GPS++: An Optimised Hybrid MPNN/Transformer for Molecular Property Prediction

Submission to the 2022 Open Graph Benchmark - Large Scale Challenge

Dominic Masters
Graphcore, UK
dominicm@graphcore.ai

Josef Dean
Graphcore, London, UK
josefd@graphcore.ai

Kerstin Klaser
Graphcore, London, UK
kerstink@graphcore.ai

Zhiyi Li
Graphcore, Cambridge, UK
zhiyil@graphcore.ai

Sam Maddrell-Mander
Graphcore, Bristol, UK
samuelm@graphcore.ai

Adam Sanders
Graphcore, Bristol, UK
adams@graphcore.ai

Hatem Helal
Graphcore, Cambridge, UK
hatemh@graphcore.ai

Deniz Beker
Graphcore, Bristol, UK
denizb@graphcore.ai

Ladislav Rampášek
Mila, Université de Montréal
ladislav.rampasek@mila.quebec

Dominique Beaini
Valence Discovery, Mila, Université de Montréal
dominique@valencediscovery.com

Abstract

This technical report presents GPS++, a top-3 finalist method in Open Graph Benchmark Large-Scale Challenge (OGB-LSC 2022) for PCQM4Mv2 molecular property prediction task. Our approach implements several key principles from the prior literature. At its core our GPS++ method is a hybrid MPNN/Transformer model that incorporates 3D atom positions and an auxiliary denoising task. The effectiveness of GPS++ is demonstrated by achieving 0.0719 mean absolute error on the independent `test-challenge` PCQM4Mv2 split. Thanks to Graphcore IPU acceleration, GPS++ scales to deep architectures (16 layers), training at 3 minutes per epoch, and large ensemble (112 models), completing the final predictions in 1 hour 32 minutes, well under the 4 hour inference budget allocated. Our implementation is publicly available at: https://github.com/graphcore/ogb-lsc-pcqm4mv2.

Keywords Molecular property prediction, Graph learning, Hybrid MPNN/Transformer, OGB-LSC PCQM4Mv2

1 Introduction

In a push to accelerate development of machine learning on graph-structured data, Open Graph Benchmark (OGB) [Hu et al., 2020] was created with a variety of graph learning tasks in mind, ranging from graph-level prediction, to link-level prediction, to node-level prediction tasks. Each of these categories has its own challenges, particularly when scaling the application to considerably larger sets of graphs or to graphs with a considerably larger number of nodes. The application at hand and the desired scaling direction thus majorly impacts the machine learning method development. To encourage development of methods for highly impactful applications, OGB Large Scale Challenge (LSC) [Hu et al., 2021] was organised and for the first time held at KDD 2021. In this technical report we present our `GPS++` submission to OGB-LSC 2022, the second installment of the challenge, for the graph-level prediction task PCQM4Mv2.

The PCQM4Mv2 dataset [Hu et al., 2021] is specifically aimed at aiding the development of machine learning methods for molecular property prediction. The task presented is to predict the HOMO-LUMO energy gap of a molecule, a property that is typically calculated using Density Functional Theory (DFT) [Kohn and Sham, 1965]. DFT is the de facto method used for accurately predicting quantum phenomena across a range of molecular systems. Unfortunately, traditional DFT can be extremely computationally expensive, prohibiting the efficient exploration of chemical space [Dobson, 2004]. Within this context the motivation for replacing it with fast and accurate machine learning models is clear. While this task does aim to accelerate the development of new methods for DFT it also serves as a proxy for other

molecular property prediction tasks. It therefore has the potential to benefit a wide range of scientific applications in fields like computational chemistry, material sciences and drug discovery.

In this work we present `GPS++`, a hybrid message passing neural network (MPNN) and transformer that builds on the General, Powerful, Scalable (GPS) framework presented by Rampášek et al. [2022]. Specifically, we combine a large and expressive message passing module with a biased self-attention layer to maximise the benefit of local inductive biases while still allowing for effective global communication. Furthermore, we integrate a grouped input masking method to exploit available 3D positional information and use a denoising loss to alleviate oversmoothing.

We accelerate the training and inference of `GPS++` model with Graphcore IPUs allowing us to train our final 44M parameter model in under 24 hours, just 3 minutes per epoch. This final model achieves comparative single-model mean absolute error (MAE) to the state of the art transformers with only 60% of the parameters.

Our final competition submission consists of a 112 model ensemble, which due to hardware acceleration completes inference in 1 hour and 32 minutes, well under the 4 hour budget allowed. We achieve a final MAE of 0.0719 on the `test-challenge` data split achieving a top-3 position in the competition.

2 Preliminaries

2.1 Notation

Throughout the paper we use the following notation. Bold lowercase letters $\mathbf{v}$ are (row) vectors, bold uppercase letters $\mathbf{M}$ are matrices, with individual elements denoted by non-bold letters i.e. v_k or M_{pq}. Blackboard bold lowercase letters $\mathbb{v}$ are categorical (integer-valued) vectors. In general, we denote by $\left[\mathbf{v}_k\right]_{k\in K}$ the vertical concatenation (stacking) of vectors $\mathbf{v}_k$. Vertical concatenation is also denoted by a semicolon, i.e. $\left[\mathbf{v}_1\ ;\ ...\ ;\ \mathbf{v}_J\right] = \left[\mathbf{v}_j\right]_{j=1}^{J} = \left[\mathbf{v}_j \text{ for } j \in \{1..J\}\right]$. Horizontal concatenation, which typically means concatenation along the feature dimension, is denoted by a vertical bar, i.e. $\left[\mathbf{v}_1\ \mid\ \mathbf{v}_2\right]$.

2.2 Dataset

The PCQM4Mv2 dataset [Hu et al., 2021] consists of 3.7M molecules defined by their SMILES strings. Each molecule can be represented as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ for nodes $\mathcal{V}$ and edges $\mathcal{E}$. In this representation, each node $i \in \mathcal{V}$ is an atom in the molecule and each edge $(u, v) \in \mathcal{E}$ is a chemical bond between two atoms. The number of atoms in the molecule is denoted by $N = |\mathcal{V}|$ and the number of edges is $M = |\mathcal{E}|$. Each molecule has on average 14 atoms and 15 chemical bonds. However, as the bonds are undirected in nature and graph neural networks act on directed edges, two bidirectional edges are used to represent each chemical bond.

The 3.7M molecules are separated into standardised sets by OGB Hu et al. [2021], namely into `training` (90%), `validation` (2%), `test-dev` (4%) and `test-challenge` (4%) sets using a scaffold split where the HOMO-LUMO gap targets are only publicly available for the `training` and `validation` splits.

Each node and edge is also associated with a list of categorical features $\mathbb{x}_i \in \mathbb{Z}^{D_{\text{atom}}}$ and $\mathbb{e}_{uv} \in \mathbb{Z}^{D_{\text{bond}}}$, respectively, for D_{atom} atom features and D_{bond} bond features. A further set of 3D atom positions $\mathbf{R} = \left[\mathbf{r}_1\ ;\ ...\ ;\ \mathbf{r}_N\right] \in \mathbb{R}^{N\times 3}$, extracted from the original DFT simulations, is also provided for the training data, but crucially not for the validation and test data.

Our algorithm operates on edge, node, and global *features*. Node features in layer ℓ are denoted by $\mathbf{x}_i^\ell \in \mathbb{R}^{d_{\text{node}}}$, and are concatenated into the $N \times d_{\text{node}}$ matrix $\mathbf{X}^\ell = \left[\mathbf{x}_1^\ell\ ;\ ...\ ;\ \mathbf{x}_N^\ell\right]$. Edge features $\mathbf{e}_{uv}^\ell \in \mathbb{R}^{d_{\text{edge}}}$ are concatenated into the edge feature matrix $\mathbf{E}^\ell = \left[\mathbf{e}_{uv}^\ell \text{ for } (u, v) \in \mathcal{E}\right]$. Global features are defined per layer as $\mathbf{g}^\ell \in \mathbb{R}^{d_{\text{global}}}$.

In this work, we set $d_{\text{node}} = 256$, $d_{\text{edge}} = 128$ and $d_{\text{global}} = 64$.

We also define an *attention bias* matrix $\mathbf{B} \in \mathbb{R}^{N\times N}$, computed from the input graph topology and 3D atom positions, described later in Section 3.2.

3 GPS++

Our `GPS++` model closely follows the GPS framework set out in Rampášek et al. [2022]. This work presents a flexible model structure for building hybrid MPNN/Transformer models for graph-structured input data. We build a specific implementation of GPS that focuses on maximising the benefit of the inductive biases of the graph structure and 3D positional information. We do this by building a large and expressive MPNN component and biasing our attention

component with structural and positional information. We also allow global information to be propagated through two mechanisms, the global attention and by using a global feature in the MPNN.

The main GPS++ block (Section 3.1) combines the benefits of both message passing and attention layers by running them in parallel before combining them with a simple summation and MLP; this is repeated 16 times. This main trunk of processing is also preceded by a an Encoder function (Section 3.2) responsible for encoding the input information into the latent space and followed by a simple Decoder function (Section 4.2).

Feature engineering is also used to improve the representation of the atoms/bonds, to provide the rich positional and structural features that increase expressivity, and to bias the attention weights with a distance embedding.

3.1 GPS++ Block

The GPS++ block is defined as follows for layers $\ell > 0$ (see Section 3.2 for the definitions of $\mathbf{X}^0, \mathbf{E}^0, \mathbf{g}^0$).

$$\mathbf{X}^{\ell+1}, \mathbf{E}^{\ell+1}, \mathbf{g}^{\ell+1} = \texttt{GPS++}\left(\mathbf{X}^{\ell}, \mathbf{E}^{\ell}, \mathbf{g}^{\ell}, \mathbf{B}\right) \tag{1}$$

$$\text{computed as} \quad \mathbf{Y}^{\ell},\ \mathbf{E}^{\ell+1},\ \mathbf{g}^{\ell+1} = \texttt{MPNN}\left(\mathbf{X}^{\ell}, \mathbf{E}^{\ell}, \mathbf{g}^{\ell}\right), \tag{2}$$

$$\mathbf{Z}^{\ell} = \texttt{BiasedAttn}\left(\mathbf{X}^{\ell}, \mathbf{B}\right), \tag{3}$$

$$\forall i: \quad \mathbf{x}_i^{\ell+1} = \texttt{FFN}\left(\mathbf{y}_i^{\ell} + \mathbf{z}_i^{\ell}\right) \tag{4}$$

The MPNN module Our MPNN module is a variation on the neural message passing module with edge and global features [Gilmer et al., 2017, Battaglia et al., 2018, Bronstein et al., 2021]. We choose this form to maximise the expressivity of the model with the expectation that over-fitting will be less of an issue with PCQM4Mv2, compared to other molecular datasets, due to its size. This MPNN module is defined as follows (see Figure 1 for a graphical representation):

$$\mathbf{Y}^{\ell},\ \mathbf{E}^{\ell+1},\ \mathbf{g}^{\ell+1} = \texttt{MPNN}\left(\mathbf{X}^{\ell}, \mathbf{E}^{\ell}, \mathbf{g}^{\ell}\right), \tag{5}$$

$$\text{computed as} \quad \forall (u,v): \quad \bar{\mathbf{e}}_{uv}^{\ell} = \texttt{Dropout}_{0.0035}\left(\texttt{MLP}_{\text{edge}}\left(\left[\mathbf{x}_u^{\ell} \mid \mathbf{x}_v^{\ell} \mid \mathbf{e}_{uv}^{\ell} \mid \mathbf{g}^{\ell}\right]\right)\right) \tag{6}$$

$$\forall i: \quad \bar{\mathbf{x}}_i^{\ell} = \texttt{MLP}_{\text{node}}\left(\left[\mathbf{x}_i^{\ell} \;\middle|\; \sum_{(u,i)\in\mathcal{E}} \left[\bar{\mathbf{e}}_{ui}^{\ell} \mid \mathbf{x}_u^{\ell}\right] \;\middle|\; \sum_{(i,v)\in\mathcal{E}} \left[\bar{\mathbf{e}}_{iv}^{\ell} \mid \mathbf{x}_v^{\ell}\right] \;\middle|\; \mathbf{g}^{\ell}\right]\right) \tag{7}$$

$$\bar{\mathbf{g}}^{\ell} = \texttt{MLP}_{\text{global}}\left(\left[\mathbf{g}^{\ell} \;\middle|\; \sum_{j\in\mathcal{V}} \bar{\mathbf{x}}_j^{\ell} \;\middle|\; \sum_{(u,v)\in\mathcal{E}} \bar{\mathbf{e}}_{uv}^{\ell}\right]\right) \tag{8}$$

$$\forall i: \quad \mathbf{y}_i^{\ell} = \texttt{LayerNorm}(\texttt{Dropout}_{0.3}(\bar{\mathbf{x}}_i^{\ell})) + \mathbf{x}_i^{\ell} \tag{9}$$

$$\forall (u,v): \quad \mathbf{e}_{uv}^{\ell+1} = \bar{\mathbf{e}}_{uv}^{\ell} + \mathbf{e}_{uv}^{\ell} \tag{10}$$

$$\mathbf{g}^{\ell+1} = \texttt{Dropout}_{0.35}(\bar{\mathbf{g}}^{\ell}) + \mathbf{g}^{\ell}, \tag{11}$$

where $\texttt{Dropout}_p$ [Srivastava et al., 2014] masks by zero each element with probability p and LayerNorm follows the normalisation procedure by Ba et al. [2016]. The three networks $\texttt{MLP}_\eta$ for $\eta \in \{\text{node}, \text{edge}, \text{global}\}$ each have two layers and are defined by:

$$\mathbf{y} = \texttt{MLP}_{\eta}(\mathbf{x}) \tag{12}$$

$$\text{computed as} \quad \bar{\mathbf{x}} = \texttt{GELU}(\texttt{Dense}(\mathbf{x})) \in \mathbb{R}^{4d_\eta} \tag{13}$$

$$\mathbf{y} = \texttt{Dense}(\texttt{LayerNorm}(\bar{\mathbf{x}})) \in \mathbb{R}^{d_\eta} \tag{14}$$

where GELU is an activation function defined in Hendrycks and Gimpel [2016].

This message passing block is principally the most similar to Battaglia et al. [2018]. Our variation is predominantly two fold: i) we concatenate the node representations to the incident edge in the formation of "messages", and ii) we concatenate inputs to the MLP rather than sum.

The BiasedAttn module Our BiasedAttn module follows the form of a biased self attention by Ying et al. [2021a] where a standard self attention block [Vaswani et al., 2017] is biased by a structural prior derived from the input graph. In our work the bias $\mathbf{B}$ is made up of two components, a shortest path distance embedding and a 3D distance bias

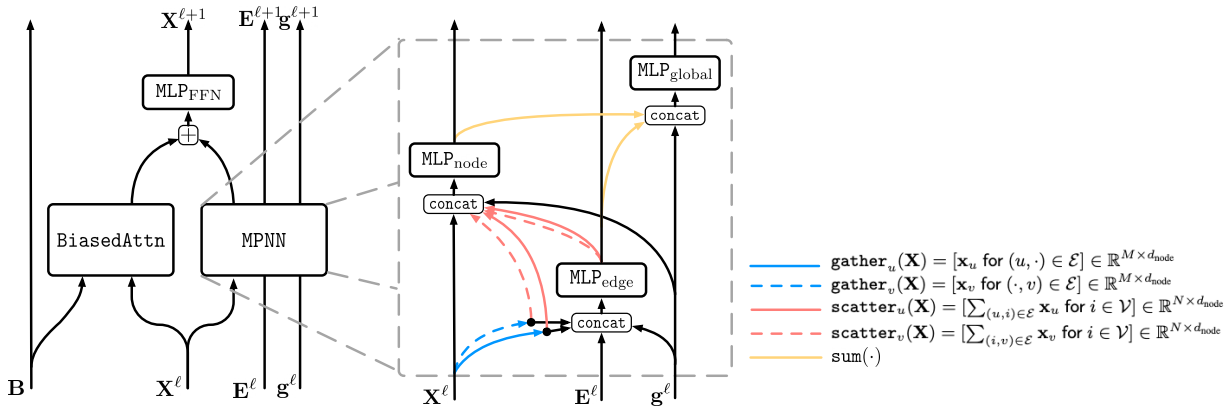

Figure 1: The main GPS++ processing block (left) is composed of a local message passing MPNN module (right) and a biased global attention BiasedAttn module. The right part of the figure shows a diagram of the used MPNN block. The gather, scatter and sum operations highlight changes in tensor shapes and are defined as above.

Table 1: Allocation of the number of parameters (in millions) throughout the model.

Location	**# Parameters (M)**	
Encoder	0.63	1.4%
GPS++ (16 blocks)	43.7	98.6%
MPNN	31.0	70.0%
MLP_{node}	22.1	50.0%
MLP_{edge}	6.84	15.4%
$\text{MLP}_{\text{global}}$	2.11	4.8%
BiasedAttn	4.22	9.5%
MLP_{FFN}	8.42	19.0%
Decoder	0.12	0.3%
Total	**44.3**	**100%**

derived from the molecular conformations as described in Section 3.2. Single-head attention is defined as:

$$\texttt{BiasedAttn}(\mathbf{X}, \mathbf{B}) = \texttt{Softmax}\left(\frac{(\mathbf{X}\mathbf{W}_Q)(\mathbf{X}\mathbf{W}_K)^\top}{\sqrt{d_{\text{node}}}} + \mathbf{B}\right)(\mathbf{X}\mathbf{W}_V) \in \mathbb{R}^{N \times d_{\text{node}}} \tag{15}$$

for learnable weight matrices $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d_{\text{node}} \times d_{\text{node}}}$, though in practice we use 32 heads.

The FFN module Finally, the feed-forward network module takes the form:

$$\mathbf{y} = \texttt{FFN}(\mathbf{x}) \tag{16}$$

$$\text{computed as} \quad \bar{\mathbf{x}} = \texttt{Dropout}_p(\texttt{GELU}(\texttt{Dense}(\mathbf{x})) \in \mathbb{R}^{4d_{\text{node}}} \tag{17}$$

$$\mathbf{y} = \texttt{GraphDropout}_{\frac{\ell}{L}0.3}(\texttt{Dense}(\bar{\mathbf{x}})) + \mathbf{x} \in \mathbb{R}^{d_{\text{node}}} \tag{18}$$

Unless otherwise stated, the dropout probability $p = 0$, however, we experiment with other values when ensembling multiple model variants.

Table 1 shows how the learnable weights are distributed through the model. Interestingly, 70% of the total parameters are located in the MPNN, highlighting the focus of this model using local structures and exploiting the inductive bias of the graph.

3.2 Input Feature Engineering

As described in Section 2, the dataset samples include the graph structure $\mathcal{G}$, a set of categorical features for the atoms and bonds $\mathbb{x}_i$, $\mathbb{e}_{uv}$, and the 3D node positions $\mathbf{r}_i$. It has been shown that there are many benefits to augmenting the

input data with additional structural, positional, and chemical information [Rampášek et al., 2022, Wang et al., 2022a, Dwivedi et al., 2022]. Therefore, we combine several feature sources when computing the input to the first GPS++ layer. There are four feature tensors to initialise; node state, edge state, whole graph state and attention biases.

$$\mathbf{X}^0 = \texttt{Dense}\left(\left[\mathbf{X}^{\text{atom}} \mid \mathbf{X}^{\text{LapVec}} \mid \mathbf{X}^{\text{LapVal}} \mid \mathbf{X}^{\text{Cent}} \mid \mathbf{X}^{\text{3D}}\right]\right) \qquad \in \mathbb{R}^{N \times d_{\text{node}}} \tag{19}$$

$$\mathbf{E}^0 = \texttt{Dense}\left(\left[\mathbf{E}^{\text{bond}} \mid \mathbf{E}^{\text{3D}}\right]\right) \qquad \in \mathbb{R}^{M \times d_{\text{edge}}} \tag{20}$$

$$\mathbf{g}^0 = \texttt{Embed}_{d_{\text{global}}}(0) \qquad \in \mathbb{R}^{d_{\text{global}}} \tag{21}$$

$$\mathbf{B} = \mathbf{B}^{\text{SPD}} + \mathbf{B}^{\text{3D}} \qquad \in \mathbb{R}^{N \times N} \tag{22}$$

The various components of each of these equations are defined over the remainder of this section. The encoding of these features also makes recurring use of the following two generic functions. Firstly, a two-layer $\texttt{MLP}_{\text{encoder}}$ that projects features to a fixed-size latent space:

$$y = \texttt{MLP}_{\text{encoder}}(x), \quad \text{where } x \in \mathbb{R}^h \tag{23}$$

$$\text{computed as} \quad \bar{x} = \texttt{ReLU}(\texttt{Dense}(\texttt{LayerNorm}(x))) \qquad \in \mathbb{R}^{2h} \tag{24}$$

$$y = \texttt{Dropout}_{0.18}(\texttt{Dense}(\texttt{LayerNorm}(\bar{x}))) \qquad \in \mathbb{R}^{32} \tag{25}$$

Secondly, a function $\texttt{Embed}_d(j) \in \mathbb{R}^d$ which selects the j^{th} row from an implicit learnable weight matrix.

Chemical Features The categorical features $\mathbb{x}$, $\mathbb{e}$ supplied with the original dataset represent a set of 9 atom and 3 bond features (described in Table 2). There are, however, a wide range of chemical features that can be extracted from the periodic table or using tools like RDKit. Ying et al. [2021b] have shown that extracting additional atom level properties can be beneficial when trying to predict the HOMO-LUMO energy gap, defining a total of 28 atom and 5 bond features. We explore the impact of a number of additional node and edge features and sweep a wide range of possible combinations.

In particular, we expand on the set defined by Ying et al. [2021b] with three additional atom features derived from the periodic table, the atom group (column), period (row) and element type (often shown by colour). We found that these three additional features were particularly beneficial. Furthermore, we hoped that this would allow us to drop the atomic number, which can be determined from the combination of group and period for all elements occurring in the dataset, enabling the model to generalise to atoms not seen in the training set. However, we found that atomic number was important to keep even in the presence of these additional features.

We also found that in many cases *removing* features was beneficial, for example, we found that all our models performed better when excluding information about chiral tag and replacing it by chiral centers. We further observe that our best feature combinations all consist of only 8 node features, where the majority of the input features stay consistent between the sets. We show the three best feature sets found in Table 2 and use *Set 1* for all experiments unless otherwise stated (e.g., during ensembling).

Finally, to embed the categorical chemical features from the dataset $\mathbb{x}_i$, $\mathbb{e}_{uv}$ into a continuous vector space, we learn a simple embedding vector for each category, sum the embeddings for all categories, and then process it with an MLP to produce $\mathbf{X}^{\text{atom}}$ and $\mathbf{E}^{\text{bond}}$, i.e.

$$\forall i: \quad \mathbf{x}_i^{\text{atom}} = \texttt{Dropout}_{0.18}\left(\texttt{MLP}_{\text{node}}\left(\sum\nolimits_{j \in \mathbb{x}_i} \texttt{Embed}_{64}(j)\right)\right) \in \mathbb{R}^{d_{\text{node}}} \tag{26}$$

$$\forall (u,v): \quad \mathbf{e}_{uv}^{\text{bond}} = \texttt{Dropout}_{0.18}\left(\texttt{MLP}_{\text{edge}}\left(\sum\nolimits_{j \in \mathbb{e}_{uv}} \texttt{Embed}_{64}(j)\right)\right) \in \mathbb{R}^{d_{\text{edge}}} \tag{27}$$

Here $\texttt{MLP}_{\text{node}}$ and $\texttt{MLP}_{\text{edge}}$ refer to the functions by the same names used in Eq. 6 and 7 in the MPNN module, yet parameterised independently.

Graph Laplacian Positional Encodings [Kreuzer et al., 2021, Dwivedi and Bresson, 2020] Given a graph with adjacency matrix $\mathbf{A}$ and degree matrix $\mathbf{D}$, the eigendecomposition of the graph laplacian $\mathbf{L}$ is formulated into a global positional encoding as follows.

$$\forall i: \quad \mathbf{x}_i^{\text{LapVec}} = \texttt{MLP}_{\text{encoder}}(\mathbf{U}\left[i, 2 \dots k^{\text{Lap}}\right]) \in \mathbb{R}^{32}, \quad \text{where } \mathbf{L} = \mathbf{D} - \mathbf{A} = \mathbf{U}^\top \mathbf{\Lambda} \mathbf{U} \tag{28}$$

$$\forall i: \quad \mathbf{x}_i^{\text{LapVal}} = \texttt{MLP}_{\text{encoder}}\left(\frac{\mathbf{\Lambda}'}{||\mathbf{\Lambda}'||}\right) \in \mathbb{R}^{32}, \quad \text{where } \mathbf{\Lambda}' = \text{diag}(\mathbf{\Lambda})\left[2 \dots k^{\text{Lap}}\right] \tag{29}$$

To produce fixed shape inputs despite variable numbers of eigenvalues / eigenvectors per graph, we truncate / pad to the lowest 7 eigenvalues, excluding the first trivial eigenvalue $\Lambda_{11} = 0$. We also randomise the eigenvector sign every epoch which is otherwise arbitrarily defined.

Table 2: Chemical input feature selection for PCQM4Mv2 dataset.

	Feature Set			
Node features	**Original**	**Set 1**	**Set 2**	**Set 3**
Atomic number	✓	✓	✓	✓
Group	×	✓	✓	✓
Period	×	✓	✓	✓
Element type	×	✓	✓	✓
Chiral tag	✓	×	×	×
Degree	✓	✓	✓	✓
Formal charge	✓	✓	×	✓
# Hydrogens	✓	✓	✓	✓
# Radical electrons	✓	✓	✓	✓
Hybridisation	✓	×	✓	✓
Is aromatic	✓	✓	✓	×
Is in ring	✓	✓	✓	✓
Is chiral center	×	✓	✓	✓
Edge features				
Bond type	✓	✓	×	×
Bond stereo	✓	✓	✓	✓
Is conjugated	✓	×	✓	✓
Is in ring	×	✓	✓	✓

Random Walk Structural Encoding [Dwivedi et al., 2022] This feature captures the probability that a random graph walk starting at node i will finish back at node i, and is computed using powers of the transition matrix $\mathbf{P}$. This feature captures information about the local structures in the neighbourhood around each node, with the degree of locality controlled by the number of steps. For this submission random walks from 1 up to $k^{\mathrm{RW}} = 16$ steps were computed to form the feature vector.

$$\forall i: \quad \mathbf{x}_i^{\mathrm{RW}} = \mathtt{MLP}_{\mathrm{encoder}}\left(\left[(\mathbf{P}^1)_{ii}, (\mathbf{P}^2)_{ii}, \cdots, (\mathbf{P}^{k^{\mathrm{RW}}})_{ii}\right]\right) \in \mathbb{R}^{32}, \quad \text{where } \mathbf{P} = \mathbf{D}^{-1}\mathbf{A} \tag{30}$$

Local Graph Centrality Encoding [Ying et al., 2021a, Shi et al., 2022] The graph centrality encoding is intended to allow the network to gauge the importance of a node based on its connectivity, by embedding the degree (number of incident edges) of each node into a learnable feature vector.

$$\forall i: \quad \mathbf{x}_i^{\mathrm{Cent}} = \mathtt{Embed}_{64}\left(D_{ii}\right) \in \mathbb{R}^{64} \tag{31}$$

Shortest Path Distance Attention Bias Graphormer [Ying et al., 2021a, Shi et al., 2022] showed that graph topology information can be incorporated into a node transformer by adding learnable biases to the self-attention matrix depending on the distance between node pairs. During data preprocessing the Shortest Path Distance (SPD) map $\Delta \in \mathbb{N}^{N\times N}$ is computed where Δ_{ij} is the number of edges in the shortest continuous path from node i to node j. During training each integer distance is embedded as a scalar attention bias term to create the SPD attention bias map $\mathbf{B}^{\mathrm{SPD}} \in \mathbb{R}^{N\times N}$.

$$\forall i,j: \quad B_{ij}^{\mathrm{SPD}} = \mathtt{Embed}_1\left(\Delta_{ij}\right) \in \mathbb{R} \tag{32}$$

Single-headed attention is assumed throughout this report for simplified notation, however, upon extension to multi-headed attention, one bias is learned per distance per head.

Embedding 3D Distances Using the 3D positional information provided by the dataset comes with a number of inherent difficulties. Firstly, the task is invariant to molecular rotations and translations, however, the 3D positions themselves are not. Secondly, the 3D conformer positions are only provided for the training data, not the validation or test data. To deal with these two issues and take advantage of the 3D positions provided we follow the approach of Luo et al. [2022].

To ensure rotational and translational invariance we use only the distances between atoms, not the positions directly. To embed the scalar distances into vector space $\mathbb{R}^K$ we first apply $K = 128$ Gaussian kernel functions, where the k^{th} function is defined as

$$\forall i,j: \quad \psi_{ij}^k = -\frac{1}{\sqrt{2\pi}\,|\sigma^k|}\exp\left(-\frac{1}{2}\left(\frac{||\mathbf{r}_i - \mathbf{r}_j|| - \mu^k}{|\sigma^k|}\right)^2\right) \in \mathbb{R} \tag{33}$$

with learnable parameters μ^k and σ^k. The K elements are concatenated into vector $\boldsymbol{\psi}_{ij}$. We then process these distance embeddings in three ways to produce attention biases, node features and edge features.

3D Distance Attention Bias The 3D attention bias map $\mathbf{B}^{\text{3D}} \in \mathbb{R}^{N \times N}$ allows the model to modulate the information flowing between two node representations during self-attention based on the spatial distance between them, and are calculated as per Luo et al. [2022]

$$\forall i, j: \quad \mathbf{B}^{\text{3D}}_{ij} = \texttt{MLP}_{\text{bias3D}}(\boldsymbol{\psi}_{ij}) \in \mathbb{R} \tag{34}$$

Upon extension to multi-headed attention with 32 heads $\texttt{MLP}_{\text{bias3D}}$ instead projects to $\mathbb{R}^{32}$.

Bond Length Encoding Whilst $\mathbf{B}^{\text{3D}}$ makes inter-node distance information available to the self-attention module in a dense all-to-all manner as a matrix of simple scalar biases, we also make this information available to the `MPNN` module in a sparse but high-dimensional manner as edge features $\mathbf{E}^{\text{3D}} = \left[\mathbf{e}^{\text{3D}}_{uv} \text{ for } (u, v) \in \mathcal{E}\right]$ calculated as

$$\forall (u, v): \quad \mathbf{e}^{\text{3D}}_{uv} = \texttt{MLP}_{\text{encoder}}(\boldsymbol{\psi}_{uv}) \in \mathbb{R}^{32} \tag{35}$$

Global 3D Centrality Encoding The 3D node centrality features $\mathbf{X}^{\text{3D}} = \left[\mathbf{x}^{\text{3D}}_1; \ldots; \mathbf{x}^{\text{3D}}_N\right]$ are computed by summing the embedded 3D distances from node i to all other nodes. Since the sum commutes this feature cannot be used to determine the distance to a specific node, so serves as a centrality encoding rather than a positional encoding.

$$\forall i: \quad \mathbf{x}^{\text{3D}}_i = W^{\text{3D}} \sum_{j \in \mathcal{V}} \boldsymbol{\psi}_{ij} \in \mathbb{R}^{32} \tag{36}$$

Here $W^{\text{3D}} \in \mathbb{R}^{K \times 32}$ is a linear projection to the same latent size as the other encoded features.

4 Experimental Setup

4.1 Hardware and Acceleration

Hardware We train our models using a Graphcore BOW-POD16 which contains 16 IPU processors, delivering a total of 5.6 petaFLOPS of float16 compute and 14.4 GB of in-processor SRAM which is accessible at an aggregate bandwidth of over a petabyte per second. This compute and memory is then distributed evenly over 1472 tiles per chip. This architecture has two key attributes that enable high performance on GNN and other AI workloads [Bilbrey et al., 2022]: memory is kept as close to the compute as possible (i.e., using on-chip SRAM rather than off-chip DRAM) which maximises bandwidth for a nominal power budget; and compute is split up into many small independent arithmetic units meaning that any available parallelism can be extremely well utilised. In particular this enables very high performance for sparse communication ops, like gather and scatter, and achieves high FLOP utilisation even with complex configurations of smaller matrix multiplications. Both of these cases are particularly prevalent in MPNN structures like those found in GPS++.

To exploit the architectural benefits of the IPU and maximise utilisation, understanding the program structure ahead of time is key. This means all programs must be compiled end-to-end, opening up a range of opportunities for optimisation but also adding the constraint that tensor shapes must be known and fixed at compile time.

Batching and Packing To enable fixed tensor sizes with variable sized graphs it is common to *pad* the graphs to the max node and edge size in the dataset. This, however, can lead to lots of compute being wasted on padding operations, particularly in cases where there are large variations in the graph sizes. To combat this it is common to *pack* a number of graphs into a fixed size shape to minimise the amount of padding required, this is an abstraction that is common in graph software frameworks like PyTorch Geometric [Fey and Lenssen, 2019] and has been shown to achieve as much as 2x throughput improvement for variable length sequence models [Krell et al., 2021]. Packing graphs into one single large pack, however, has a couple of significant downsides: the memory and compute complexity of all-to-all attention layers is $\mathcal{O}(n^2)$ in the pack size not the individual graph sizes, and allowing arbitrary communication between all nodes in the pack forces the compiler to choose sub-optimal parallelisation schemes for the gather/scatter operations.

To strike a balance between these two extremes we employ a two tiered hierarchical batching scheme that packs graphs into a fixed size but then batches multiple packs to form the micro-batch. We define the maximum pack size to be 60 nodes, 120 edges and 8 graphs then use a simple streaming packing method where graphs are added to the pack until either the total nodes, edges or graphs exceeds the maximum size. This achieves 87% packing efficiency of the nodes and edges with on average 3.6 graphs per pack, though we believe that this could be increased by employing a more complex packing strategy Krell et al. [2021]. We then form micro-batches of 8 packs which are pipelined [Huang et al., 2018] over 4 IPUs accumulating over 8 micro-batches and replicated 4 times to form a global batch size of 921 graphs distributed over 16 IPUs.

Numerical Precision To maximise compute throughput and maximise memory efficiency it is now common practice to use lower precision numerical formats in deep learning [Micikevicius et al., 2017]. On Graphcore IPUs using float16 increases the peak FLOP rate by 4x compared to float32 but also makes more effective usage of the high bandwidth on-chip SRAM. For this reason we use float16 for nearly all[1] compute but also use float16 for the majority[2] of the weights, this is made possible, without loss of accuracy, by enabling the hardware-level stochastic rounding of values. While we do use a loss scaling Micikevicius et al. [2017] value of 1024 we find that our results are robust to a wide range of choices. We also use float16 for the first-order moment in Adam but keep the second-order moment in float32 due to the large dynamic range requirements of the sum-of-squares.

4.2 Model Training

Training Configuration Our model training setup uses the Adam optimiser [Kingma and Ba, 2015] with a gradient clipping value of 5, a peak learning rate of 4e-4 training for a total of 450 epochs. We used a learning rate warmup period of 10 epochs followed by a linear decay schedule.

Decoder and Loss The final model prediction is formed by global sum-pooling of all node representations and then passing it through a 2-layer MLP. The regression loss is the mean absolute error (L1 loss) between a scalar prediction and the ground truth HOMO-LUMO gap value.

Noisy Nodes/Edges Noisy nodes [Godwin et al., 2022, Zaidi et al., 2022] has previously been shown to be beneficial for molecular GNNs including on the PCQM4M dataset. The method adds noise to the input data then tries to reconstruct the uncorrupted data in an auxilliary task. Its benefits are claimed to be two-fold: it adds regularisation by inducing some noise on the input, but also combats over-smoothing by forcing the node level information to remain discriminative throughout the model. This has been shown to be particularly beneficial when training deep GNNs [Godwin et al., 2022]. We follow the method of Godwin et al. [2022] that applies noise to the categorical node features by randomly choosing a different category with probability p_{corrupt} but also extend this to the categorical edge features, too. A simple categorical cross entropy loss is then used to reconstruct the uncorrupted features at the output. We set $p_{\text{corrupt}} = 0.01$ and weight the cross-entropy losses such that they have a ratio 1:1.2:1.2 for losses `HOMO-LUMO:NoisyNodes:NoisyEdges`.

Grouped Input Masking As described in Section 2 the 3D positional features $\mathbf{R}$ are only defined for the training data. We must therefore make use of them in training without requiring them for validation/test. We found that the method proposed by Luo et al. [2022] achieved the most favourable results so we adopt a variation hereon referred to as *grouped input masking*.

This method stochastically masks out any features derived from the 3D positional features $\mathbf{R}$ to build robustness to their absence. Specifically, this is done by defining two input sets to be masked:

$$\mathcal{X}^{\text{Spatial}} = \{\mathbf{X}^{\text{3D}}, \mathbf{E}^{\text{3D}}, \mathbf{B}^{\text{3D}}\}, \quad \mathcal{X}^{\text{Topological}} = \{\mathbf{B}^{\text{SPD}}\}, \tag{37}$$

and three potential masking groups: 1. Mask $\mathcal{X}^{\text{Spatial}}$, 2. Mask $\mathcal{X}^{\text{Topological}}$, and 3. No masking. These masking groups are then sampled randomly throughout training with ratio 1:3:1. If 3D positions are not defined, for example in validation/test, masking group 1 is always used.

Training Time Our `GPS++` model trains at 17500 graphs per second on 16 IPUs which means each epoch completes in 3 minutes and a full 450 epoch training run takes 24 hours.

Dataset Splits While the original training set already contains 98% of the labelled data and all of the data with 3D positional information, we still aim to train on all available data for our final model. This, however, comes with many pitfalls due to inability to calculate a reliable measure of performance. Therefore, to understand the value of this additional data we consider an intermediate split configuration where we randomly sample half of the validation set (not resampled per run) to train on, holding the other half out for validation. As a result we have three dataset splits to consider: `original`, `train+valid` and `train+half_valid`.

Ensembling Ensembling models has long been used to improve generalisation of machine learning models and has become an indispensable tool for practitioners entering machine learning competitions. Here we outline our ensembling strategy that aims to achieve confidence while training without a validation set, but also allows on-the-fly tuning and weighting of model ensembles. The main idea is to build two comparative model sets to ensemble, a *proxy* set and a

[1] A few operations like the sum of squares the variance calculations are up-cast to float32 by the compiler.

[2] A small number of weights are kept in float32 for simplicity of the code rather than numerical stability.

Table 3: Comparing single model performance on PCQM4Mv2 dataset.

Model	Model Type	Validation MAE ↓	# Param.
GRPE [Park et al., 2022]	Transformer	0.0890	46.2M
EGT [Hussain et al., 2022]	Transformer	0.0869	89.3M
Graphormer [Shi et al., 2022]	Transformer	0.0864	48.3M
GPS [Rampášek et al., 2022]	Hybrid	0.0858	19.4M
GEM-2 [Liu et al., 2022]	Transformer	0.0793	32.1M
Global-ViSNet [Wang et al., 2022b]	Transformer	0.0784	78.5M
Transformer-M [Luo et al., 2022]	Transformer	0.0772	69M
GPS++	Hybrid	0.0778	44.3M
GPS++*	Hybrid	0.0755	44.3M

* Trained on `train+half_valid` data split.

Table 4: Ensembled model performance on PCQM4Mv2 dataset. Models in the proxy set are trained on the `train+half_valid` data split whereas those in the full set are trained on all available data.

	Proxy Set			Main Set	
		Valid MAE			Ensembling
Case	# Models	Avg.	Ensembled	# Models	Weight
1: Baseline	10	0.0755	0.0725	35	1
2: No Atomic Number	4	0.0761	0.0734	16	0.5
3: FNN Dropout = 0.412	8	0.0759	0.0729	14	1
4: FNN Dropout = 0.412; No Atomic Number	5	0.0761	0.0736	7	0.5
5: Feature Set 2†	4	0.0755	0.0731	15	1
6: Feature Set 3†	4	0.0754	0.0731	14	1
7: Masking Weights = [1,2,2]	4	0.0754	0.0730	15	1
All	**39**	**0.0756**	**0.0722**	**112**	

† As defined in Table 2.

main set. The proxy set is designed to be qualitatively similar to the main set but maintain a clean held out validation set by training on `train+half_valid`, whereas the main set is trained on `train+valid` . We aim for the proxy set to match the main set in all aspects apart from the training data and the number of models; to be able to focus computational resources on the main set we aim to build the proxy set to be approximately 25% of its size.

5 Results

Single Model Performance In Table 3 we compare the single model performance of our model with prior work. It shows that we achieve comparable performance to the best Transformer only model [Luo et al., 2022] with just 64% of the parameters. We believe that this efficiency is driven by the strong inductive bias in the message passing layers which constitute a large proportion of the compute, as also noted in Rampášek et al. [2022] when using an MPNN and in Kreuzer et al. [2021] when biasing the attention towards direct neighbours. Furthermore, throughout testing we found that without the inclusion of the 3D distance matrix the self-attention layers often only provided minimal improvement. We therefore hypothesise that one of the main strengths of the attention in this scenario is that it is a highly effective place to integrate the three dimensional distance matrix. We plan to further investigate these points in later revisions of this work.

Ensembled Model Performance Diversity of models in an ensemble is a well known way to boost model performance [Zhou et al., 2002, Lakshminarayanan et al., 2017]. In this work we propose six adjustments to the hyperparameters to form seven different model configurations to ensemble. In choosing these configurations we aim to build diversity in three main areas: input feature description (cases 2, 4, 5 and 6), regularisation strength (cases 2 and 3) and reliance on 3D features (case 7). As described in Section 4 we build a *proxy* set of models to help guide our ensembling strategy as well as a *main* set trained on all available data. The details of our seven configurations and their evaluation are shown in Table 4.

Analysing the proxy set we found that reducing the weighting for the two worst performing models gave a small improvement to our final ensembled MAE, we therefore apply this is our main final ensemble too.

The final 112 model ensemble achieves an MAE of **0.0719** on the test challenge set as reported by the competition organisers. This was completed in 1 hour 32 minutes using 1 IPU and AMD EPYC 7742 64-Core CPU including all the feature processing, program compilation and model inference, which is well under the 4 hour budget allocated.

6 Conclusions

In this work we define GPS++, a hybrid MPNN/Transformer model, optimised for the PCQM4Mv2 molecular property prediction task [Hu et al., 2021]. Our model builds on the works of Rampášek et al. [2022], Luo et al. [2022], Godwin et al. [2022] with a particular focus on building a powerful and expressive message passing component. We believe that the strong inductive bias induced by this part of the model is a strong driver behind the efficiency of this model which achieves performance comparable to the best published transformer while using only 64% of the parameters.

To achieve the best overall model performance we build a diverse ensemble of 112 models and train all of these models on all the available data to form our submission to the OGB-LSC 2022 challenge. Our final GPS++ ensemble achieved `test-challenge` MAE of **0.0719**, placing amongst the top-3 winners for this dataset.

Acknowledgements

We would like to thank all the people from our respective organisations that have supported this work, in particular Douglas Orr, Carlo Luschi, Andrew Fitzgibbon and Ellie Dobson from Graphcore; Therence Bois and Prudencio Tossou from Valence Discovery; and Michael Galkin from Mila.

References

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *34th Conference on Neural Information Processing Systems*, 2020.

Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. OGB-LSC: A large-scale challenge for machine learning on graphs. In *35th Conference on Neural Information Processing Systems: Datasets and Benchmarks Track*, 2021.

W. Kohn and L. J. Sham. Self-consistent equations including exchange and correlation effects. *Phys. Rev.*, 140:A1133–A1138, Nov 1965. doi:10.1103/PhysRev.140.A1133. URL https://link.aps.org/doi/10.1103/PhysRev.140.A1133.

Christopher M. Dobson. Chemical space and biology. *Nature*, 432(7019):824–828, December 2004. ISSN 1476-4687. doi:10.1038/nature03192. URL https://www.nature.com/articles/nature03192.

Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a General, Powerful, Scalable Graph Transformer. *arXiv:2205.12454*, 2022.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261*, 2018.

Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv:2104.13478*, 2021.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv:1607.06450*, 2016.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv:1606.08415*, 2016.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In *Advances in Neural Information Processing Systems*, 2021a.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

Haorui Wang, Haoteng Yin, Muhan Zhang, and Pan Li. Equivariant and stable positional encoding for more powerful graph neural networks. In *International Conference on Learning Representations*, 2022a.

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. In *International Conference on Learning Representations*, 2022.

RDKit. RDKit: Open-source cheminformatics. http://www.rdkit.org. [Online].

Chengxuan Ying, Mingqi Yang, Shuxin Zheng, Guolin Ke, Shengjie Luo, Tianle Cai, Chenglin Wu, Yuxin Wang, Yanming Shen, and Di He. First place solution of KDD Cup 2021 & OGB large-scale challenge graph prediction track. *arXiv:2106.08279*, 2021b.

Devin Kreuzer, Dominique Beaini, William L. Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. In *Advances in Neural Information Processing Systems*, 2021.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv:2012.09699*, 2020.

Yu Shi, Shuxin Zheng, Guolin Ke, Yifei Shen, Jiacheng You, Jiyan He, Shengjie Luo, Chang Liu, Di He, and Tie-Yan Liu. Benchmarking graphormer on large-scale molecular modeling datasets. *arXiv:2203.04810*, 2022.

Shengjie Luo, Tianlang Chen, Yixian Xu, Shuxin Zheng, Tie-Yan Liu, Liwei Wang, and Di He. One transformer can understand both 2d & 3d molecular data. *arXiv:2210.01765*, 2022.

Jenna A. Bilbrey, Kristina M. Herman, Henry Sprueill, Soritis S. Xantheas, Payel Das, Manuel Lopez Roldan, Mike Kraus, Hatem Helal, and Sutanay Choudhury. Reducing down(stream)time: Pretraining molecular gnns using heterogeneous ai accelerators. *arXiv preprint arXiv:2211.04598*, 2022.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Mario Michael Krell, Matej Kosec, Sergio P. Perez, and Andrew Fitzgibbon. Efficient sequence packing without cross-contamination: Accelerating large language models without impacting performance. *arXiv preprint arXiv:2107.02027*, 2021.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965*, 2018.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015. URL http://arxiv.org/abs/1412.6980.

Jonathan Godwin, Michael Schaarschmidt, Alexander L Gaunt, Alvaro Sanchez-Gonzalez, Yulia Rubanova, Petar Veličković, James Kirkpatrick, and Peter Battaglia. Simple GNN regularisation for 3D molecular property prediction and beyond. In *International Conference on Learning Representations*, 2022.

Sheheryar Zaidi, Michael Schaarschmidt, James Martens, Hyunjik Kim, Yee Whye Teh, Alvaro Sanchez-Gonzalez, Peter Battaglia, Razvan Pascanu, and Jonathan Godwin. Pre-training via denoising for molecular property prediction. *arXiv preprint arXiv:2206.00133*, 2022.

Wonpyo Park, Woonggi Chang, Donggeon Lee, Juntae Kim, and Seung won Hwang. GRPE: Relative positional encoding for graph transformer. *arXiv:22201.12787*, 2022.

Md Shamim Hussain, Mohammed J Zaki, and Dharmashankar Subramanian. Global self-attention as a replacement for graph convolution. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 655–665, 2022.

Lihang Liu, Donglong He, Xiaomin Fang, Shanzhuo Zhang, Fan Wang, Jingzhou He, and Hua Wu. GEM-2: Next generation molecular property prediction network with many-body and full-range interaction modeling. *arXiv preprint arXiv:2208.05863*, 2022.

Yusong Wang, Shaoning Li, Tong Wang, Zun Wang, Xinheng He, Bin Shao, and Tie-Yan Liu. How to better introduce geometric information in equivariant message passing? https://github.com/ogb-visnet/Global-ViSNet/blob/master/ViSNet_Tech_Report.pdf, 2022b. Accessed: 2022-11-16.

Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. Ensembling neural networks: many could be better than all. *Artificial intelligence*, 137(1-2):239–263, 2002.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.