Graph Neural Networks in Modern AI-aided Drug Discovery

Odin Zhang[1,#], Haitao Lin[2,#], Xujun Zhang[1,#], Xiaorui Wang[1,#], Zhenxing Wu[1], Qing Ye[1], Weibo Zhao[1], Jike Wang[1], Kejun Ying[3], Yu Kang[1], Chang-yu Hsieh[1,*], Tingjun Hou[1,*]

[1]College of Pharmaceutical Sciences, Zhejiang University, Hangzhou 310058, Zhejiang, China

[2]School of Engineering, Westlake University, Hangzhou 310024, Zhejiang, China

[3]T. H. Chan School of Public Health, Harvard University, Boston, MA, USA

[#]Equal Contributions

Corresponding authors

Tingjun Hou

E-mail: tingjunhou@zju.edu.cn

Chang-Yu Hsieh

E-mail: kimhsieh@zju.edu.cn

Abstract

Graph neural networks (GNNs), as topology/structure-aware models within deep learning, have emerged as powerful tools for AI-aided drug discovery (AIDD). By directly operating on molecular graphs, GNNs offer an intuitive and expressive framework for learning the complex topological and geometric features of drug-like molecules, cementing their role in modern molecular modeling. This review provides a comprehensive overview of the methodological foundations and representative applications of GNNs in drug discovery, spanning tasks such as molecular property prediction, virtual screening, molecular generation, biomedical knowledge graph construction, and synthesis planning. Particular attention is given to recent methodological advances, including geometric GNNs, interpretable models, uncertainty quantification, scalable graph architectures, and graph generative frameworks. We also discuss how these models integrate with modern deep learning approaches, such as self-supervised learning, multi-task learning, meta-learning and pre-training. Throughout this review, we highlight the practical challenges and methodological bottlenecks encountered when applying GNNs to real-world drug discovery pipelines, and conclude with a discussion on future directions.

Keywords: Graph neural network; Molecular graph; Deep learning; AI-aided drug discovery; Molecular modeling

Contents

1. Introduction

1.1 Graph is a Natural Representation in Drug Discovery

Drug discovery is a resource-intensive and time-consuming endeavor. It is estimated that bringing a new drug to market requires over two billion US dollars in investment and typically spans more than a decade of research and development[1]. The high costs and risks associated with this process have limited the spectrum of diseases that can be effectively treated through pharmacological interventions. As such, any methodological breakthrough in drug discovery, particularly those that reshape foundational paradigms, has attracted considerable attention from both academic and industrial communities[2]. Over time, drug discovery has evolved from early reliance on serendipitous discovery to the advent of high-throughput screening and, more recently, to mechanism-based rational design, progressing steadily toward enhanced controllability and specificity[3].

In recent years, the rapid development of artificial intelligence (AI) has infused new momentum into drug discovery, accelerating the transition from rule-based heuristics to data-driven strategies. The 2024 Nobel Prize in Chemistry, awarded breakthroughs in protein design, highlights the transformative potential of incorporating AI into the standard practices of drug dsicovery[4]. Beyond algorithmic advances, the growing availability of high-quality experimental data, from structural biology to multi-omics, has further solidified the foundation for AI applications. Databases such as the Protein Data Bank[5] (PDB) and omics resources like The Cancer Genome Atlas (TCGA)[6] provide large-scale structural, transcriptomic, and proteomic data essential for data-driven modeling. In this context, AI-aided drug discovery (AIDD) is emerging as a system-level paradigm, encompassing target identification, hit discovery, lead optimization, and early risk mitigation[7]. AIDD is primarily deployed in the early stages of drug development, and its key objective is to find and improve the specificity and developability of candidate molecules while minimizing toxicity and the risk of clinical trial failure. Collectively, these advancements contribute to improved efficiency and reduced costs in the drug development pipelines[8].

To fully harness the potential of AI in drug discovery, molecular structures must first be encoded into machine-interpretable representations. To this end, a variety of molecular representation strategies have been developed, including string-based linearizations (e.g., SMILES[9] and SELFIES[10]), molecular fingerprints[11], and Graphs-based and 3D voxel grid representations[12], as shown in **Figure 1A**. When coupled with architectures like DNNs, CNNs[13], and Transformers[14], these representations have shown promising results in tasks such as quantitative structure–activity relationship (QSAR) modeling and scoring function development[15]. However, each representation has inherent limitations. Fingerprints and descriptors, which rely on expert-defined features, often suffer limited expressiveness due to their dependence on predefined statistical or physicochemical patterns[16]. String-based methods linearize molecular topology, facilitating compatibility with sequence-based models, but they struggle to capture structural symmetry, reversibility, and long-range dependencies[17].

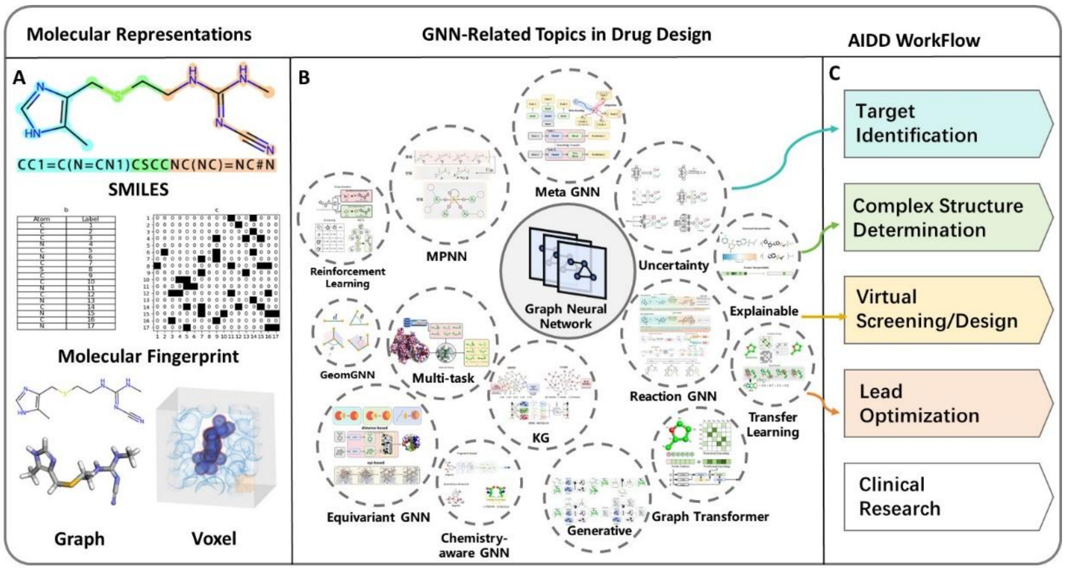

Figure 1. Overview of molecular representations and graph neural network (GNN) applications in drug discovery. (A) Common molecular representations; (B) GNN-related topics in AI-aided Drug Design; (c) AIDD workflow, with colord stages indicating where GNNs can be integrated.

Ontologically, molecules are inherently graph-structured objects, where xatoms constitute nodes and chemical bonds serve as edges, with additional three-dimensional (3D) coordinates defining their conformational geometry. This formulation naturally aligns with the modeling paradigm of graph neural networks (GNNs), which have shown exceptional promise in molecular modeling tasks. GNNs operate through local message-passing mechanisms to capture atomic coordination patterns and hierarchically aggregate them to encode molecular-level semantics. With the incorporation of equivariant or invariant operators (e.g., SO(3)-equivariant GNNs), GNNs can model stereochemistry, charge distributions, and spatial configurations in a symmetry-aware manner. Compared to traditional "flattened" representations, graph-based representations offer three key advantages: (i) physical consistency, preserving topological and geometric information; (ii) representational universality, enabling deployment across tasks such as property prediction, generative modeling, and pharmacological profiling; and (iii) enhanced interpretability through subgraph attention, edge attribution, and structural saliency maps.

As a result, graph representations are increasingly becoming the standard in molecular modeling. GNNs have been widely adopted by pharmaceutical companies (e.g., Roche, Novartis), AI-driven biotech startups (e.g., BenevolentAI, Insilico Medicine), and research communities supported by open datasets such as OGB-Mol[18] and GEOM[19]. These developments reflect not only the central role of GNNs in

AIDD but also the growing need for specialized GNN architectures tailored to diverse biomedical applications.

1.2. Scope and Organization of This Review

This review aims to provide a comprehensive and structured overview of GNN in modern AIDD, with a focus on their methodological foundations and practical challenges. In contrast to existing surveys that often center on specific tasks or empirical comparisons, we approach the subject from a model-centric perspective, examining the conceptual underpinnings of GNNs and their roles across diverse molecular design tasks, as shown in **Figure 1B**. Special attention is given to recent advances in large-scale GNNs, geometric deep learning, interpretability, uncertainty quantification, graph generative modeling, and reinforcement learning (RL). We further discuss how these architectures integrate with contemporary deep learning paradigms, such as self-supervised learning, multi-task learning, pretraining, and meta-learning, to improve data efficiency, generalization, and robustness in real-world drug discovery settings. This review highlights a unified modeling framework that harmonizes mathematical rigor with chemical intuition. It is intended for researchers in chemistry and biology applying AI to drug discovery, as well as those in computational sciences developing models for biochemical applications.

By highlighting the underlying logic and structural dependencies that connect seemingly disparate tasks, we aim to help readers develop a coherent conceptual framework that facilitates the effective adaptation of GNN-based techniques to address diverse challenges in drug discovery. The remainder of this review is organized as follows. Section 2 presents the methodological foundations of GNNs: (2.1) core principles of graph learning, including spectral and spatial frameworks; (2.2) symmetry-aware design principles, covering invariant and equivariant models for 3D molecular systems; (2.3) scalability challenges in large GNNs, such as oversmoothing, over-squashing, and pretraining strategies; (2.4) graph generation and structure editing methods; (2.5) graph editing and reinforcement learning techniques. Section 3 discusses molecular property prediction, including interpretability and uncertainty quantification in GNNs. Section 4 addresses virtual screening, including binding site prediction, protein-ligand docking, affinity scoring, and advances in structure prediction models such (e.g., AlphaFold). Section 5 presents GNN-based generative and reinforcement learning methods for molecular design and optimization. Section 6 explores the use of GNNs in knowledge graph construction and reasoning, particularly for biomedical entities. Section 7 focuses on chemical synthesis modeling, covering retrosynthesis, reaction condition prediction, and synthesis planning from a molecular generation perspective. Together, these sections are designed to connect theoretical models with practical applications, offering a two-way bridge between fundamental methodologies and their real-world deployment in drug discovery.

2. Advances in Graph Neural Networks

2.1 General Graph Theory

In principle, GNNs can be broadly categorized into two methodological paradigms based on their underlying theoretical foundations: **spectral-based** and **spatial-based** approaches, as illustrated in the **Figure 2**. Spectral methods extend the concept of convolution from Euclidean domains, which is used in traditional convolutional neural networks (CNNs), to arbitrary graph structures by leveraging spectral graph theory and graph Fourier analysis[20]. Representative works include SCNN[21], ChebNet[22], and GCN[23]. These models typically define graph convolution operations via the eigendecomposition of the graph Laplacian, leading to mathematically elegant yet relatively inflexible architectural designs. In contrast, spatial-based methods aggregate information directly from a node's local neighborhood using learnable functions that mimic the localized receptive fields of CNNs[24]. This approach enables efficient and intuitive modeling of local node interactions. Among spatial methods, the Message Passing Neural Networks (MPNN) framework[25] has emerged as a general abstraction, unifying the vast majority of modern GNN architectures, including GraphSAGE[26], GIN[27], and GAT[28]. Most contemporary graph deep learning libraries, such as PyG[29] and DGL[30], are built upon this framework, facilitating widespread adoption and practical implementation. Due to its conceptual clarity and ecosystem support, spatial-based graph convolution has become the predominant paradigm in chemistry-related tasks. In the first part of this section, we briefly review the core principles of spectral graph convolution to provide historical context. The remainder of this chapter focuses on spatial-based methods, with particular emphasis on MPNN-based frameworks and their applications in molecular modeling and drug design. The summary of these basic GNN architectures are provided in **Table 1**.

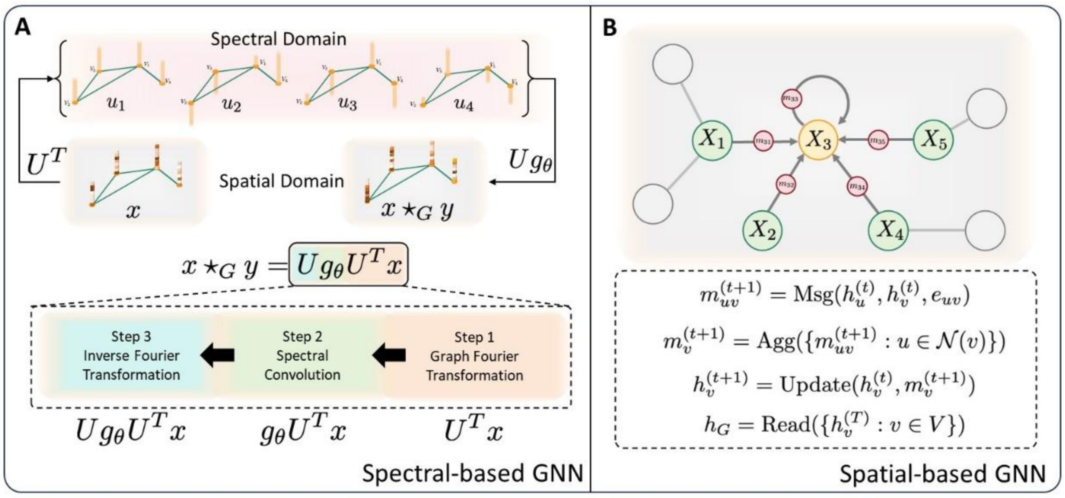
A
Spectral Domain
u_1
u_2
u_3
u_4
U^T
Ug_θ
Spatial Domain
x
$x \star_G y$
$x \star_G y = Ug_\theta U^T x$
Step 3 Inverse Fourier Transformation
Step 2 Spectral Convolution
Step 1 Graph Fourier Transformation
$Ug_\theta U^T x$
$g_\theta U^T x$
$U^T x$
Spectral-based GNN
B
X_1
X_2
X_3
X_4
X_5
m_{31}
m_{32}
m_{33}
m_{34}
m_{35}
$m_{uv}^{(t+1)} = \mathrm{Msg}(h_u^{(t)}, h_v^{(t)}, e_{uv})$
$m_v^{(t+1)} = \mathrm{Agg}(\{m_{uv}^{(t+1)} : u \in \mathcal{N}(v)\})$
$h_v^{(t+1)} = \mathrm{Update}(h_v^{(t)}, m_v^{(t+1)})$
$h_G = \mathrm{Read}(\{h_v^{(T)} : v \in V\})$
Spatial-based GNN

Figure 2. Comparison between spectral-based and spatial-based GNNs. (A) Spectral-based GNNs perform convolution in the spectral domain by applying learnable filters to the graph Laplacian

eigenbasis; (B) Spatial-based GNNs operate directly in the spatial domain by propagating and aggregating messages across graph neighborhoods. The formulas are presented in the MPNN framework.

Table 1. Overview of Spectral- and Spatial-based GNNs.

Category	Models	Keywords
Spectral-based	SCNN[21]	g_θ is a parameterized diagonal matrix
	ChebNet[22]	Approximate g_θ with a fixed-order polynomial.
	GCN[23]	Only keeping 1st and 2nd polynomial
Spatial-based	MPNN[25]	General Spatial-based GNN framework
	GraphSAGE[26]	Sample and aggregate locally
	GIN[27]	Discuss the upper bound of GNNs
	GAT[28]	Introduce attention mechanism in GNNs

2.1.1 Spectral-based GNN: Begin with Spectral Graph Theory

Convolution, broadly defined, describes how an input signal is modulated or filtered through the intrinsic characteristics of a given system, serving as a powerful mechanism for extracting informative features from raw input data. In classical signal processing, Fourier transforms are widely employed to project time-domain signals into the frequency domain, facilitating separate analysis of high- and low-frequency components, for instance, distinguishing between dolphin echolocation and high-pitched buzzes[31]. Similarly, in the context of graph-structured data, spectral methods seek to transform signals defined over nodes (i.e., signals in the spatial domain) into a spectral representation governed by the graph's topology. The ultimate goal is to design convolutional filters that encode the graph's frequency preferences, which can then be applied to extract and amplify meaningful node features from a given signal vector.

The Laplacian operator (Δ) offers a natural mathematical formulation for characterizing graph structures. Conceptually, the (generalized) Laplacian quantifies the difference between the value of a function at a given node and the average value across its local neighborhood. In image processing, the Laplacian kernel is commonly employed to detect local variations with high sensitivity. A classic example is:

$$L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

When applied to a particular pixel, this operator effectively subtracts a weighted center value from the sum of its neighbors, emphasizing local intensity changes. Translating this concept to graphs, the graph Laplacian is defined analogously as:

$$L = D - A$$

where A denotes the adjacency matrix (encoding connectivity) and D the degree matrix (encoding local node degree). Linear algebra dictates that L admits an eigendecomposition with n orthonormal eigenvectors and non-negative eigenvalues. Let U denote the matrix of eigenvectors (after normalization), satisfying:

$$UU^T = I$$

Each column of U forms an orthogonal basis vector, which corresponds to the graph's harmonic components in the spectral domain. In this framework, multiplying a signal vector by U projects it from the spatial domain into the spectral domain. To perform spectral graph convolution, a filter function g is defined in the frequency domain to reweight (i.e., amplify or suppress) specific spectral components. In summary, the graph convolution of node features x with filter g can be decomposed into two steps: (1) projecting x into the spectral domain, and (2) performing point-wise multiplication with the spectral representation of g, followed by projection back to the node domain:

$$x \star_G g = \mathcal{F}^{-1}\big(\mathcal{F}(x) \odot \mathcal{F}(g)\big) = U(U^T g \odot U^T x)$$

where $\star_G$ denotes graph convolution, $\mathcal{F}$ is graph Fourier transform, and $\odot$ is element-wise multiplication. To make the filters learnable, g is parameterized as a functiong_w, yielding:

$$x \star_G g_w = U(U^T g_w \odot U^T x)$$

For generalization, the spectral filter can be represented as a diagonal matrix in the eigenbasis of the Laplacian:

$$g_w{}' = \begin{pmatrix} g_{w_1(\lambda_1)} & & \\ & \cdots & \\ & & g_{w_n(\lambda_n)} \end{pmatrix}$$

Thus, the generalized spectral convolution can be formulated as:

$$x \star_G g'_w = U g'_w U^T x$$

This equation represents the most general and foundational formulation of spectral graph convolution. It was first introduced in its complete form by Shuman *et al.*[21]. The learnable spectral filter g'_w is commonly denoted as g_θ, where the subscript θ denotes the set of trainable parameters. By adopting different parameterizations or applying various constraints to g_θ, a broad spectrum of spectral GNN architectures can be derived, each with distinct inductive biases and computational properties. In the following sections, we highlight three of the most representative models that exemplify these design strategies.

SCNN: The Spectral Convolutional Neural Network (SCNN) formulates the filter g_θ as a diagonal matrix of size $n \times n$, where each diagonal entry is a learnable parameter:

$$g_\theta = \begin{pmatrix} \theta_1 & & \\ & \ddots & \\ & & \theta_n \end{pmatrix}$$

This model represents one of the earliest attempts at leveraging spectral graph convolution and is widely regarded as the pioneering work in this domain. However, SCNN also suffers from several inherent limitations. First, computing the eigendecomposition of the graph Laplacian L incurs a computational cost of $O(n^3)$, making it impractical for large-scale graphs. Second, the number of parameters in SCNN scales linearly with the number of nodes, i.e., $O(n)$, increasing the risk of overfitting on large graphs.

ChebNet: In SCNN, each entry of the filter matrix is learned independently, limiting the model's generalization across graphs of different sizes. ChebNet[22] addresses this challenge by expressing the spectral filter g_θ as a shared polynomial function over the graph spectrum. Among various polynomial bases, Chebyshev polynomials are particularly suitable due to their optimal approximation properties and numerical stability. Therefore, the spectral filter of ChebNet is approximated as:

$$g_\theta = \begin{pmatrix} \sum_{k=0}^{K} \beta_k T_k(\hat{\lambda}_1) & & \\ & \ddots & \\ & & \sum_{k=0}^{K} \beta_k T_k(\hat{\lambda}_2) \end{pmatrix}, \qquad \hat{\lambda} = \frac{2}{\lambda_{max}}\lambda - 1$$

The compact expression is:

$$g_\theta(\Lambda) = \sum_{k=0}^{K} \beta_k T_k(\hat{\Lambda}), \qquad \hat{\Lambda} = \frac{2}{\lambda_{max}}\Lambda - I_n$$

Thus, the graph convolution becomes:

$$x \star_G g_\theta = U g_\theta U^T x = \sum_{k=0}^{K} \beta_k T_k(\hat{L})\, x$$

where $\hat{L} = \frac{2}{\lambda_{max}} L - I_n$. This formulation eliminates the need to explicitly compute the eigenvectors U, making the model scalable to large graphs. Furthermore, the number of learnable parameters depends only on the polynomial order K, rather than the graph size, thereby alleviating overfitting risks. ChebNet also enforces strict spatial locality: the polynomial order K directly corresponds to the receptive field radius, ensuring that only nodes within K-hop neighborhoods are included in the convolution operation. This yields a localized filtering mechanism analogous to the fixed-size kernels used in conventional CNNs.

GCN: Building on ChebNet, Graph Convolutional Networks[23] (GCN) simplify the spectral filter even further by restricting the Chebyshev polynomial to the first two terms ($k = 0$ and $k = 1$):

$$g_\theta = \begin{pmatrix} \beta_0 T_0(\hat{\lambda}_1) + \beta_1 T_1(\hat{\lambda}_1) & & \\ & \ddots & \\ & & \beta_0 T_0(\hat{\lambda}_2) + \beta_1 T_1(\hat{\lambda}_2) \end{pmatrix},$$

Given that $T_0(\hat{L}) = 1$ and $T_1(\hat{L}) = \hat{L}$, the convolution reduces to:

$$x \star_G g_\theta = (\beta_0 + \beta_1 \hat{L})x$$

In the original GCN formulation, $\hat{L}$ is expressed by the normalized Laplacian L_{sym}, which is

$$L_{sym} = D^{-1/2} L D^{-1/2} = D^{-1/2}(D - W)D^{-1/2} = I_n - D^{-1/2} W D^{-1/2}$$

where A is the adjacency matrix and D the degree matrix. This normalization simplifies the spectral filter and ensures that the eigenvalues are bounded within a small interval, improving numerical stability and facilitating effective information propagation. To further streamline the model, the authors impose the constraint $\beta_0 = -\beta_1 = \theta$, yielding:

$$x \star_G g_\theta = (\theta(D^{-1/2} W D^{-1/2} + I_n))x$$

To mitigate gradient explosion in deep architectures, GCN introduces a renormalization trick by adding self-loops to the adjacency matrix, :

$$\tilde{W} = W + I_n, \qquad I_n + D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{W} \tilde{D}^{-\frac{1}{2}}$$

leading to the final form:

$$x \star_G g_\theta = \theta(\tilde{D}^{-1/2} \tilde{W} \tilde{D}^{-1/2})x$$

In this expression, $\tilde{D}^{-1/2} \tilde{W} \tilde{D}^{-1/2}$ depends solely on the graph topology and defines a fixed operator, while x represents node features and θ is the shared trainable weight. As in CNNs, this structure enforces a fixed, one-hop neighborhood receptive field and drastically reduces the number of parameters. Although GCN provides a clean and efficient bridge between spectral and spatial GNNs, some studies have noted that its simplicity may limit its expressiveness on more complex tasks[32].

In summary, SCNN, ChebNet, and GCN all fall within the spectral graph convolution framework, differing primarily in how they parameterize and constrain the spectral filter g_θ. Fundamentally, they follow a common procedure: node features x are projected into the spectral domain, filtered element-wise via g_θ, and then transformed back to the spatial domain to yield updated node representations. These models provide a principled and mathematically grounded approach to capturing structural patterns in graph-structured data.

2.1.2 Spatial-based GNN: Unified Framework with MPNN

In contrast to spectral graph convolution, which relies on graph spectral theory and Fourier transforms, **spatial graph convolution** operates directly on a node's local neighborhood. It aggregates messages from neighboring nodes to update the central node's representation. The spatial approach is intuitive, adaptable, and particularly well-suited to dynamic graph structures. Beyond numerous model variants, the spatial GNN literature has also given rise to several **unified frameworks** that abstract common mechanisms among diverse architectures. Among these, the **Message Passing Neural Network**[25] (MPNN) framework stands out as one of the most widely adopted and foundational paradigms. Below, we use MPNN as a lens through which to introduce the core ideas of spatial GNNs, drawing comparisons to spectral methods (e.g., GCN) and highlighting representative models such as GraphSAGE, GAT, and GIN.

The MPNN framework formalizes GNNs through four key functions, whose specific implementations determine the model's behavior. These are:

Message Function: A message represents a pairwise interaction between two nodes. For a node v with k neighbors, there are k messages generated from each neighbor $u \in \mathcal{N}(v)$. Formally, for edge $(u, v) \in E$, the message is computed as:

$$m_{uv}^{(t+1)} = \mathrm{Msg}(h_u^{(t)}, h_v^{(t)}, e_{uv})$$

where $h_u^{(t)}$ and $h_v^{(t)}$ are the features of source and target nodes at step t, and e_{uv} denotes edge features.

Aggregation Function: Aggregation summarizes all incoming messages for a node:

$$m_v^{(t+1)} = \mathrm{Agg}(\{m_{uv}^{(t+1)}: u \in \mathcal{N}(v)\})$$

Update Function: The node updates its representation using its current state and the aggregated message:

$$h_v^{(t+1)} = \mathrm{Update}\left(h_v^{(t)}, m_v^{(t+1)}\right)$$

Readout Function: To obtain graph-level representations (for tasks such as graph classification), the node features are pooled:

$$h_G = \mathrm{Read}(\{h_v^{(T)}: v \in V\})$$

For subgraph-level tasks, V can be replaced with a subset V_{sub}.

The MPNN framework captures the essence of spatial GNNs: direct message exchange and iterative neighborhood aggregation. It provides a flexible abstraction adopted by most modern GNN toolkits (e.g., PyG[29], DGL[30]), allowing researchers to focus on architectural innovation rather than implementation complexity.

GCN: Although GCN[23] was originally derived from the spectral theory, its receptive field—limited to one-hop neighbors—aligns it more closely with spatial GNNs in terms of operational behavior. Within the MPNN framework, GCN can be described as:

$$\textbf{Message: } m_{uv}^{(t+1)} = h_u^{(t)}$$

$$\textbf{Aggregate: } m_v^{(t+1)} = \sum\nolimits_{u \in \mathcal{N}(v)} \frac{1}{\sqrt{\hat{d}_{\mathrm{u}}\, \hat{d}_{\mathrm{v}}}} m_{uv}^{(t+1)} + h_v^{(t)}$$

$$\textbf{mUpdate: } h_v^{(t+1)} = \sigma(W^{(t)} m_v^{(t+1)})$$

Here, $\hat{d}_v$ denotes the degree of node v including the self-loop, $W^{(t)}$ is a learnable weight matrix, and σ is a nonlinear activiation function (e.g., ReLU). This can also be written more compactly as:

$$h_v^{(t+1)} = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{\hat{d}_{\mathrm{u}}\, \hat{d}_{\mathrm{v}}}} m_{uv}^{(t+1)} W^{(t)} h_u^{(t)}\right)$$

GCN's simplicity makes it a natural baseline for comparison with more expressive spatial GNNs.

GraphSAGE[26] (Sample and Aggregate) introduces two key innovations: (1) sampling a fixed-size neighborhood instead of using the entire graph, and (2) learning aggregation functions to combine neighbor features. Under MPNN, GraphSAGE is:

$$\textbf{Message:}\ m_{uv}^{(t+1)} = h_u^{(t)}$$

$$\textbf{Aggregation:}\ m_v^{(t+1)} = \mathrm{Agg}(m_{uv}^{(t+1)}: u \in \mathcal{N}(v))$$

$$\textbf{Update:}\ h_v^{(t+1)} = \sigma(W^{(t)}[h_v^t || m_v^{(t+1)}])$$

The aggregation function can be **mean**, **LSTM-based**, or **pooling (e.g., max)**. A critical contribution of GraphSAGE is the emphasis on **permutation invariance**: the output should not depend on the order of neighbors. For LSTM aggregators, this is enforced by randomly permuting the input sequence.

Graph Isomorphism Network (GIN)[27] was developed to explore the theoretical expressive power of GNNs. Drawing inspiration from the Weisfeiler-Lehman (WL) test, GIN improves upon GraphSAGE by resolving cases where mean or max aggregation cannot distinguish different neighborhood structures with identical summaries. In MPNN form:

$$\textbf{Message:}\ m_{uv}^{(t+1)} = h_u^{(t)}$$

$$\textbf{Aggregation:}\ m_v^{(t+1)} = \sum\nolimits_{(u,v)\in E} h_u^{(t)}$$

$$\textbf{Update:}\ h_v^{(t+1)} = \mathrm{MLP}((1 + \epsilon^{(t+1)})h_v(u) + m_v^{(t+1)})$$

where $\epsilon^{(t+1)}$ is a learnable scalar that balances the importance of a node's own features during update. Although simple in structure, GIN has been shown to match the WL test in its ability to distinguish graph structures, making it a strong candidate for tasks requiring high discriminative power.

Graph Attention Network (GAT)[28] integrates attention mechanisms into spatial aggregation, enabling the model to assign varying importance to each neighbor. The attention score between node u and node v is computed as:

$$e_{uv} = a^T \mathrm{LeakyReLU}(Wh_u + Wh_v)$$

$$\alpha_{uv} = \frac{\exp(e_{uv})}{\sum_{k\in\mathcal{N}(v)} \exp(e_{vk})}$$

where and α_{uv} is the attention weights after normalization. Under MPNN, GAT can be expressed:

$$\textbf{Message:}\ m_{uv}^{(t+1)} = \alpha_{uv}^{(t)} h_u^{(t)}$$

$$\textbf{Aggregation:}\ m_v^{(t+1)} = \sum\nolimits_{u\in\mathcal{N}(v)} m_{uv}^{(t+1)}$$

$$\textbf{Update:}\ h_v^{(t+1)} = \sigma(W^{(t)}[m_v^{(t+1)}])$$

Compared to GraphSAGE or GIN, GAT explicitly weighs each neighbor's contribution, offering a highly expressive, data-dependent aggregation strategy. Variants such as AttentiveFP[33] (which uses GRU-based readout) and EGAT[34] (which applies attention on edges) have further extended the utility of attention mechanisms in molecular property prediction and related tasks. Notably, GAT can be seen as a practical instantiation of models that achieve the expressiveness upper bound outlined by GIN.

2.2 Symmetric GNN: Incorporate Physical Laws

In traditional GNN applications, such as social networks, nodes typically possess limited attributes and relations. In contrast, molecular graphs in drug discovery and materials science are characterized by both their 2D topological structures and 3D geometric conformations. Specifically, each atom in a molecule is represented by a set of Cartesian coordinates in the 3D space. While these coordinates may vary under different reference frames, the molecular geometry, defined by the relative spatial arrangement of atoms, remains physically unchanged.

To make the model adhere to real-world physical constraints, it is essential to incorporate two types of symmetry principles into GNN design: **invariance** and **equivariance**[35]. **Invariance** applies to tasks where the output should remain unchanged under coordinate transformations. For example, when predicting scalar molecular properties such as thermodynamic stability or HOMO-LUMO energy gaps, the model's prediction must be independent of the molecule's orientation or position in space. **Equivariance**, on the other hand, is necessary for predicting vectorial or tensorial properties such as atomic forces or displacement directions. In these cases, if the input is rotated or translated, the output should transform in the same way. Conventional architectures like multilayer perceptrons (MLPs) do not inherently satisfy these symmetry constraints. To this end, it is necessary to embed appropriate physical inductive biases into the network architecture. Below, we review three representative approaches for incorporating geometric symmetries into GNNs within the broader message-passing framework.

Most symmetry-aware GNNs are formulated within the framework of **group theory**, which provides a rigorous mathematical formalism for defining invariance and equivariance. In the context of molecular systems, the most relevant symmetries lie in the **3D Euclidean space**, particularly:

- **SO(3)**: The group of all 3D rotation operations.
- **E(3)**: The Euclidean group, encompassing translations, rotations, and mirror reflections.
- **SE(3)**: The special Euclidean group, a subgroup of E(3), including only translations and rotations (excluding reflections).

These symmetry groups dictate how molecules can be transformed while preserving their physical identity. In the following section, we categorize invariant and equivariant GNNs into three classes, as summarized in **Table 2**.

Table 2. Overview of symmetric GNN models and their performance on several metrics. G is one of the scalar properties prediction tasks in QM9, and F is the force prediction on the Aspirin molecule in MD17, the reported results are mean absolute error (MAE).

Models	G meV	F (Aspirin) meV/Å	Keywords
SchNet[36]	14	58.5	RBF basis function for handling d
DimeNet[37]	8	21.6	Second-order MPNN considering θ
GemNet[38]	-	10.3	Third-order MPNN considering θ and τ
SphereNet[39]	18	-	Consider the ϕ between two planes rather than τ

ComENet[40]	7.98	-	Consider the d, θ, τ, ϕ
TFN[41]	-	-	Early CG-tensor product-based equivariant GNN
SE(3)-Transformer[42]	-	-	Introduce the dot product attention to the TFN
Cormorant[43]	20	-	Introduce non-linearity to TFN-like models
SEGNN[44]	15	-	Gated layer for equivariant non-linearity
EquiFormer[45]	7.63	6.6	Depth-wise tensor product, MLP-based attention
MACE[46]	-	6.6	Atomic Cluster Expansion, only 2-order interaction
GVP[47]	-	-	Isolate transformation and model multiplication
EGNN[48]	12	-	Atomic coord. difference multiplies invariant message
NewtonNet[49]	-	15.1	Extend EGNN to type, force, coords, and energy module
EQGAT[50]	23	-	Implement GVP equivariant mechanism in GAT
PaiNN[51]	20	14.7	Introduce Gated equivariant block to EGNN

2.2.1 Internal Coordinate-based Invariant GNN

While Cartesian coordinates are reference-frame dependent, certain internal features, such as bond lengths, bond angles, and dihedral angles, remain invariant across reference frames. These internal coordinates are foundational in classical molecular mechanics and quantum chemistry (**Figure 3A**), as they represent E(3)-invariant quantities critical for energy and force calculations[52]. Consequently, one intuitive strategy is to convert Cartesian inputs into internal coordinates before feeding them into a GNN. This principle is adopted by many state-of-the-art molecular representation models.

SchNet[36]: One of the earliest and most influential models in this line of work is SchNet, which introduces continuous filter convolutions to model interactions between atoms. The core design centers around interatomic distances, ensuring SE(3) invariance when predicting molecular properties such as total energy. To enhance the representational capacity of distance-based features, SchNet employs Radial Basis Functions (RBFs), which is also called **Gaussian Smearing**, to map scalar distances into high-dimensional vectors, mitigating the limitation of using single scalar values as filter inputs. This RBF-based technique has since been widely adopted in subsequent models. Within the MPNN framework, SchNet can be abstracted as follows:

$$\textbf{Message function:}\ m_{uv}^{(t+1)} = W^{(t)} f_{\text{filter}}(d_{uv}) \odot h_u^{(t)}$$

$$\textbf{Aggregation:}\ m_v^{(t+1)} = \sum_{u \in \mathcal{N}(v)} m_{uv}^{(t+1)}$$

$$\textbf{Update:}\ h_v^{(t+1)} = \sigma\left(W^{(t)} h_v^{(t)} + m_v^{(t+1)} + b^{(t)}\right)$$

Here, the filter function is defined as:

$$f_{\text{filter}}(d_{uv}) = \exp\left(-\gamma(\| r_u - r_v \| - \mu)^2\right)$$

where γ controls the width and μ the center of the Gaussian filter. Through this distance-based representation, SchNet ensures rotational and translational invariance for scalar predictions such as total energy E. The model also supports equivariant force prediction via the energy gradient:

$$E = \sum_{u \in V} h_u$$

$$F_i(Z_1, \dots, Z_n, \boldsymbol{r}_1, \dots, \boldsymbol{r}_n) = -\frac{\partial E}{\partial \boldsymbol{r}_i}(h_1, \dots, h_n, \boldsymbol{r}_1, \dots, \boldsymbol{r}_n)$$

Because h_u is SE(3)-invariant, the energy E is also SE(3)-invariant, and the force prediction F_i transforms equivariantly under rotations:

$$F(Rr) = -\nabla E(Rr) = -R\nabla E(r) = RF(r)$$

This ensures that F produces physically consistent outputs under coordinate transformations, i.e., SE(3)-equivariant predictions.

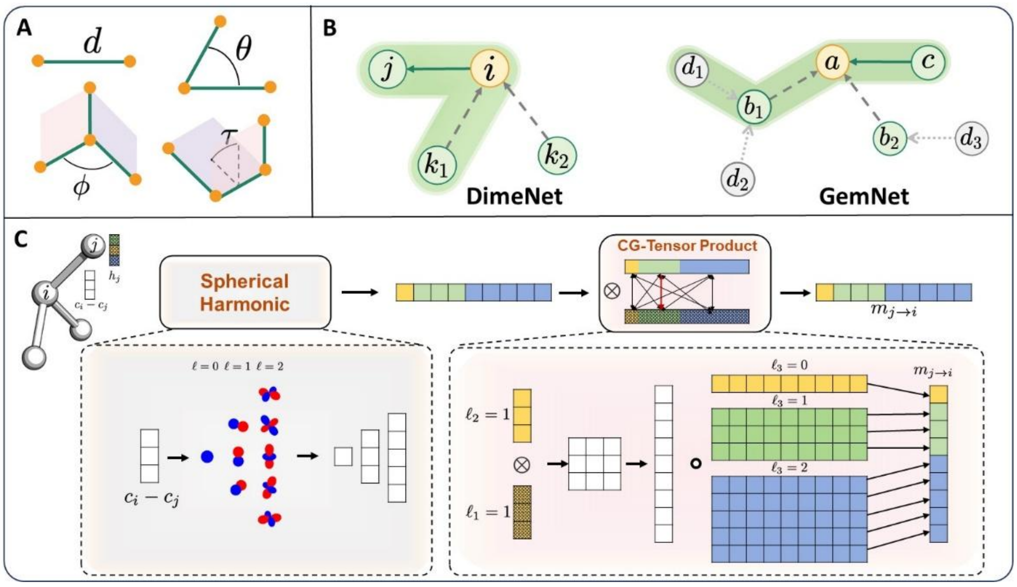

Figure 3. A) Internal coordinate system; B) The illustration of the DimeNet and GemNet message passing frameworks, which represent the second- and third-order interactions, respectively. C) The illustration of the CG-Tensfor Product method, where the CG-Tensor product operates between input geometric features and the spherical harmonics of relative position vectors.

DimeNet[37]**:** While SchNet captures pairwise interactions based solely on distance, it cannot distinguish molecular conformers with identical pairwise distances but different angular arrangements. To address this, **DimeNet** introduces directional message passing by incorporating angle-based features into the convolution process, enabling the model to capture richer geometric information, as illustrated in **Figure 3B**. Conceptually, its message function is defined as:

$$m_{ji}^{(t+1)} = f_{\text{update}}(m_{ji}^{(t)}, \sum_{k \in N(j) \setminus \{i\}} f_{\text{int}}(m_{kj}^{(t)}, e_{\text{rbf}}(d_{ji}), e_{\text{cbf}}(d_{ji}, \theta_{kji}))))$$

Here, $e_{\text{rbf}}(d_{ji})$ and $e_{\text{cbf}}(d_{ji}, \theta_{kji})$ denote radial and spherical basis embeddings of distances and angles, respectively. These embeddings are constructed using a combination of Bessel functions j_l, spherical harmonics $Y_l^0(\theta)$ and an envelope function $u(d)$ to map geometric inputs into high-dimensional latent spaces.

From the MPNN perspective, DimeNet captures second-order interactions: each message along edge $j \rightarrow i$ not only encodes the pairwise distance between j and i, but also aggregates geometric context from a third atom k, forming angle θ_{kji}. This higher-order aggregation improves the model's expressivity and aligns with a second-order Weisfeiler-Lehman test. Empirically, DimeNet outperforms SchNet by approximately 1.5× on benchmark datasets such as MD17 (aspirin data), which would be discussed later in the molecular property prediction section.

GemNet: While DimeNet accounts for angular information, a complete reconstruction of 3D molecular conformations also requires dihedral angles. To this end, the authors of DimeNet proposed **GemNet**[38], which extends directional message passing to include third-order geometric interactions via additional aggregation layers, as illustrated in **Figure 3B**. Here, we provide the conceptual message function in GemNet for better understanding:

$$m_{ca}^{(t+1)} = f_{\text{update}}(m_{ca}^{(t)}, \sum_{d \in N(b) \setminus \{a,c\}} \sum_{b \in N(a) \setminus \{c\}} f_{\text{int}}(m_{db}^{(t)}, e_{\text{rbf}}(d_{db}), e_{\text{cbf}}(d_{ba}, \theta_{abd}), e_{\text{sbf}}(d_{ca}, \theta_{cab}, \phi_{cabd}))))$$

This formulation integrates a four-body interaction involving nodes $c \rightarrow a \leftarrow b \leftarrow d$, allowing GemNet to capture dihedral dependencies ϕ_{cabd}, alongside distances d_{db}, d_{ba}, d_{ca} and angles $\theta_{cab}, \theta_{abd}$, which aligns a third-order Weisfeiler-Lehman test. While GemNet improves expressivity, achieving approximately 2× performance improvement over DimeNet on datasets like MD17, it comes at a higher computational cost due to quadruplet enumeration, limiting its scalability for larger molecules.

Other Models: Beyond the aforementioned models, several other approaches pursue similar goals of capturing **multi-body geometric interactions**. For instance, **SphereNet**[39] models the angle between planes (dihedral-like) via a variable ϕ. **ComENet** [6] combines multiple geometric terms, including distances, angles, and dihedrals, into a unified message formulation. These models aim to enhance expressivity by modeling higher-order interactions $f(h_1, h_2, \dots, h_k)$ within k-hop neighbors. While SchNet, DimeNet, and GemNet capture first-, second-, and third-order interactions, respectively, SphereNet and ComENet explore alternative three-body definitions based on plane-plane angular system.

In summary, these invariant neural architectures leverage SE(3)-invariant features internally, from which they extract physically meaningful embeddings. Under the MPNN framework, these features are propagated and updated across graph layers, enabling the model to predict not only scalar properties (e.g., energy E) but also equivariant outputs (e.g., atomic forces $F(r) = -\nabla E(r)$). However, it is important to note that in such models, equivariant outputs are derived from gradients of invariant scalars, which inherently limits their flexibility. For tasks requiring directly learnable equivariant outputs, more general equivariant neural architectures are required.

Additionally, spherical harmonics play a dual role in these models: they embed angular information in a physically grounded way and also serve as angular basis functions. In the next section on tensor-product-based equivariant GNNs, we will see how these harmonics construct higher-order (type-l) equivariant features, enabling more expressive and generalizable models.

2.2.2 Tensor Product-based Equivariant GNN

An alternative strategy for achieving equivariance in GNN is to leverage group-invariant convolutional theory to operate directly on geometric features. While the associated mathematical tools are more complex, models based on these principles (e.g., SE(3)-Transformer) have been widely adopted in large biomolecular structure prediction tasks, such as RosettaFold[53] and AlphaFold[54], underscoring their strong research and application potential. The fundamental idea is to 1) extend common SO(3) equivariance to higher-order ones; 2) enforce operations equivariant via Glebsch-Gorden-tensor products (CG-tensor product), and 3) parameterize interactions across multiple representation orders using neural networks. The basic idea is illustrated in **Figure 3C**. Readers interested in the mathematical background may refer to **SI Part 1**.

The tensor product operation itself is inherently equivariant: when input features transform under rotations, the resulting tensor product preserves the same transformation behavior. According to the group theory discussed in the **SI Part 1**, scalar distances can be viewed as order $l = 0$ features, while coordinate vectors correspond to order $l = 1$. Most models begin with such inputs and enrich the geometric information by computing spherical harmonic embeddings Y^l of relative coordinates r_{ij}, yielding higher-order equivariant type-l features. These features are then passed through learnable CG-tensor products to generate output features across orders $\{0, \ldots, m\}$. For scalar prediction such as energy or charge, the $l = 0$ output is used; for vector predictions such as force, the $l = 1$ output is used. On this foundation, we would discuss two representative models in detail: Tensor Field Networks (TFN) and SE(3)-Transformer.

Tensor Field Networks[41] (TFN) is among the first to apply group-invariant convolution theory to GNNs. The concept builds upon early developments in rotation-equivariant convolution, such as Harmonic Networks[55] and Spherical CNNs[56], and formalizes equivariant message passing through the use of CG tensor products. The message function is defined as:

$$m_{ij}^l = [h_i \otimes_{cg}^{\psi(d_{ij})} Y(e_{ij})]^l$$

where h_i^l is the type-l representation of node i, d_{ij} is the interatomic distance, and e_{ij} is the unit vector between nodes. The filter is constructed by modulating the spherical harmonic Y with a learnable radial embedding ψ and applying a CG product. The superscript $[\cdot]^l$ selects the type-l component from the output. To be more specific, the message is:

$$m_{ij,m}^{(l)} = \psi_m(d_{ij}) \sum_{m_1=-l_1}^{l_1} \sum_{m_2=-l_2}^{l_2} C_{(l_1,m_1)(l_2,m_2)}^{(l,m)} h_{i,m_1}^{(l_1)} Y_{m_2}^{(l_2)}(e_{ij})$$

where $C^{(l,m)}_{(l_1,m_1)(l_2,m_2)}$ determine how the input features are coupled into output components of type-l. By construction, this operation ensures that the output message m^l_{ij} remains equivariant. Aggregation and update functions are defined as:

$$\Delta h^l_i = \sum_{j \in \mathcal{N}(i)} m^l_{ij}$$

$$h^{l,(t+1)}_i = w^{l(t)} h^{l(t)}_i + \Delta h^{l,(t)}_i$$

where $w^{l(t)}$ is a learnable parameter, and t is the layer index. The model output is split into the scalar ($l = 0$) and higher-order ($l > 0$) parts:

$$h^0_{\mathrm{i,pred}} = W_{\mathrm{pred}} h^0_i$$

$$h^l_{\mathrm{i,pred}} = \mathrm{MLP}\left(h^l_i\right), \qquad l > 0$$

Some works use a more general formulation of TFN's equivariant kernel:

$$h^l_{\mathrm{out},i} = \sum_{k \geq 0} \sum_{j \in \mathcal{N}(i)}^{n} W^{lk}\,(x_j - x_i) h^k_{\mathrm{in},j}$$

$$W^{lk}\left(\vec{r}_j - \vec{r}_i\right) = \sum_{J=|k-l|}^{k+l} \psi^{lk}_J\left(\| d_{ij} \|\right) W^{lk}_J\left(\frac{x_j - \mathrm{x_i}}{\| d_{ij} \|}\right), \qquad W^{lk}_J = \sum_{m=-J}^{J} Y^{(J)}_m \left(\frac{x_j - \mathrm{x_i}}{\| d_{ij} \|}\right) C^{lk}_{Jm}$$

where C^{lk}_{Jm} are the Clebsch–Gordan coefficients, and the spherical harmonics ensure angular dependency. This construction allows the kernel to transform type-k inputs into type-l outputs while preserving equivariance.

SE(3)-Transformer: Building upon TFN, the **SE(3)-Transformer** introduces a self-attention mechanism while maintaining the equivariance. Its message function is written as:

$$m^l_{ij} = \alpha_{ij} [h_i \otimes^{\psi(d_{ij})}_{cg} Y\left(e_{ij}\right)]^l$$

where α_{ij} is an attention coefficient that modulates the influence of neighbor j on node i. To preserve equivariance, α_{ij} must be invariant under rotation. This is achieved by defining α_{ij} via invariant queries and keys:

$$\mathrm{q}_i = \bigoplus_{\ell \geq 0} \sum_{k \geq 0} \mathrm{W}^{\ell k}_Q \mathrm{h}^k_{\mathrm{in},i}, \qquad \mathrm{k}_{ij} = \bigoplus_{\ell \geq 0} \sum_{k \geq 0} \mathrm{W}^{\ell k}_K (\mathrm{x}_j - \mathrm{x}_i) \mathrm{h}^k_{\mathrm{in},j}$$

$$\alpha_{ij} = \frac{\exp(\mathrm{q}_i^{\top} \mathrm{k}_{ij})}{\sum_{j' \in \mathcal{N}_i \setminus i} \exp\,(\mathrm{q}_i^{\top} \mathrm{k}_{ij'})},$$

The attention-enhanced message is then aggregated and updated similarly to TFN, with an attention-based update coefficient:

$$\Delta h^l_i = \sum_{j \in \mathcal{N}(i)} m^l_{ij}$$

$$h^{l,(t+1)}_i = w^{l(t)}_{\mathrm{attn}} h^{l(t)}_i + \Delta h^{l,(t)}_i$$

where $w_{\mathrm{attn}}^{l(t)} = \mathrm{MLP}(h_i^{l(t)} \cdot h_i^{l(t)})$ is a non-linear self-attention weight. Beyond TFN and SE(3)-Transformer, many related models refine CG-tensor product-based equivariant graph convolution. These architectures focus on improving either expressiveness or computational efficiency. **Cormorant**[43] introduces nonlinear interactions across tensors between different orders; **SEGNN**[44] introduces gating mechanisms for equivariant nonlinearity, where scalars are passed through standard activations and higher-order tensors are modulated via learned scalar gates:

$$(\bigoplus_i \phi_i(x_i)) \oplus (\bigoplus_j \phi_j(g_j) y_j),$$

where x_i are input scalar features, y_j are input type-l $(l > 0)$ tensors, and g_j are gating scalars. This gating mechanism has become standard in subsequent works. **EquiFormer**[45] builds on SE(3)-Transformer by replacing dot-product attention with an MLP attention scheme and substituting full tensor products with depth-wise tensor products to reduce computational load while expanding model capacity. **MACE**[46] introduces Atomic Cluster Expansion (ACE) into TFN-like architectures. By using only pairwise convolutions of the form $h_i \otimes_{cg}^{\psi(d_{ij})} Y(e_{ij})$, MACE approximates many-body interactions through a complete basis, offering strong expressiveness with minimal cost.

2.2.3 Vector-based Equivariant GNN

Previous sections discuss invariant GNNs based on internal coordinates and equivariant GNNs leveraging CG tensor products. This section introduces a third paradigm: **vector graph neural networks**. In many practical tasks, the relevant physical quantities correspond to low-order rotational features, typically with angular momentum $l \leq 1$. Examples include atomic charges or molecular solubility ($l = 0$), and atomic velocities or force vectors ($l = 1$). While CG tensor products can represent higher-order equivariant features, such models often incur significant computational overhead. To improve efficiency, researchers have developed alternative equivariant operations tailored to scalar and vector features, avoiding the full complexity of CG algebra. These approaches, collectively referred to as vector GNNs, are introduced below through representative models.

Geometric Vector Perceptron (GVP)[47] employs an elegant equivariant design for updating scalar–vector pairs. The core formulation is as follows:

$$s', \vec{v}' = \mathrm{GVP}\left(s, \vec{v}\right)$$

$$s' = \sigma\left(W_m \begin{bmatrix} \| W_h \vec{v} \|_2 \\ s \end{bmatrix} + b\right)$$

$$\vec{v}' = \sigma\left(\left\| W_\mu W_h \vec{v} \right\|\right) \odot W_\mu W_h \vec{v}$$

Here, $\odot$ denotes element-wise multiplication. The vector feature $\vec{v} \in \mathbb{R}^{C \times 3}$ is designed such that linear transformations are applied from the left, and rotation matrices act from the right, ensuring equivariance under SO(3) rotations. While many implementations follow the PyTorch convention of applying linear layers to the final dimension (i.e., right-multiplication), the original GVP formulation adheres to the left-multiplication tradition of linear algebra. Readers may refer to the **SI Part 2** for a formal proof of GVP's equivariance. A GVP-based vector GNN typically follows this structure:

$$m_{ij}^{t+1}, \vec{m}_{ij}^{t+1} = \mathrm{GVP}(s_i^t \| s_i^t \| e_{ij}, \vec{v}_i^t \| \vec{v}_j^t \| \vec{e}_{ij})$$

$$m_i^{t+1}, \vec{m}_i^{t+1} = \frac{1}{M} \sum_{j \in \mathcal{N}(i)} (m_{ij}^{t+1}, \vec{m}_{ij}^{t+1})$$

$$h_i^{t+1}, \vec{h}_j^{t+1} = \left(h_i^t, h_j^t\right) + \mathrm{GVP}(m_i^{t+1}, \vec{m}_i^{t+1})$$

where $s_i, \vec{v}_i$ denote scalar and vector node features, and $e_i, \vec{e}_i$ represent edge features. A similar principle was adopted by Deng et al.[57] in the Vector Neurons framework, applied to point cloud data. Their model, VN-PointNet, demonstrated significant performance gains when SO(3) equivariance was enforced.

Equivariant Graph Neural Network (EGNN): Subsequent studies, such as Jing et al.[47], showed that GVP performs well for low-order geometric features. Victor et al. [48] later proposed **EGNN**, a model that became central to geometry-aware GNN research. Unlike GVP, which focuses on SO(3) equivariance, EGNN introduces a broader E(n)-equivariant formulation by using coordinate differences during message passing.

The model separates scalar and vector messages:

$$m_{ij}^t = \phi_e\left(h_i^t, h_j^t, d_{ij}, e_{ij}\right), \qquad \vec{m}_{ij}^t = \left(x_i^t - x_j^t\right)\phi_x\left(m_{ij}^t\right)$$

where ϕ_e, ϕ_x are neural message functions, often implemented as MLPs. The design ensures that scalar messages remain invariant under transformations, while vector messages transform equivariantly. This can be formally verified as follows:

$$\begin{aligned} \vec{m}_{ij}^{t'} &= \left(Rx_i^t + g - R_{x_j}^t - g\right)\phi_x\left(m_{ij}^t\right) \\ &= R\left(x_i^t - x_j^t\right)\phi_x\left(m_{ij}^t\right) \\ &= R\,\vec{m}_{ij}^t \end{aligned}$$

Thus, equivariance is preserved under translation and rotation. The aggregation and update steps are defined as:

$$m_i^t = \sum_{j \in \mathcal{N}(i)} m_{ij}^t, \qquad \vec{m}_i^t = \frac{1}{M} \sum_{j \in \mathcal{N}(i)} \vec{m}_{ij}^t$$

$$h_i^{t+1} = \phi_h(h_i^t, m_i^t), \qquad \mathbf{x}_i^{t+1} = \mathbf{x}_i^l + \vec{m}_i^t$$

EGNN directly updates node coordinates, making it well-suited for structural prediction and dynamic simulations. For instance, **KarmaDock**[58] uses this mechanism to iteratively guide ligand fitting into protein pockets via force field-inspired optimization.