and target latent representations:

$$z'_s = \mathrm{Transfer}_{s\to LF}(z_s) \qquad \hat{z}_s = \mathrm{Transfer}^{-1}{}_{LF\to s}(z'_s) \tag{9}$$

$$z'_t = \mathrm{Transfer}_{t\to LF}(z_t) \qquad \hat{z}_t = \mathrm{Transfer}^{-1}{}_{LF\to t}(z'_t) \tag{10}$$

Here, z'_s and z'_t denote the source and target latent vectors transformed into a locally flat (LF) coordinate frame. The inverse transformations $\hat{z}_s$ and $\hat{z}_t$ pull these vectors back into their respective original coordinate frames. This formulation allows us to explicitly model and supervise both forward and backward mappings, ensuring consistency in the learned diffeomorphic transformations between latent spaces.

Since the source and target latent vectors originate from the same input, their representations in the locally flat (LF) coordinate frame should coincide. We enforce this constraint through a consistency loss:

$$l_{\mathrm{cons}} = \mathrm{MSE}(z'_s, z'_t) \tag{11}$$

This loss ensures that, after transformation into the LF frame, both domains share an identical latent representation, thereby preserving geometric alignment.

Furthermore, we introduce an additional constraint to ensure that the coordinate transformation from the task-specific latent space to the locally flat (LF) coordinate system is geometrically consistent. Although different molecular property prediction tasks may employ different encoders, they are constructed from the same input SMILES representation. We therefore assume that the underlying molecular geometry is shared across task-specific latent spaces, while each encoder may provide a different coordinate parameterization of this common structure.

Let z denote a latent representation in the original task-specific latent space, and let z' denote its corresponding representation in the LF coordinate system. The local coordinate transformation is characterized by the Jacobian matrix, defined as

$$J^i{}_j = \frac{\partial z'^i}{\partial z^j}. \tag{12}$$

The corresponding coordinate basis in derivative form transforms according to the inverse Jacobian:

$$\frac{\partial}{\partial z'^i} = \frac{\partial z^j}{\partial z'^i}\frac{\partial}{\partial z^j} = \left(J^{-1}\right)^j{}_i \frac{\partial}{\partial z^j}. \tag{13}$$

In parallel, the transfer module is designed to map the coordinate basis of the original latent space to the LF coordinate system. However, directly aligning latent representations is not sufficient to guarantee geometric consistency, because the Ricci scalar depends on the induced local geometry, including the transformed coordinate basis and metric structure, rather than only on the pointwise latent coordinates. Therefore, we impose a differential consistency constraint requiring that the basis transformation induced by the transfer module coincides with the basis transformation induced by the Jacobian:

$$\left(J^{-1}\right)^j{}_i \frac{\partial}{\partial z^j} = \mathrm{Transfer}\left(\frac{\partial}{\partial z^i}\right). \tag{14}$$

Accordingly, we define the differential consistency loss as

$$\mathcal{L}_{\mathrm{diff}} = \mathrm{MSE}\left(\left(J^{-1}\right)^j{}_i \frac{\partial}{\partial z^j}, \mathrm{Transfer}\left(\frac{\partial}{\partial z^i}\right)\right). \tag{15}$$

This constraint ensures that the transfer module and the Jacobian define the same local basis transformation between the task-specific latent space and the LF coordinate system.

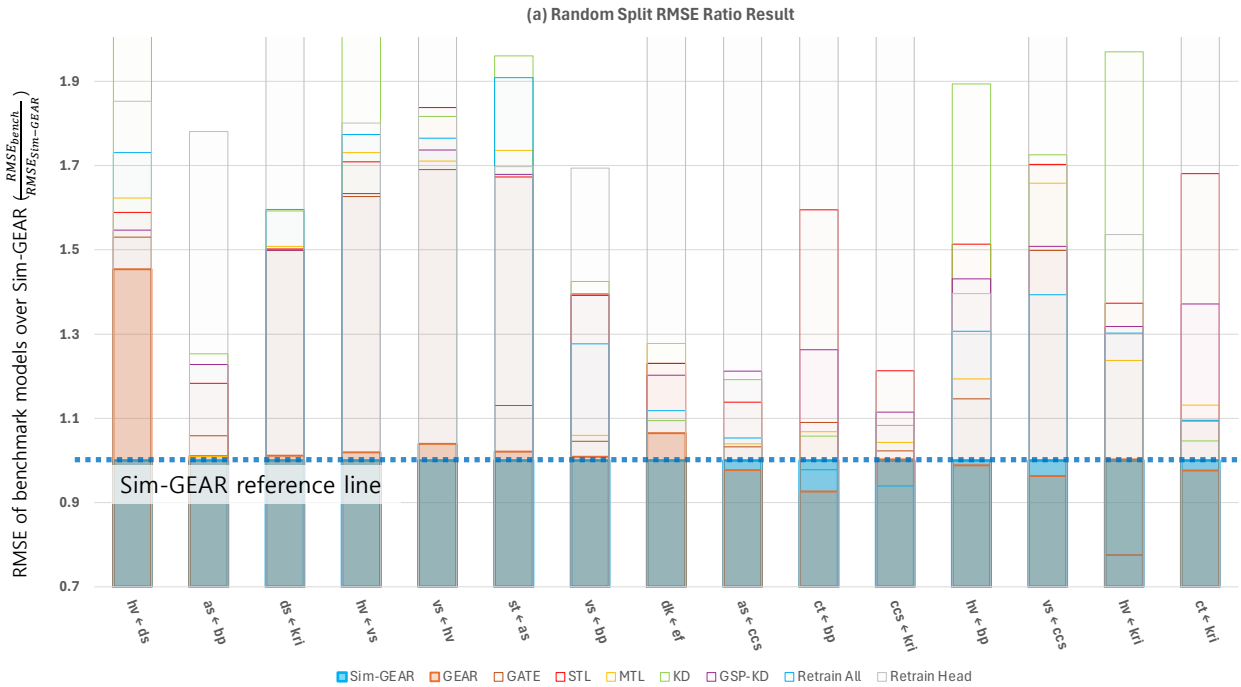

Figure 2. The figure presents model performance under a random data split scheme. Results are normalized by the RMSE of Sim-GEAR, where values below 1 indicate better performance and values above 1 indicate worse performance relative to Sim-GEAR. The primary competitor is GEAR, which leverages full curvature information. As shown, Sim-GEAR achieves performance comparable to GEAR and, in many cases, outperforms it.

Together with the LF metric constraint, which encourages the induced metric in the LF frame to approach the identity matrix for each task, this alignment allows latent spaces induced from different molecular property tasks to share a consistent local geometric structure. Consequently, the Ricci scalar computed in the LF coordinate system is encouraged to remain invariant across task-specific latent spaces derived from the same molecular input.

Next, we define the metric loss. To compute the metric tensor on a curved manifold, we recall the definition of the induced metric tensor 4, which in local coordinates can be expressed as

$$g_{ij} = \frac{\partial z'^k}{\partial z^i}\frac{\partial z'^l}{\partial z^j}\eta_{kl} \tag{16}$$

where z^i denotes the coordinates on the manifold, z'^k the embedding in the locally flat (LF) frame, and η_{ab} the Euclidean metric in that frame.

Given this, we can compute the induced metric tensors for both the source and target embeddings in the LF frame as:

$$g^s_{ij} = \frac{\partial z'^k_s}{\partial z^i}\frac{\partial z'^l_s}{\partial z^j}\eta_{kl} \tag{17}$$

$$g^t_{ij} = \frac{\partial z'^k_t}{\partial z^i}\frac{\partial z'^l_t}{\partial z^j}\eta_{kl} \tag{18}$$

Here the Jacobians can be computed from taking a derivative of the given transfer module. We consider MLP type network that using SiLU as an activation, the explicit expression for the derivative of a single layer is as follows.

$$\frac{dz^{(n+1)i}}{dz^{(n)j}} = W^{(n+1)i}{}_j\sigma^i + (z^{(n+1)})^i W^{(n+1)a_3}{}_j E^i{}_{a_3}(\sigma^2)^i \tag{19}$$

Where, $W^{(n)i}{}_j$ and $b^{(n)}{}_j$ are weight and biases of n-th layer in the transform network. The new notations introduced in the equation above are defined as follows. LS in the equation denotes the logistic function and σ^i and $E^i{}_j$ are as follows:

$$\sigma^i = \frac{1}{1+e^{-(W^{(1)i}{}_j z^j + b^{(1)i})}},$$
$$\mathrm{E}^i{}_l \equiv (e^{-(W^l{}_j z^j + b^j)})^i{}_l = \begin{cases} e^{-(W^l{}_j z^j + b^l)} & \text{if } l = i \\ 0 & \text{if } l \neq i \end{cases} \tag{20}$$

And $(\sigma^2)^i$ is the element-wise square of the original. From

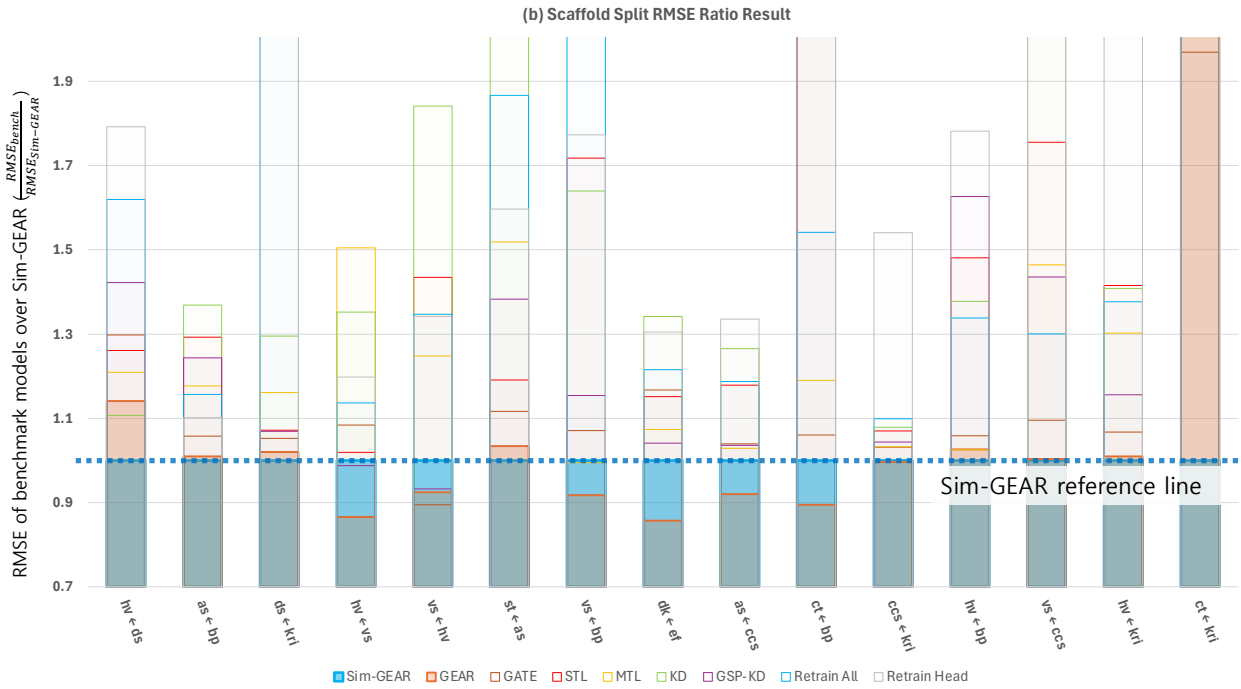

Figure 3. The figure presents model performance under a scaffold data split scheme, reflecting the models' extrapolation behavior. As shown, Sim-GEAR outperforms most baselines, including the primary competitor, GEAR.

the chain rule, one can easily combine multiple set of Jacobians to compute full Jacobian of an MLP network.

$$J^i_{\ j} = \frac{dz'^i}{dz^j} = \frac{dz'^i}{dz^{(n-1)k_{n-1}}} \frac{dz^{(n-1)k_{n-1}}}{dz^{(n-2)k_{n-2}}} \cdots \frac{dz^{(1)k_1}}{dz^j} \tag{21}$$

Finally, we define the metric losses. The metric loss plays a crucial role, as the space formed by the transfer module lacks any form of direct supervision. Without proper regularization, the space is not guaranteed to be locally flat, since there are infinitely many ways to define a basis that still satisfy the previously introduced constraints. The metric loss guides the transfer module toward preserving local flatness. It is defined as the discrepancy between the induced flat metric and the Euclidean metric, which in this case is represented by the identity matrix η_{ij}.

$$\begin{aligned} l_{\text{metric}} &= \text{MSE}(\eta_{ij}, \eta_{(s)_{ij}}) + \text{MSE}(\eta_{ij}, \eta_{(t)_{ij}}) \\ \eta_{(s)ij} &= \left(\frac{\partial \hat{z}_s}{\partial \hat{z}'_s}\right)^m_i \left(\frac{\partial z'_s}{\partial \hat{z}_s}\right)^k_m \eta_{kl} \left(\frac{\partial z'_s}{\partial \hat{z}_s}\right)^l_n \left(\frac{\partial \hat{z}_s}{\partial \hat{z}'_s}\right)^n_j \\ \eta_{(t)ij} &= \left(\frac{\partial \hat{z}_t}{\partial \hat{z}'_t}\right)^m_i \left(\frac{\partial z'_t}{\partial \hat{z}_t}\right)^k_m \eta_{kl} \left(\frac{\partial z'_t}{\partial \hat{z}_t}\right)^l_n \left(\frac{\partial \hat{z}_t}{\partial \hat{z}'_t}\right)^n_j \end{aligned} \tag{22}$$

$\eta_{(s)_{ij}}$ and $\eta_{(t)_{ij}}$ are the induced flat metrics obtained by loop computations with the inverse transfer from the source and target, respectively, to the transfer module. The symbol $\hat{z}'$ denotes the transferred latent vector, which is mapped from the flat space to the curved space, and then back to the flat space.

This encourages the embeddings from the source and target domains to exhibit identical local geometric structures in the LF frame. When combined with the consistency and differential losses, these conditions ensure equality of their curvature tensors without requiring the explicit computation of second-order derivatives. In **GEAR**, computing the Ricci scalar curvature requires evaluating all Christoffel symbols and their derivatives, resulting in a computational complexity of $\mathcal{O}(d^4)$ for a d-dimensional latent space. In contrast, our metric-based formulation only relies on first-order Jacobians, reducing the complexity to $\mathcal{O}(d^3)$ and entirely eliminating the need for second-order derivatives. Moreover, the curvature and mapping losses can be unified into a single diffeomorphism loss, which noticeably simplifies the model architecture.

Finally, by incorporating the standard regression loss:

$$l_{\text{reg}} = \text{MSE}(y_t, \hat{y}_t) \tag{23}$$

where y_t and $\hat{y}_t$ denote the ground-truth label of the input data and the predicted value, respectively, the total loss can be expressed as a weighted sum of all components:

$$l_{\text{tot}} = l_{\text{reg}} + \alpha\, l_{\text{auto}} + \beta\, l_{\text{cons}} + \gamma\, l_{\text{diff}} + \delta\, l_{\text{metric}} \tag{24}$$

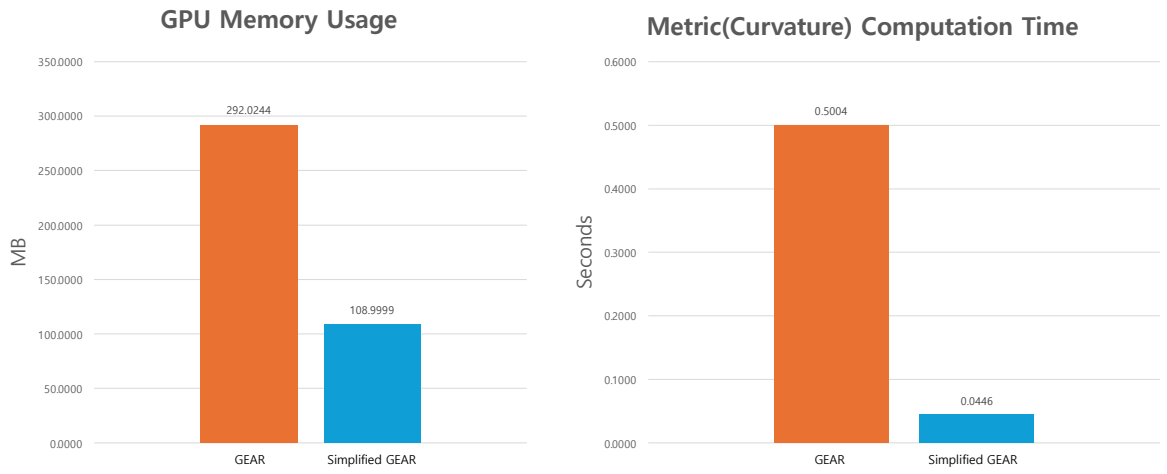

Figure 4. The figure shows that the Simplified GEAR achieves comparable accuracy to GEAR while offering significant speed improvements and reduced memory consumption.

Here, α, β, γ, and δ are hyperparameters that balance the contribution of each loss term during training.

3. Experiments

3.1. Experimental settings

The experiments are conducted on several publicly available databases: OCHEM (Sushko et al., 2011), PubChem (Kim et al., 2022). We utilized 11 different molecular property datasets sourced from the two open-access databases, as detailed in Table 4 and the descriptions below. We selected 15 distinct task combinations to demonstrate the model's performance and to provide a direct comparison with GEAR (Ko et al., 2025). We provide an explicit description of the physical meaning associated with each dataset to facilitate better understanding and context in the appendix section 4.

To simulate extrapolation scenarios, which frequently arise in practice, we employ two different data-splitting strategies to evaluate model performance under both standard (random split) and out-of-distribution (scaffold split) settings. In the scaffold split, all samples containing a specific scaffold are assigned to the test set, ensuring a strict OOD evaluation. A total of 15 dataset pairs are constructed, each evaluated under both splitting strategies. This results in 30 experimental settings in total: 15 with random splits and 15 with scaffold-based splits.

All experiments were conducted using a single NVIDIA A40 GPU with a 4-fold cross-validation setup. For reproducibility, we report the complete set of hyperparameters used during training in the appendix tables 1, 2 and 3.

3.2. Main results

The primary objective of our study is to demonstrate that the **Sim-GEAR** achieves performance comparable to that of the original **GEAR** (Ko et al., 2025). Despite the algorithmic simplification, we emphasize that this improvement is achieved *without loss of generality*. As illustrated in Figures 2 and 3, the streamlined architecture achieves predictive performance that is highly comparable to that of GEAR, with only marginal differences in average RMSE. Specifically, Sim-GEAR shows an improvement of 0.020 under the random split setting, while exhibiting a slight decrease of 0.003 under the scaffold split. Given the variability typically observed across datasets, these differences are negligible, indicating that the predictive capability of Sim-GEAR remains effectively on par with that of GEAR. Notably, this level of performance is achieved despite the significantly simplified architecture of Sim-GEAR, which removes components designed to explicitly model full curvature information.

Beyond predictive performance, the simplified design of Sim-GEAR leads to substantial improvements in computational efficiency. By reducing architectural complexity, the model minimizes both parameter overhead and intermediate memory usage during training. To systematically evaluate these advantages, we conducted a series of experiments measuring GPU memory consumption and training time per batch, using a batch size of 512 samples. These metrics provide a practical assessment of scalability, particularly in large-scale or resource-constrained settings.

The results, summarized in Figure 4, demonstrate that Sim-GEAR achieves a **62.6% reduction in GPU memory usage** compared to GEAR. In addition, the training time per batch is reduced by **91.1%**, indicating a substantial speedup

in model optimization. Such improvements are especially meaningful in real-world applications, where training efficiency directly impacts iteration speed, cost, and feasibility of large-scale experimentation.

Overall, these findings highlight a key advantage of Sim-GEAR: it successfully preserves the predictive strength of the original GEAR framework while significantly enhancing computational efficiency. This balance between accuracy and efficiency makes Sim-GEAR a more practical and scalable alternative, particularly in scenarios where computational resources are limited or rapid model development is required.

4. Discussion

We propose a novel transfer learning framework based on metric tensor matching. As discussed in the Methods section, when a deep learning model employs differentiable activation functions, the resulting latent space is smooth and differentiable. Since any smooth manifold admits a Riemannian metric, such latent representations can naturally be interpreted as Riemannian manifolds. In principle, one could compute curvature tensors directly from the diffeomorphic mapping between coordinate systems. However, we demonstrate that enforcing consistency of the metric tensor, together with the diffeomorphism, is sufficient to guarantee equivalence of the Ricci scalar curvature between latent spaces. This key observation eliminates the need to explicitly compute full curvature tensors, thereby significantly reducing model complexity while improving reproducibility and training efficiency—without compromising predictive performance.

To further validate the proposed approach, we are currently preparing more extensive experimental evaluations. These include comprehensive ablation studies across diverse combinations of molecular property prediction tasks, aimed at providing statistically robust comparisons with existing methods and standard benchmarks. In addition, the proposed framework offers considerable flexibility in the choice of non-linear embedding architectures, making it naturally extensible beyond single-modality settings. As a future direction, we plan to explore multi-modal applications, such as integrating language-based representations with graph-structured molecular data, as well as extending the method to cross-domain transfer scenarios.

References

Basu, S., Katdare, P., Sattigeri, P., Chenthamarakshan, V., Driggs-Campbell, K., Das, P., and Varshney, L. R. Efficient equivariant transfer learning from pretrained models, 2023. URL https://arxiv.org/abs/2305.09900.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4): 18–42, 2017.

Bruna, J., Zaremba, W., Szlam, A., and Lecun, Y. Spectral networks and locally connected networks on graphs. 12 2013.

Carroll, S. M. *Spacetime and Geometry: An Introduction to General Relativity*. Addison-Wesley, San Francisco, 2004.

Coley, C., Jin, W., Rogers, L., Jamison, T. F., Jaakkola, T. S., Green, W. H., Barzilay, R., and Jensen, K. F. A graph-convolutional neural network model for the prediction of chemical reactivity. *Chem. Sci.*, 10:370–377, 2019. doi: 10.1039/C8SC04228D. URL http://dx.doi.org/10.1039/C8SC04228D.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. 06 2016.

Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. Convolutional networks on graphs for learning molecular fingerprints. *Advances in Neural Information Processing Systems (NIPS)*, 13, 09 2015.

Grattarola, D., Livi, L., and Alippi, C. Adversarial autoencoders with constant-curvature latent manifolds. *Applied Soft Computing*, 81:105511, 2019.

Hu, W., Zhan, H., Tian, Y., Xiong, Y., and Lu, Y. Enhanced video clustering using multiple riemannian manifold-valued descriptors and audio-visual information. *Expert Systems with Applications*, 246:123099, 2024.

Jin, W., Yang, K., Barzilay, R., and Jaakkola, T. Learning multimodal graph-to-graph translation for molecular optimization, 12 2018.

Joshi, C. K., Liu, F., Xun, X., Lin, J., and Foo, C. S. On representation knowledge distillation for graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Kim, S., Chen, J., Cheng, T., Gindulyte, A., He, J., He, S., Li, Q., Shoemaker, B. A., Thiessen, P. A., Yu, B., Zaslavsky, L., Zhang, J., and Bolton, E. E. PubChem 2023 update. *Nucleic Acids Research*, 51(D1):D1373–D1380, 10 2022. ISSN 0305-1048. doi: 10.1093/nar/gkac956. URL https://doi.org/10.1093/nar/gkac956.

Ko, S. M., Cho, S., Jeong, D.-W., Han, S., Lee, M., and Lee, H. Grouping matrix based graph pooling with adaptive number of clusters. *Proceedings of*

the AAAI Conference on Artificial Intelligence, 37(7): 8334–8342, June 2023a. ISSN 2159-5399. doi: 10.1609/aaai.v37i7.26005. URL http://dx.doi.org/10.1609/aaai.v37i7.26005.

Ko, S. M., Lee, S., Jeong, D.-W., Lim, W., and Han, S. Geometrically aligned transfer encoder for inductive transfer in regression tasks, 2023b. URL https://arxiv.org/abs/2310.06369.

Ko, S. M., Lee, S., Jeong, D.-W., Kim, H., Lee, C., Yim, S., and Han, S. Multitask extension of geometrically aligned transfer encoder, 2024. URL https://arxiv.org/abs/2405.01974.

Ko, S. M., Lee, J., Lee, S., Yim, S., Bae, K., and Han, S. Geometric embedding alignment via curvature matching in transfer learning, 2025. URL https://arxiv.org/abs/2506.13015.

Kulis, B., Saenko, K., and Darrell, T. What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. *CVPR 2011*, pp. 1785–1792, 2011. URL https://api.semanticscholar.org/CorpusID:7419723.

Lee, C., Jeong, D.-W., Ko, S. M., Lee, S., Kim, H., Yim, S., Han, S., Kim, S., and Lim, S. Scalable multi-task transfer learning for molecular property prediction, 2024. URL https://arxiv.org/abs/2410.00432.

Lee, J. M. *Introduction to Riemannian Manifolds*. Springer, 2nd edition, 2018.

Lee, Y., Kim, S., Choi, J., and Park, F. A statistical manifold framework for point cloud data. In *International Conference on Machine Learning*, pp. 12378–12402. PMLR, 2022.

Li, Y., Fei, C., Wang, C., Shan, H., and Lu, R. Geometry flow-based deep riemannian metric learning. *IEEE/CAA Journal of Automatica Sinica*, 10(9):1882–1892, 2023.

Long, M., Wang, J., Ding, G., Cheng, W., Zhang, X., and Wang, W. *Dual Transfer Learning*, pp. 540–551. doi: 10.1137/1.9781611972825.47. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611972825.47.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Pan, J., Cui, T., Duy Le, T., Li, X., and Zhang, J. Multi-group transfer learning on multiple latent spaces for text classification. *IEEE Access*, 8:64120–64130, 2020. doi: 10.1109/ACCESS.2020.2984571.

Park, Y.-H., Kwon, M., Choi, J., Jo, J., and Uh, Y. Understanding the latent space of diffusion models through the lens of riemannian geometry. *Advances in Neural Information Processing Systems*, 36:24129–24142, 2023.

Pegios, P., Feragen, A., Hansen, A. A., and Arvanitidis, G. Counterfactual explanations via riemannian latent space traversal. *CoRR*, 2024.

Quattoni, A., Collins, M., and Darrell, T. Transfer learning for image classification with sparse prototype representations. *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2, 03 2008. doi: 10.1109/CVPR.2008.4587637.

Radhakrishnan, A., Ruiz Luyten, M., Prasad, N., and Uhler, C. Transfer learning with kernel methods. *Nature Communications*, 14(1):5570, September 2023.

Raghu, M., Zhang, C., Kleinberg, J. M., and Bengio, S. Transfusion: Understanding transfer learning with applications to medical imaging. *CoRR*, abs/1902.07208, 2019. URL http://arxiv.org/abs/1902.07208.

Scarselli, F., Gori, M., Tsoi, A., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 20:61–80, 01 2009. doi: 10.1109/TNN.2008.2005605.

Sun, X., Liao, D., MacDonald, K., Zhang, Y., Liu, C., Huguet, G., Wolf, G., Adelstein, I., Rudner, T. G., and Krishnaswamy, S. Geometry-aware generative autoencoders for warped riemannian metric learning and generative modeling on data manifolds. *CoRR*, 2024.

Sushko, I., Novotarskyi, S., Körner, R., Pandey, A. K., Rupp, M., Teetz, W., Brandmaier, S., Abdelaziz, A., Prokopenko, V. V., Tanchuk, V. Y., et al. Online chemical modeling environment (ochem): web platform for data storage, model development and publishing of chemical information. *Journal of computer-aided molecular design*, 25:533–554, 2011.

Wald, R. M. *General Relativity*. University of Chicago Press, Chicago, 1984.

Weinberg, S. *Gravitation and Cosmology: Principles and Applications of the General Theory of Relativity*. John Wiley & Sons, New York, 1972.

Wenzel, F., Dittadi, A., Gehler, P. V., Simon-Gabriel, C.-J., Horn, M., Zietlow, D., Kernert, D., Russell, C., Brox, T., Schiele, B., Schölkopf, B., and Locatello, F. Assaying out-of-distribution generalization in transfer learning, 2022. URL https://arxiv.org/abs/2207.09239.

Yang, K., Swanson, K., Jin, W., Coley, C., Eiden, P., Gao, H., Guzman-Perez, A., Hopper, T., Kelley, B., Mathea, M., Palmer, A., Settels, V., Jaakkola, T., Jensen, K., and Barzilay, R. Analyzing learned molecular representations for property prediction. *Journal of Chemical Information and Modeling*, 59, 07 2019. doi: 10.1021/acs.jcim.9b00237.

Yang, T., Arvanitidis, G., Fu, D., Li, X., and Hauberg, S. Geodesic clustering in deep generative models. *arXiv preprint arXiv:1809.04747*, 2018.

Yim, S., Jeong, D.-W., Ko, S. M., Lee, S., Kim, H., Lee, C., and Han, S. Task addition in multi-task learning by geometrical alignment, 2024. URL https://arxiv.org/abs/2409.16645.

Yu, X., Wang, J., Hong, Q.-Q., Teku, R., Wang, S.-H., and Zhang, Y.-D. Transfer learning for medical images analyses: A survey. *Neurocomputing*, 489:230–254, 2022. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2021.08.159. URL https://www.sciencedirect.com/science/article/pii/S0925231222003174.

Zhuang, F., Luo, P., Xiong, H., He, Q., Xiong, Y., and Shi, Z. Exploiting associations between word clusters and document classes for cross-domain text categorization†. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 4(1):100–114, 2011. doi: https://doi.org/10.1002/sam.10099. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sam.10099.

Zhuang, F., Luo, P., Du, C., He, Q., and Shi, Z. Triplex transfer learning: Exploiting both shared and distinct concepts for text classification. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pp. 425–434, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450318693. doi: 10.1145/2433396.2433449. URL https://doi.org/10.1145/2433396.2433449.

Zhuang, F., Luo, P., Du, C., He, Q., Shi, Z., and Xiong, H. Triplex transfer learning: Exploiting both shared and distinct concepts for text classification. *IEEE Transactions on Cybernetics*, 44(7):1191–1203, 2014. doi: 10.1109/TCYB.2013.2281451.

A. Notations

Our notation follows index notation and the Einstein summation convention. The functions and matrices used in our algorithm are defined as follows.

$$X \text{ : Vector} \tag{25}$$
$$X^{\mu} \text{ : Vector Field} \tag{26}$$
$$dx_{\mu} \text{ : Basis} \tag{27}$$
$$X_{\mu} \text{ : Dual Vector Field} \tag{28}$$
$$dx^{\mu} \text{ : Dual Basis} \tag{29}$$
$$T \text{ : Tensor} \tag{30}$$
$$T^{\nu_1 \cdots \nu_p}{}_{\mu_1 \cdots \mu_q} \text{ : (p, q) Tensor Field} \tag{31}$$
$$g_{\mu\nu} \text{ : Metric Tensor} \tag{32}$$
$$\delta_{\mu\nu} \text{ : Kronecker Delta} \tag{33}$$
$$\nabla_{\mu} \text{ : Covariant Derivative} \tag{34}$$
$$\mathcal{L}_X \text{ : Lie Derivative} \tag{35}$$
$$\Gamma^{\rho}{}_{\mu\nu} \text{ : Christoffel Symbol} \tag{36}$$

All indices are raised and lowered by the metric $g_{\mu\nu}$. For instances,

$$g^{\mu}{}_{\nu} = g^{\mu\rho} g_{\rho\nu} \tag{37}$$

where

$$g^{\mu\nu} g_{\mu\nu} = \delta^{\mu}{}_{\nu} = D \tag{38}$$

Here D is the number of dimensions.

B. Motivations and Theoretical Backgrounds

We prepared this section to assist readers who may not be familiar with the mathematical foundations of differential geometry. The content is essentially a summarized compilation of well-established textbook materials, including (Weinberg, 1972; Carroll, 2004; Wald, 1984). In addition, the material in subsections B.3, B.5, and B.6 is also covered in the original GATE paper (Ko et al., 2023b).

B.1. Justification for the Riemannian Geometry Assumption

In this subsection, we clarify the motivations and the rationale behind this assumption and why it is both theoretically sound and practically justified in the context of deep learning. While this was briefly mentioned in Section 2 of our manuscript, we acknowledge that a more explicit theoretical justification is warranted. Below, we provide a detailed rationale to clarify why this assumption is both mathematically valid and practically appropriate in the context of our method.

The assumption that latent spaces in deep learning models can be treated as Riemannian manifolds rests on the following logical reasoning:

1. A **Riemannian manifold** is formally defined as a *smooth manifold* equipped with a *Riemannian metric*—a smoothly varying inner product on each tangent space.
2. A classical result in differential geometry establishes that **any smooth manifold admits a Riemannian metric**. This is a well-known theorem found in standard references such as (Lee, 2018)
3. The critical question, then, is whether the latent space of a deep learning model qualifies as a smooth manifold. This can be affirmed based on the construction of modern deep neural networks:

- *Linear transformations* are inherently smooth mappings.
- *Nonlinear activation functions* (e.g., Tanh, Sigmoid, SiLU) are continuously differentiable and thus smooth.

4. Therefore, when smooth activation functions are used, the entire model becomes a composition of smooth functions. The resulting latent space—formed by mappings from the input through the network—is itself smooth and hence forms a smooth manifold.

Based on the above, it follows directly that the latent space **can be equipped with a Riemannian metric**, rendering it a Riemannian manifold.

This assumption not only holds mathematically but is also aligned with practices in prior literature, including GATE and other geometric learning frameworks. Modeling latent spaces as Riemannian manifolds enables the use of powerful geometric tools—such as curvature—to capture structural properties that are otherwise inaccessible through Euclidean assumptions. In our case, this motivates the introduction of Ricci curvature alignment as a principled approach to improving transfer performance.

B.2. Motivation for Ricci Curvature Matching

Our work builds on the GATE architecture (Ko et al., 2023b) and thus inherits foundational assumptions from that framework. As described in the GATE paper, each data point within a task lies on a manifold, and the set of such points forms a coordinate patch, interpretable as a task-specific coordinate system. This is reasonable because many downstream tasks originate from a universal molecular representation (e.g., SMILES), with task-specific latent representations viewed as coordinate transformations of the same underlying structure.

Given the latent space's smoothness, it can be modeled as a Riemannian manifold (as argued in the previous section). Accordingly, task-specific latent spaces can be connected via diffeomorphisms—smooth, invertible mappings between manifolds.

We require the following assumptions to hold for the dataset:

- The source and target tasks are correlated.
- Their distributions share overlapping regions.

These assumptions are realistic, as our dataset includes many scientifically correlated task pairs, and most molecules have multiple annotated properties.

The fundamental strategy in both GATE and GEAR is to use source-task data to compensate for underrepresented regions in the target task. Given a Riemannian latent space, we can transfer knowledge across tasks by learning diffeomorphic mappings between latent representations.

To make this concrete, consider the following example: Suppose we have two molecules—water and oil. We know the melting point of water but not its boiling point; for oil, we have both values. If boiling point prediction is the target task and melting point is the source task, then we can train the model to learn a mapping from oil's melting-point representation to its boiling-point representation. Once trained, the model can infer water's boiling-point representation from its melting-point latent vector, transferring knowledge via the learned transformation. This enables improved performance on the target task.

Understanding the geometry of latent spaces is essential for meaningful transfer. Riemannian manifolds are inherently curved, and standard derivatives are insufficient for accurately describing vector displacement. Instead, one must use the *covariant derivative*, which accounts for curvature through the Christoffel symbol. This term varies across coordinate systems, making it equivariant rather than invariant.

The curvature tensor characterizes the intrinsic geometry of a manifold. While the full Riemann curvature tensor (rank-4) and the Ricci tensor (rank-2) provide detailed geometric information, computing and using them during training can be prohibitively expensive. We therefore focus on the Ricci scalar curvature, which is a diffeomorphism-invariant scalar summary of curvature. Although it is a coarse descriptor compared to tensor-valued curvature, it captures global geometric variations while remaining computationally tractable, making it well-suited for our curvature-matching objective.

Since all Riemannian manifolds enjoy diffeomorphism invariance, this property provides freedom in coordinate choice. Formally, a diffeomorphism is a smooth bijective map with differentiable inverse. Practically, it means that a vector's intrinsic properties remain unchanged even when expressed in a new coordinate basis. Consequently, one can always find a coordinate frame in which the manifold appears locally flat.

To uncover the latent manifold's geometry, one could:

- Solve the Einstein field equations to obtain the metric tensor.
- Propose a suitable *Ansatz* and verify that it satisfies the Einstein equations.
- Define a mapping function and derive the curved metric from a flat one using the Jacobian.

The third method is the most practical in deep learning. General solutions to Einstein's equations are unknown for arbitrary settings, and crafting a good *Ansatz* is difficult and task-dependent. However, the Jacobian-based approach is well-established: the curved metric is computed from the Jacobian and its inverse, composed with a flat metric.

Using this method, we analytically compute curvature for task-specific manifolds and compare their geometry through Ricci scalars.

We chose to extend GATE by replacing local perturbation alignment with Ricci curvature matching. As discussed in our Introduction, this provides several key benefits:

- It captures **global geometric structure**, rather than relying on limited local perturbations.
- It removes ambiguity in choosing "infinitesimal" scales—especially relevant when latent vector magnitudes vary or curvature is large.
- It eliminates the need for perturbation-based sampling and supports a **universal embedding space** that enables non-linear mappings and potential multi-modal extensions.

In summary, Ricci curvature offers a mathematically principled, empirically effective, and computationally viable means of aligning task-specific latent spaces in transfer learning.

B.3. The Definition of Riemannian Manifold

A curved space is complicated to comprehend in general. Since the late 19th century, there has been immense development in differential geometry to formally interpret curved spaces. One of the best-known intuitive geometries is Riemannian geometry. Riemannian geometry possesses a handful of useful mathematical properties that can be utilized in the real world. The formal definition of Riemannian geometry is as follows:

Definition B.1 (Riemannian Manifold)**.** A Riemannian metric on a smooth manifold M is a choice at each point $x \in M$ of a positive definite inner product $g_p : T_pM \times T_pM \to \mathbb{R}$ on T_xM. The smooth manifold endowed with the metric g is a Riemannian manifold, denoted (M, g).

As stated above, a Riemannian manifold is smooth and differentiable everywhere on the manifold, along with its derivatives. Moreover, a Riemannian manifold enjoys diffeomorphism invariance, induced by the Lie derivative $\mathcal{L}_X$. It can be readily observed that the composition of two different Lie derivatives forms a group, known as the diffeomorphism group. This isometry guarantees that coordinate choices can be made without altering the global geometry of the space.

$$X' = X'^{\mu} dX'_{\mu} = X'^{\mu} \frac{\partial X^{\nu}}{\partial X'^{\mu}} dX_{\nu} = X^{\nu} dX_{\nu} = X \tag{39}$$

As shown in Eq. 39, the transformed vector remains unchanged. Moreover, it is always possible to fix the transformed coordinates in a locally flat space.

$$\xi^{\mu} = \frac{\partial \xi^{\mu}}{\partial X^{\nu}} X^{\nu} \tag{40}$$

Where ξ^{μ} is a vector on a locally flat frame. To ensure the vector is on a flat frame, one must impose the following condition:

$$\frac{\partial^2}{\partial t^2} \xi^{\mu}(t) \equiv 0 \tag{41}$$

Since a vector is on a flat frame, it should be in free-falling motion, and thus its acceleration should be trivial. On a locally flat frame, the metric also reduces to the flat Euclidean metric.

$$g_{\mu\nu} = 1_{\mu\nu} \tag{42}$$

B.4. Covariance

A vector should transform consistently across any coordinate frame. However, if the space is no longer flat, the ordinary derivative no longer preserves this property. To address this, let us consider the derivative of a vector in a general curved space.

$$\partial_\mu \to \partial'_\mu = \frac{\partial x^\mu}{\partial x'^\nu}\partial_\nu \tag{43}$$

Where $\partial_\mu = \frac{\partial}{\partial x^\mu}$, the vector transformation can be written as follows:

$$\partial_\nu X^\mu \to \partial'_\nu X'^\mu = \frac{\partial x^\lambda}{\partial x'^\nu}\frac{\partial}{\partial x^\lambda}(\frac{\partial x'^\mu}{\partial x^\rho}V^\rho) \tag{44}$$

$$= \frac{\partial x'^\nu}{\partial x^\lambda}\Big(\frac{\partial x'^\rho}{\partial x^\nu}\partial^\lambda V^\rho + \frac{\partial^2 x'^\mu}{\partial x^\lambda \partial x^\rho}V^\rho\Big) \tag{45}$$

As shown above, the transformation of a vector on a curved space using an ordinary derivative is no longer covariant. Therefore, it is necessary to introduce an additional term to restore covariance, namely the affine connection. With this addition, one can define the covariant derivative, which replaces the ordinary derivative.

$$\nabla_\mu = \partial_\mu + \Gamma^\lambda{}_{\mu\nu} \tag{46}$$

By imposing the covariance condition on the covariant derivative,

$$\nabla_\lambda \to \nabla'_\lambda V'^\mu = \frac{\partial x^\rho}{\partial x'^\nu}\frac{\partial x'^\mu}{\partial x^\nu}\nabla_\rho V^\nu \tag{47}$$

one can derive the explicit form of the connection.

$$\nabla_\mu V^\nu = \partial_\mu V^\nu + \Gamma^\nu{}_{\mu\lambda}V^\lambda \tag{48}$$

Under coordinate transformation,

$$\frac{\partial}{\partial x'^\mu}(\frac{\partial x'^\nu}{\partial x^\lambda}V^\lambda) + \Gamma'^\nu{}_{\mu\sigma}V'^\sigma = \frac{\partial x^\rho}{\partial x'^\mu}\frac{\partial x'^\nu}{\partial x^\lambda}\partial_\rho V^\lambda + \frac{\partial x^\rho}{\partial x'^\mu}\frac{\partial^2 x'^\nu}{\partial x^\rho \partial x^\lambda}V^\lambda + \Gamma'^\nu{}_{\mu\sigma}V'^\sigma \tag{49}$$

Here, to make the derivative of a vector covariant, the following condition must be satisfied:

$$\frac{\partial x^\rho}{\partial x'^\mu}\frac{\partial^2 x'^\nu}{\partial x^\rho \partial x^\lambda}V^\lambda + \Gamma'^\nu{}_{\mu\sigma}V'^\sigma = \frac{\partial x^\rho}{\partial x'^\mu}\frac{\partial x'^\nu}{\partial x^\lambda}\Gamma^\lambda{}_{\rho\sigma}V^\sigma \tag{50}$$

Which is

$$\Gamma'^\nu{}_{\mu\sigma}(\frac{\partial x'^\sigma}{\partial x^\tau}V^\tau) = \frac{\partial x^\rho}{\partial x'^\mu}\frac{\partial'^\nu}{\partial x^\lambda}\Gamma^\lambda{}_{\rho\sigma}V^\sigma - \frac{\partial x^\rho}{\partial x'^\mu}\frac{\partial x^\rho}{\partial x'^\mu}\frac{\partial^2 x'^\nu}{\partial x^\rho \partial x^\lambda}V^\lambda \tag{51}$$

$$\Gamma'^\nu{}_{\mu\kappa}V^\tau = \frac{\partial x^\rho}{\partial x'^\kappa}\frac{\partial x^\rho}{\partial x'^\mu}\frac{\partial x'^\nu}{\partial x^\lambda}\Gamma^\lambda{}_{\rho\sigma}V^\sigma - \frac{\partial x^\tau}{\partial x'^\kappa}\frac{\partial x^\rho}{\partial x'^\mu}\frac{\partial^2 x'^\nu}{\partial x^\rho \partial x^\lambda}V^\lambda \tag{52}$$

This leads to the explicit form of how the Christoffel symbols transform under coordinate changes.

$$\Gamma'^\nu{}_{\mu\kappa} = \frac{\partial x^\tau}{\partial x'^\kappa}\frac{\partial x^\rho}{\partial x'^\mu}\frac{\partial x'^\nu}{\partial x^\lambda}\Gamma^\lambda{}_{\rho\tau} - \frac{\partial x^\tau}{\partial x'^\kappa}\frac{\partial x^\rho}{\partial x'^\mu}\frac{\partial^2 x'^\nu}{\partial x^\rho \partial x^\tau} \tag{53}$$

Since the Kronecker delta is a constant matrix, it is clear that its derivative must vanish. By applying the chain rule to the delta, one can derive the following relation, which simplifies the transformation rule described above.

$$\frac{\partial}{\partial x'^\mu}\delta^\nu_\kappa = \frac{\partial}{\partial x'^\mu}\frac{\partial x'^\nu}{\partial x'^\kappa} = \frac{\partial}{\partial x'^\mu}(\frac{\partial x^\tau}{\partial x'^\kappa}\frac{\partial x'^\nu}{\partial x^\tau}) = 0 = \frac{\partial x^\tau}{\partial x'^\kappa}\frac{\partial x^\rho}{\partial x'^\mu}\frac{\partial^2 x'^\nu}{\partial x^\rho \partial x^\tau} + \frac{\partial x'^\nu}{\partial x^\tau}\frac{\partial x'^\nu}{\partial x^\tau}\frac{\partial^2 x^\tau}{\partial x'^\mu \partial x'^\rho} \tag{54}$$

Finally, the transformation rule for the Christoffel symbols is given by:

$$\Gamma'^{\nu}{}_{\mu\kappa} = \frac{\partial x^\tau}{\partial x'^\kappa}\frac{\partial x^\rho}{\partial x'^\mu}\frac{\partial x'^\nu}{\partial x^\lambda}\Gamma^\lambda{}_{\rho\tau} + \frac{\partial x'^\nu}{\partial x^\tau}\frac{\partial^2 x^\tau}{\partial x'^\mu \partial x'^\rho} \tag{55}$$

By the same reasoning, one can easily determine how covariant derivatives act on differential forms.

$$\nabla_\mu V_\nu = \partial_\mu V_\nu - \Gamma^\lambda{}_{\mu\nu} V_\lambda \tag{56}$$

B.5. Explicit Form of Christoffel Symbol

The metric serves as the ruler of a given geometry; therefore, it should remain invariant with respect to position in a coordinate system. In the case of Euclidean space, this invariance is trivial to observe, as the metric is simply $\delta_{\mu\nu}$, a constant matrix.

$$\frac{\partial}{\partial x^\lambda}\delta_{\mu\nu} = 0 \tag{57}$$

However, in the curved case, the above principle must still hold to interpret the metric as a ruler. Nevertheless, this condition does not hold when using an ordinary derivative. Here, the covariant derivative comes into play, replacing the ordinary derivative. When taking the covariant derivative of the curved metric, the resulting term vanishes.

$$\nabla_\lambda g_{\mu\nu} = 0 \tag{58}$$

One can express this condition in terms of a flat metric combined with a diffeomorphism transformation factor.

$$g_{\mu\nu}(x) = \frac{\partial \xi^\lambda}{\partial x^\mu}\frac{\partial \xi^\rho}{\partial x^\nu}\delta_{\lambda\rho}(\xi) \tag{59}$$

Taking the derivative with respect to x on both sides, the equation becomes:

$$\begin{aligned}
\frac{\partial}{\partial x^\sigma} g_{\mu\nu}(x) &= \frac{\partial^2 x^\lambda}{\partial x^\sigma \partial x^\mu}\frac{\xi^\rho}{\partial x^\nu}\delta_{\lambda\rho} + \frac{\partial^2 \xi^\rho}{\partial x^\sigma \partial x^\nu}\frac{\partial \xi^\lambda}{\partial x^\mu}\delta_{\lambda\rho} && (60)\\
&= \frac{\partial^2 \xi^\rho}{\partial x^\sigma \partial x^\nu}\frac{\partial x^\tau}{\partial \xi^\rho}\frac{\partial \xi^\rho}{\partial x^\tau}\frac{\partial \xi^\lambda}{\partial x^\mu}\delta_{\lambda\rho} + \frac{\partial^2 \xi^\lambda}{\partial x^\sigma \partial x^\mu}\frac{\partial x^\tau}{\partial \xi^\lambda}\frac{\partial \xi^\lambda}{\partial x^\tau}\frac{\partial \xi^\rho}{\partial x^\nu}\delta_{\lambda\rho} && (61)\\
&= \frac{\partial^2 \xi^\rho}{\partial x^\sigma \partial x^\nu}\frac{\partial x^\tau}{\partial \xi^\rho} g_{\mu\tau} + \frac{\partial^2 \xi^\lambda}{\partial x^\sigma \partial x^\mu}\frac{\partial x^\tau}{\partial \xi^\lambda} g_{\tau\nu} && (62)
\end{aligned}$$

From Eq. 58, one can easily derive the explicit form of the Christoffel symbol in terms of the derivatives of the curved and flat coordinates.

$$\frac{\partial}{\partial x^\sigma} g_{\mu\nu} = \Gamma^\tau{}_{\sigma\mu} g_{\tau\nu} + \Gamma^\tau{}_{\nu\sigma} g_{\mu\sigma} \tag{63}$$

$$\Gamma^\tau{}_{\sigma\mu} = \frac{\partial^2 \xi^\lambda}{\partial x^\sigma \partial x^\mu}\frac{\partial x^\tau}{\partial \xi^\lambda}(x) \tag{64}$$

Since the metric should always be symmetric, the lower indices of the Christoffel symbol should also be symmetric. It is called a torsion-free condition. Furthermore, by utilizing a simple mathematical trick, one can obtain the Christoffel symbol in terms of the metric $g_{\mu\nu}$.

$$\begin{aligned}
\frac{\partial}{\partial x^\sigma} g_{\mu\nu} &= \Gamma^\tau{}_{\sigma\mu} g_{\tau\nu} + \Gamma^\tau{}_{\sigma\nu} g_{\mu\tau} && (65)\\
\frac{\partial}{\partial x^\mu} g_{\nu\sigma} &= \Gamma^\tau{}_{\mu\nu} g_{\tau\sigma} + \Gamma^\tau{}_{\mu\sigma} g_{\nu\tau} && (66)\\
\frac{\partial}{\partial x^\nu} g_{\sigma\mu} &= \Gamma^\tau{}_{\nu\sigma} g_{\tau\mu} + \Gamma^\tau{}_{\nu\mu} g_{\sigma\tau} && (67)
\end{aligned}$$

Adding the first two equations and subtracting the last one leads to:

$$\Gamma^\lambda{}_{\mu\nu} = \frac{1}{2} g^{\lambda\rho}\left(\frac{\partial}{\partial x^\mu} g_{\nu\rho} + \frac{\partial}{\partial x^\nu} g_{\rho\mu} - \frac{\partial}{\partial x^\rho} g_{\mu\nu}\right) \tag{68}$$

B.6. Geodesic Equations

The shortest path between two points is simple to define in flat space. However, in curved space, this notion becomes more complicated. The shortest path in a curved space is defined as a geodesic. There are several ways to derive the geodesic equation, one of which is by imposing the free-fall condition.

$$\frac{\partial^2 \xi^\mu(\tau)}{\partial \tau^2} = 0 \tag{69}$$

By a diffeomorphism, one can transform a coordinate into an arbitrary coordinate x.

$$0 = \frac{\partial}{\partial \tau}(\frac{\partial \xi^\mu}{\partial x^\nu}\frac{\partial x^\nu}{\partial \tau}) = \frac{\partial \xi^\mu}{\partial x^\nu}\frac{\partial^2 x^\nu}{\partial \tau^2} + \frac{\partial^2 \xi^\mu}{\partial x^\lambda \partial x^\nu}\frac{\partial x^\lambda}{\partial \tau}\frac{\partial x^\nu}{\partial \tau} \tag{70}$$

$$\frac{\partial^2 x^\rho}{\partial \tau^2} + \frac{\partial^2 \xi^\mu}{\partial x^\lambda \partial x^\nu}\frac{\partial x^\rho}{\partial \xi^\mu}\frac{\partial x^\lambda}{\partial \tau}\frac{\partial x^\nu}{\partial \tau} = \frac{\partial^2 x^\rho}{\partial \tau^2} + \Gamma^\rho{}_{\lambda\nu}\frac{\partial x^\lambda}{\partial \tau}\frac{\partial x^\nu}{\partial \tau} = 0 \tag{71}$$

Another way to derive the equation is by minimizing the distance in curved space.

$$S = \int \sqrt{g_{\mu\nu}\frac{dx^\mu}{d\tau}\frac{dx^\nu}{d\tau}}d\tau \tag{72}$$

By varying the above equation and requiring the variation to vanish, one can compute its minimum value, and after some tedious calculations, the geodesic equation can be obtained.

B.7. Riemann Curvature

The Riemann curvature can be defined through the concept of parallel transport of a vector. In flat space, a vector remains unchanged under parallel transport along any path. However, in curved space, the vector's outcome depends on the path taken. This leads to the idea of curvature as the difference between the results of transporting a vector along two different paths from the same starting point to the same endpoint. This difference quantitatively characterizes the curvature of the space.

$$[\nabla_\mu, \nabla_\nu]V^\lambda \tag{73}$$

Here, the bracket denotes the commutation relation between the entities. Since the covariant derivative acts as the generator of parallel transport, the equation can be interpreted as the vector V^λ being transported along two different paths: one generated by applying ∇_μ followed by ∇_ν, and the other by reversing the order. The resulting computation takes the form:

$$\begin{aligned}\nabla_\mu\nabla_\nu V^\lambda &= \partial_\mu(\nabla_\nu V^\lambda) + \Gamma^\lambda{}_{\mu\rho}\nabla_\nu V^\rho - \Gamma^\rho{}_{\mu\nu}\nabla_\rho V^\lambda \\ &= \partial_\mu\partial_\nu V^\lambda + \partial_\mu\Gamma^\lambda{}_{\nu\rho}V^\rho + \Gamma^\lambda{}_{\nu\rho}\partial_\mu V^\rho + \Gamma^\lambda{}_{\mu\rho}\partial_\nu V^\rho \\ &\quad + \Gamma^\lambda{}_{\mu\rho}\Gamma^\rho{}_{\nu\sigma}V^\sigma - \Gamma^\rho{}_{\mu\nu}\partial_\rho V^\lambda - \Gamma^\rho{}_{\mu\nu}\Gamma^\lambda{}_{\rho\sigma}V^\sigma\end{aligned} \tag{74}$$

Where,

$$[\nabla_\mu, \nabla_\nu]V^\lambda = (\partial_\mu\Gamma^\lambda{}_{\nu\rho} - \partial_\nu\Gamma^\lambda{}_{\mu\rho} + \Gamma^\lambda{}_{\mu\sigma}\Gamma^\sigma{}_{\nu\rho} - \Gamma^\lambda{}_{\nu\sigma}\Gamma^\sigma{}_{\mu\rho})V^\rho - 2\Gamma^\rho_{[\mu\nu]}\nabla_\rho V^\lambda \tag{75}$$

Since the connection is symmetric under the permutation of its lower indices, the last term in the above equation can be eliminated. We can then finally define the Riemann tensor.

$$R^\lambda{}_{\rho\mu\nu} := \partial_\mu\Gamma^\lambda{}_{\nu\rho} - \partial_\nu\Gamma^\lambda{}_{\mu\rho} + \Gamma^\lambda{}_{\mu\sigma}\Gamma^\sigma{}_{\nu\rho} - \Gamma^\lambda{}_{\nu\sigma}\Gamma^\sigma{}_{\mu\rho} \tag{76}$$

The Riemann curvature tensor possesses several useful properties.

$$\begin{aligned}&R_{\lambda\rho\mu\nu} = -R_{\lambda\rho\nu\mu} \\ &R_{\lambda\rho\mu\nu} = -R_{\rho\lambda\nu\mu} \\ &R_{\lambda\rho\mu\nu} = -R_{\nu\mu\lambda\rho} \\ &R_{\lambda\rho\mu\nu} + R_{\lambda\mu\nu\rho} + R_{\lambda\nu\rho\mu} = 0 \\ &\nabla_\sigma R_{\lambda\rho\mu\nu} + \nabla_\mu R_{\lambda\rho\nu\sigma} + \nabla_\nu R_{\lambda\rho\sigma\mu} = 0 \\ &\nabla_\sigma R_{\lambda\rho\mu\nu} + \nabla_\lambda R_{\rho\sigma\nu\mu} + \nabla_\rho R_{\sigma\lambda\mu\nu} = 0 \\ &R_{\lambda\rho\mu\nu} + R_{\rho\mu\lambda\nu} + R_{\mu\lambda\rho\nu} = 0\end{aligned} \tag{77}$$

With the curved metric $g_{\mu\nu}$, one can construct the Ricci curvature tensor and the Ricci scalar by contracting the first and third, and the second and fourth indices of the Riemann tensor, respectively.

$$\begin{aligned}&R_{\rho\nu} = g^{\lambda\mu}R_{\lambda\rho\mu\nu} \\ &R = g^{\rho\nu}R_{\rho\nu} = g^{\rho\nu}g^{\lambda\mu}R_{\lambda\rho\mu\nu}\end{aligned} \tag{78}$$

B.8. Quadratic Case for Computation Check

This entire sequence is indeed both tedious and complex. Therefore, we introduce the simplest case for each process to verify the validity of the code and the formulas. Here, we set the activation function to the quadratic of the input signal and maintain the number of layers at two. Under these conditions, the transformed vector becomes:

$$x'^i = W^{(2)i}{}_j (W^{(1)}x + b^{(1)})^{2j} + b^{(2)i} \tag{79}$$

Now, the Jacobian can be easily derived from the above equation.

$$J^i{}_j = W^{(2)i}{}_k 2(W^{(1)}x + b^{(1)})^k{}_m W^{(1)m}{}_j \tag{80}$$

Here, $(W^{(1)}x + b^{(1)})^i{}_j$ has a diagonal matrix form as follows:

$$(W^{(1)}x + b^{(1)})^i{}_j = \begin{cases} W^{(1)i}{}_k x^k + b^{(1)i} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{81}$$

Then, the metric can be written in the following form:

$$g_{ij} = W^{(2)l}{}_k 2(W^{(1)}x + b^{(1)})^k{}_m W^{(1)m}{}_i W^{(2)}{}_{ln} 2(W^{(1)}x + b^{(1)})^n{}_o W^{(1)o}{}_j \tag{82}$$

Finally, one can compute the derivative of the metric.

$$\begin{aligned} \partial_k g_{ij} = \quad & 4|W^{(2)o}{}_q|^2 (W^{(1)})^{2q}{}_p \delta^p{}_{ik} W^{(1)n}{}_o (W^{(1)}x + b^{(1)})^o{}_j \\ & +4|W^{(2)}{}_{nq}|^2 W^{(1)q}{}_m (W^{(1)}x + b^{(1)})^m{}_i (W^{(1)})^{2n}{}_p \delta^p{}_{jk} \end{aligned} \tag{83}$$

where $|W|^2 = W^T W$ and $W^2 = W^i{}_k W^k{}_j$.

B.8.1. 2-Dim Simplest Example for Square Activation

To cross-check the computation results, we hereby introduce the simplest example for metric computation in 2D. The weights and biases for each layer are defined as follows:

$$W^{(1)i}{}_j = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \tag{84}$$

$$W^{(2)i}{}_j = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \tag{85}$$

$$b^{(1)i} = \begin{pmatrix} 3 \\ 4 \end{pmatrix} \tag{86}$$

$$x = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{87}$$

Then, the Jacobian can be computed as expressed in Eq. 80. We will break down the Jacobian piece by piece and verify the validity of the equation.

$$2(W^{(1)}x + b^{(1)})^m{}_j = \begin{pmatrix} 16 & 0 \\ 0 & 30 \end{pmatrix} \tag{88}$$

By multiplying the weights from both layers, the equation becomes:

$$J^i{}_j = \begin{pmatrix} 620 & 880 \\ 832 & 1184 \end{pmatrix} \tag{89}$$

Finally, the metric can be expressed as the square of the Jacobian.

$$g_{ij} = (J^T)_{ik} J^k{}_j = \begin{pmatrix} 1076624 & 1530688 \\ 1530688 & 2176256 \end{pmatrix} \tag{90}$$

As shown above, the metric is symmetric.

B.8.2. 2-DIM SIMPLEST EXAMPLE FOR SILU ACTIVATION

In practice, a simple square activation is insufficient to capture the complex structure of curved space. Therefore, we adopt the SiLU activation function to better express the model's geometric structure. The SiLU function behaves similarly to the ReLU activation but enjoys smoothness across the entire domain.

$$\text{SiLU}(x) = x \times \text{LS}(x) = \frac{x}{1+e^{-x}} \tag{91}$$

Since the SiLU activation contains the inverse of the exponential function in its expression, the input values should be kept smaller than 1 to prevent the activation from converging to a trivial value.

$$W^{(1)i}{}_j = \begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{pmatrix} \tag{92}$$

$$W^{(2)i}{}_j = \begin{pmatrix} 0.5 & 0.6 \\ 0.7 & 0.8 \end{pmatrix} \tag{93}$$

$$b^{(1)i} = \begin{pmatrix} 0.3 \\ 0.4 \end{pmatrix} \tag{94}$$

$$x = \begin{pmatrix} 0.1 \\ 0.2 \end{pmatrix} \tag{95}$$

Using the input example set described above, one can compute the explicit equations, with the results as follows. We will first introduce the main components used in the calculation. One key component is the sigmoid function, which is utilized in the SiLU computation.

$$\sigma^i = \begin{pmatrix} 0.5866 \\ 0.6248 \end{pmatrix} \tag{96}$$

Another component is the diagonalized exponential term, which appears in the derivative of the vector exponential.

$$E^i{}_j = \begin{pmatrix} 0.7047 & 0 \\ 0 & 0.6005 \end{pmatrix} \tag{97}$$

By combining the two expressions above with the weights and biases, it is possible to obtain the full Jacobian.

$$J^i{}_j = \begin{pmatrix} 0.1676 & 0.2458 \\ 0.2257 & 0.3322 \end{pmatrix} \tag{98}$$

Finally, by squaring the Jacobian, the induced metric g_{ij} can be defined.

$$g_{ij} = (J^T)_{ik} J^k{}_j = \begin{pmatrix} 0.0790 & 0.1161 \\ 0.1161 & 0.1708 \end{pmatrix} \tag{99}$$

As shown above, the metric is well-defined and forms a symmetric structure in this setup as well.

B.9. General Analytic computation strategy

A deep learning model is composed of multiple smooth layers. Therefore, if differentiable activation functions are used, it becomes possible to compute the curvature tensor of the curved space induced by the model. However, when the model consists of many layers, it becomes convenient to define building blocks that allow the full Jacobian to be computed by simply multiplying them. These building blocks can be expressed in terms of the weights and biases of each layer. Starting from the full Jacobian, and by applying the chain rule, the Jacobian can be decomposed into the Jacobians of individual layers.

$$J^i{}_j = \frac{dx'^i}{dx^j} = \frac{dx'^i}{dx^{(n-1)k_{n-1}}} \frac{dx^{(n-1)k_{n-1}}}{dx^{(n-2)k_{n-2}}} \cdots \frac{dx^{(1)k_1}}{dx^j} \tag{100}$$

Here, n denotes the layer index of the transfer module in the model, as illustrated in Figure **??**. Therefore, when similar mathematical structures appear across layers—as is often the case—it becomes possible to define a fundamental building

block of the full Jacobian using the Jacobian of a single representative layer. In our setup, each layer follows a linear MLP structure with the SiLU activation function. The fundamental Jacobian block can then be expressed in the following form:

$$\begin{aligned} \frac{dx^{(n+1)i}}{dx^{(n)j}} &= W^{(n+1)i}{}_{k}(((x^{(n)k})e^{-x^{(n)k}} \times \mathrm{LS}(x^{(n)k}) + 1)\mathrm{LS}(x^{(n)k}))^{k}{}_{j} \\ &= (W^{(n+1)i}{}_{j}\sigma^{i} + (W^{(n+1)}x^{(n)} + b^{(n+1)})^{i}W^{(n+1)a_3}{}_{j}E^{i}{}_{a_3}(\sigma^{2})^{i}) \end{aligned} \tag{101}$$

Here, $W^{(n)i}{}_{j}$ and $b^{(n)i}$ are weights and biases of n-th layer in the transfer module. The new notations introduced in the equation above are defined as follows. First, $\mathrm{LS}(x)$ denotes the logistic function and σ^i and $E^i{}_l$ are expressed as follows:

$$\sigma^{i} = \frac{1}{1 + e^{-(W^{(n)i}{}_{j}x^{j} + b^{(n)i})}}, \qquad \mathrm{E}^{i}{}_{l} \equiv (e^{-(W^{l}{}_{j}x^{j} + b^{j})})^{i}{}_{l} = \begin{cases} e^{-(W^{l}{}_{j}x^{j} + b^{l})} & \text{if } l = i \\ 0 & \text{if } l \neq i \end{cases} \tag{102}$$

$(\sigma^2)^i$ denotes the element-wise square of σ^i. By utilizing Eq. 101, it is now possible to compute the full Jacobian of the transfer module. The induced metric can then also be specified by Eq. 4.

C. Base Graph Neural Network Model

In general, molecule is represented in a graph form. Therefore, in order to handle molecule dataset, it is inevitable to utilize graph neural networks. We chose directional message passing network (DMPNN) (Yang et al., 2019) for our backbone, since it outperforms other GNN architectures in molecular domain. Given a graph, DMPNN initializes the hidden state of each edge (i, j) based on its edge feature E_{ij} with node feature X_i. At each step t, directional edge summarizes incident edges as a message m_{ij}^{t+1} and updates its hidden state to h_{ij}^{t+1}.

$$m_{ij}^{t+1} = \sum_{k \in \mathcal{N}(i) \setminus j} h_{ki}^{t} \tag{103}$$

$$h_{ij}^{t+1} = \mathrm{ReLU}(h_{ij}^{0} + W_e m_{ij}^{t+1}) \tag{104}$$

Where $\mathcal{N}(i)$ denotes the set of neighboring nodes and W_e a learnable weight.he hidden states of nodes are updated by aggregating the hidden states of incident edges into message m_i^{t+1}, and passing its concatenation with the node feature X_i into a linear layer followed by ReLU non-linearity

$$m_{i}^{t+1} = \sum_{j \in \mathcal{N}(i)} h_{ij}^{t} \tag{105}$$

$$h_{i}^{t+1} = \mathrm{ReLU}(W_n \mathrm{concat}(X_i, m_i^{t+1})) \tag{106}$$

Similarly, W_n denotes a learnable weight. Assuming DMPNN runs for T timesteps, we use $(X_{out}, E_{out}) = \mathrm{GNN}(A, X, E)$ to denote the output representation matrices containing hidden states of all nodes and edges, respectively (i.e., $X_{out,i} = h_i^T$ and $E_{out,ij} = h_{ij}^T$).

For graph-level prediction, the node representations after the final GNN layer are typically sum-pooled to obtain a single graph representation $h_G = \sum_i h_i$, which is then passed to a FFN prediction layer.

D. Architecture and Hyperparameters

The model architecture consists of five distinct neural networks, with their parameter sizes summarized in Table 1 and 2. As illustrated in 5, each task comprises an embedding network, encoder network, transfer network, inverse transfer network, and head network.

The embedding network, denoted as $f_m(x)$, adopts the DMPNN (Directed Message Passing Neural Network) architecture with a depth of 3. It converts the input molecular representation x into a latent representation a in the embedding space. The input vector to the embedding module is constructed as follows, using the same featurization scheme as (Yang et al., 2019): atom features are represented using a 134-dimensional one-hot encoded vector that captures atomic properties such as type, degree, formal charge, hybridization, and aromaticity. Bond features are encoded as a 149-dimensional one-hot vector reflecting bond type, conjugation, ring membership, stereochemistry, and atom-pair-derived descriptors.

The encoder network follows a bottleneck architecture implemented as an autoencoder with multilayer perceptrons (MLPs). The output from the encoder, $f_e(a)$, is then passed to both the transfer network and the head network for subsequent processing.

The output of transfer network $f_t(z)$, denoted as m, is used to calculate consistency loss. The induced flat metrics $\eta_{(s)ij}$ and $\eta_{(t)ij}$ from the source and target mappings are iterated K times, with $K = 2$ in our setup. m is also fed into inverse transfer network, so that the output from inverse transfer network $f_i(m)$ can be used to calculate autoencoder loss. Both modules are utilized to compute mapping, metric and curvature losses. The output from head network, $f_h \circ f_i(m)$, is used to calculate regression loss and mapping loss. We trained the model for 1000 epochs with batch size 512 while using AdamW (Loshchilov & Hutter, 2017) for optimization with learning rate 5e-5. The hyperparameters for $\alpha, \beta, \gamma, \delta, \epsilon$ are 0.1, 0.1, 0.2, 0.1, 0.2 respectively.

Table 1. Common Network Parameters

network	layer	input, output size	hidden size	dropout
embedding	DMPNN	[134, 149], 100	200	0
encoder	MLP layer	100, 50	50	0.2
transfer	MLP layer	50, 50	50,50,50	0
inverse transfer	MLP layer	50, 50	50,50,50	0
head	MLP layer	50, 1	25,12	0.2

E. Detailed Explanation of Datasets and Experimental Setups

E.1. Datasets

We utilized 14 different molecular property datasets sourced from three open-access databases, as detailed in Table 4 and the descriptions below, for the evaluation of GEAR. Prior to training, the datasets were carefully curated to remove entries with incorrectly specified units, typographical errors, or extreme measurement conditions. All datasets were normalized using their respective means and standard deviations to ensure consistency during training.

From these datasets, we selected 23 source–target task pairs, considering the number of data points available in each dataset to maintain balance. Additionally, to ensure a fair and unbiased evaluation, we deliberately selected task pairs exhibiting a wide range of correlations, as illustrated in Figure 6.

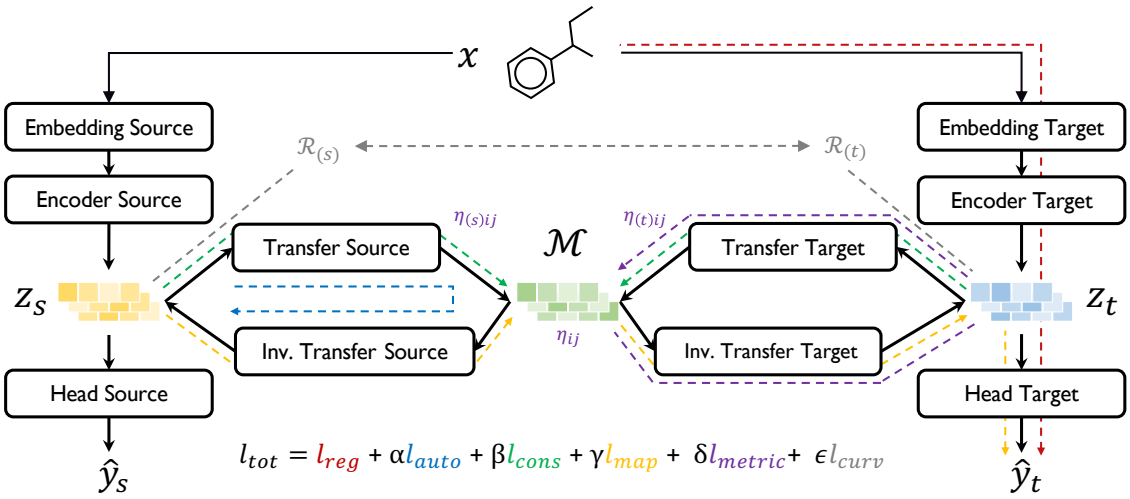

Figure 5. Detailed schematics of GEAR with specific loss function components.

Table 2. Task Specific Encoder Parameters

Tasks	Random Split Encoder Parameters	Scaffold Split Encoder Parameters
hv ← **ds**	[200, 200], [200, 200]	[300], [300]
as ← **bp**	[200, 200], [200, 200]	[300, 300], [300, 100]
ds ← **kri**	[200], [200, 200]	[150, 100], [300]
hv ← **vs**	[200, 200], [200, 200]	[200], [200, 200, 200]
vs ← **hv**	[200, 200], [200, 200]	[200, 200, 200], [200]
st ← **as**	[200, 200, 200], [200, 200, 200]	[200, 200, 200], [200, 200, 200]
vs ← **bp**	[200, 200, 200], [200, 200, 200]	[200, 200, 200], [200, 200, 200]
dk ← **ef**	[200, 200], [200]	[200], [200]
as ← **ccs**	[200, 200], [200]	[200], [200]
ct ← **bp**	[200], [200, 200, 200]	[200], [200]
ccs ← **kri**	[200, 200], [200]	[200, 200], [200, 200]
hv ← **bp**	[100, 100], [100]	[200, 200], [200, 200]
vs ← **ccs**	[200, 200], [200]	[200, 200], [200]
hv ← **kri**	[200, 200], [200, 200]	[200], [200, 200, 200]
ct ← **kri**	[100, 100, 100], [100, 100, 100]	[200], [200, 200]

Table 3. Hyperparameters

learning rate	0.00005
optimizer	AdamW
batch size	512
epoch	1000
$\alpha, \beta, \gamma, \delta, \epsilon$	0.1, 0.1, 0.2, 0.1, 0.2

Finally, we provide an explicit description of the physical meaning associated with each dataset to facilitate better understanding and context.

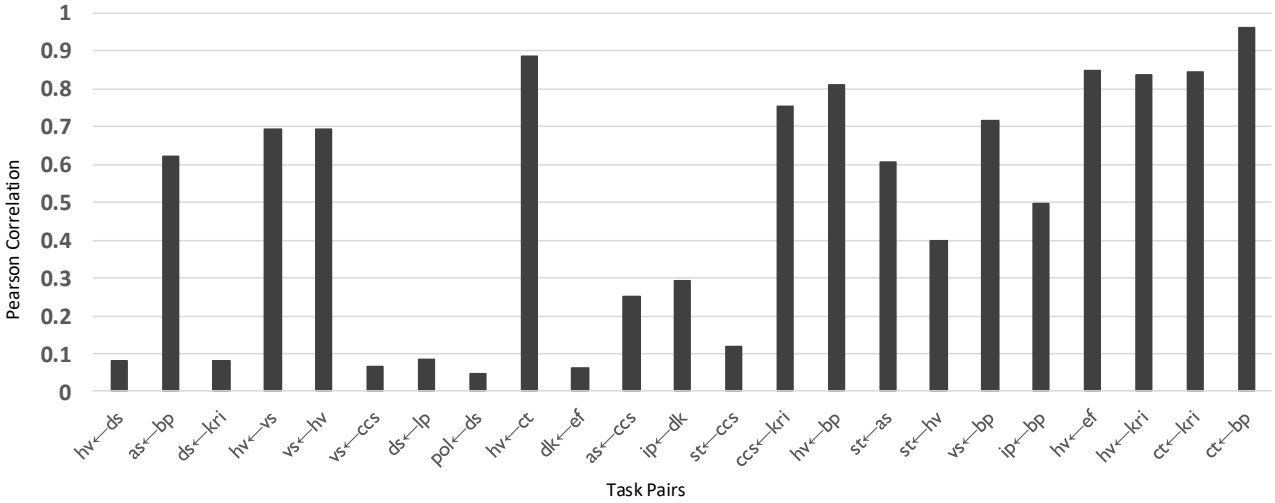

Figure 6. Pearson correlation between overlapping data points in target dataset and source dataset.

- **AS** : The solute dipolarity/polarizability.
- **BP** : The temperature at which this compound changes state from liquid to gas at a given atmospheric pressure.

Table 4. Detailed information about the datasets.

name	acronym	source	count	mean	std
Abraham Descriptor S	AS	Ochem	1925	1.05	0.68
Boiling Point	BP	Pubchem	7139	198.99	108.88
Collision Cross Section	CCS	Pubchem	4006	205.06	57.84
Critical Temperature	CT	Ochem	242	626.04	120.96
Dielectric Constant	DK	Ochem	1007	0.80	0.41
Density	DS	Pubchem	3079	1.07	0.29
Enthalpy of Fusion	EF	Ochem	2188	1.32	0.32
Ionization Potential	IP	Pubchem	272	10.00	1.63
Kovats Retention Index	KRI	Pubchem	73507	2071.20	719.34
Log P	LP	Pubchem	28268	11.17	9.89
Polarizability	POL	CCCB	241	0.84	0.26
Surface Tension	ST	Pubchem	379	29.01	10.36
Viscosity	VS	Pubchem	294	0.47	0.87
Heat of Vaporization	HV	Pubchem	525	43.77	18.08

- **CCS** : The effective area for the interaction between an individual ion and the neutral gas through which it is traveling.
- **CT** : The temparature when no gas can become liquid no matter how high the pressure is.
- **DK** : The ratio of the electric permeability of the material to the electric permeability of free space.
- **DS** : The mass of a unit volume of a compound.
- **EF** : The change in enthalpy resulting from the addition or removal of heat from 1 mole of a substance to change its state from a solid to a liquid.
- **IP** : The amount of energy required to remove an electron from an isolated atom or molecule.
- **KRI** : The rate at which a compound is processed through a gas chromatography column.
- **LP** : Logarithmic form of the ratio of concentrations of a compound in a mixture of octanol and water at equilibrium.
- **POL** : The tendency of matter, when subjected to an electric field, to acquire an electric dipole moment in proportion to that applied field.
- **ST** : The property of the surface of a liquid that allows it to resist an external force
- **VS** : A measure of a fluid's resistance to flow.
- **HV** : The quantity of heat that must be absorbed if a certain quantity of liquid is vaporized at a constant temperature.

E.2. Experimental Setups

For the evaluation of Sim-GEAR, we compared its performance against seven benchmark models: GEAR, GATE, STL, MTL, KD, global structure-preserving loss-based KD (GSP-KD), and transfer learning (implemented either by retraining the entire model or only the head network). All baselines share the same base architecture, with only minor method-specific modifications.

GEAR and GATE use encoder and head networks that are nearly identical to those of Sim-GEAR. However, GEAR imposes a constraint on the transfer module requiring consistent input and output dimensions across layers. To satisfy this, the hidden dimensions were adjusted to [50, 50, 50] instead of [100, 100, 100]. All other hyperparameters follow the original implementation (Ko et al., 2023b).

In the MTL setting, the backbone and bottleneck layers are shared across tasks, while separate task-specific head networks are employed. For the KD baseline, latent representations from the bottleneck layer are used as distillation targets, with the distillation loss weighted by 0.1.

Graph Contrastive Representation Distillation (G-CRD) originally combines contrastive and global structure-preserving (GSP) losses (Joshi et al., 2022). Since contrastive loss is not suitable for regression tasks, we retain only the GSP component. In the resulting GSP-KD setup, pairwise distances between node features from the final backbone layer are used as distillation targets. The GSP-based distillation loss is also weighted by 0.1.

All models are trained for up to 600 epochs, with the best-performing model selected based on early stopping.