Pacific Northwest
NATIONAL LABORATORY

PNNL-36839

Unifying Combinatorial and Graphical Methods in Artificial Intelligence

September 2024

Sinan G. Aksoy
Bo Fang
Roberto Gioiosa (co-PI)
Bill Kay
Hyungro Lee
Jenna Pope
Madelyn Shapiro
Stephen J. Young (co-PI)

U.S. DEPARTMENT OF ENERGY

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor Battelle Memorial Institute, nor any of their employees, makes **any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights**. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or Battelle Memorial Institute. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from
the Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062
www.osti.gov
ph: (865) 576-8401
fox: (865) 576-5728
email: reports@osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
or (703) 605-6000
email: info@ntis.gov
Online ordering: http://www.ntis.gov

PNNL-36839

Unifying Combinatorial and Graphical Methods in Artificial Intelligence

September 2024

Sinan G. Aksoy
Bo Fang
Roberto Gioiosa (co-PI)
Bill Kay
Hyungro Lee
Jenna Pope
Madelyn Shapiro
Stephen J. Young (co-PI)

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99354

Summary

Recently, a new graph Laplacian, called the inner product Laplacian, was introduced which generalizes many existing Laplacians, including the normalized and combinatorial Laplacian and their weighted variants. The key observation behind the inner product Laplacian is that by defining appropriate inner product spaces on the vertices and edges, the standard Laplacians can be recovered as Hodge Laplacians over the simplicial complex formed by the edges and vertices. These inner product spaces form a natural way to incorporate non-combinatorial information into the definition of a domain-specific Laplacian. In particular, in contrast to current domain-specific weighting schemes which rely solely on edge weights, information regarding the similarity of non-adjacent vertices and arbitrary pairs of edges can be effectively incorporated into the Laplacian. In order to illustrate this approach, we consider the problem of calculating the potential energy of an atomistic configuration using Graph Neural Networks. In comparison with start-of-the-art approaches, such as SchNet, our approach replaces a learned (via auto-encoder) representation of the atom types with an inner product space on atoms based on scientific knowledge (e.g., electronegativity). Our results show that if there is significant domain information that can be encoded into the inner product, replacing the normalized Laplacian with an inner product Laplacian can result in over a 10-fold reduction in training loss (which is transferable to validation loss). However, when there is relatively little domain knowledge to be incorporated the performance of the standard graph convolutional layers is 2-3 times better (in terms of absolute error). Likely this performance difference highlights the ability of networks like SchNet to rapidly “learn” a representation of a small amount of domain knowledge tailored to the particular use case. This hypothesis is reenforced by experiments where the chosen edge inner product is tuned to resulting in a 2-fold decrease the error.

However, to compute the resulting Laplacian involves a mixture of sparse and dense matrix computation and yields a dense matrix as the basis for the graph convolution. This dense convolutional kernel necessitates moving away from the standard message passing framework for graph neural networks and increases the computational cost of applying the kernel. In order to mitigate these costs we investigate means of leveraging the mixed sparse and dense computations to reduce the overall computational cost and how these approaches can be automatically transferred to energy efficient hardware (e.g., field programmable gate arrays (FPGAs)). Overall, the size of the training examples (no more than 50 atoms in a molecules) means that the overhead of advanced algorithmic approaches results in lesser performance both in terms of overall runtime and energy consumption. However, individual kernels (such as the transpose) can benefit from custom accelerators such as FPGA in terms of energy performance while still providing comparable runtimes. We speculate that larger graphs (such as those arising from the power grid) may make advanced algorithmic approaches more feasible in terms of energy usage and computational time.

Acknowledgments

This research was supported by the **Mathematics for Artificial Reasoning in Science (MARS)** initiative, under the Laboratory Directed Research and Development (LDRD) Program at Pacific Northwest National Laboratory (PNNL). PNNL is a multi-program national laboratory operated for the U.S. Department of Energy (DOE) by Battelle Memorial Institute under Contract No. DE-AC05-76RL01830.

Contents

List of Figures

List of Tables

PNNL-36839

UNIFYING COMBINATORIAL AND GRAPHICAL METHODS IN ARTIFICIAL INTELLIGENCE

SINAN G. AKSOY, BO FANG, ROBERTO GIOIOSA, BILL KAY, HYUNGRO LEE, JENNA POPE, MADELYN SHAPIRO, AND STEPHEN J. YOUNG

Abstract. Recently, a new graph Laplacian, called the inner product Laplacian, was introduced which generalizes many existing Laplacians, including the normalized and combinatorial Laplacian and their weighted variants. The key observation behind the inner product Laplacian is that by defining appropriate inner product spaces on the vertices and edges, the standard Laplacians can be recovered as Hodge Laplacians over the simplicial complex formed by the edges and vertices. These inner product spaces form a natural way to incorporate non-combinatorial information into the definition of a domain-specific Laplacian. In particular, in contrast to current domain-specific weighting schemes which rely solely on edge weights, information regarding the similarity of non-adjacent vertices and arbitrary pairs of edges can be effectively incorporated into the Laplacian. In order to illustrate this approach we consider the problem of calculating the potential energy of an atomistic configuration using Graph Neural Networks. In comparison with start-of-the-art approaches, such as SchNet, our approach replaces a learned (via auto-encoder) representation of the atom types with an inner product space on atoms based on scientific knowledge (e.g., electronegativity). We will illustrate how this approach captures key chemical properties of the molecules and compare the energy calculations with state-of-the-art neural network approaches.

However, to compute the resulting Laplacian involves a mixture of sparse and dense matrix computation and yields a dense matrix as the basis for the graph convolution. This dense convolutional kernel necessitates moving away from the standard message passing framework for graph neural networks and increases the computational cost of applying the kernel. In order to mitigate these costs we investigate means of leveraging the mixed sparse and dense computations to reduce the overall computational cost and how these approaches can be automatically transferred to energy efficient hardware (e.g., field programmable gate arrays (FPGAs)).

1. Introduction

Over the last 10 years, the advent of efficient algorithms and hardware to train large scale neural networks has spawned an entirely new collection of approaches to scientific discovery, often referred to *Scientific Machine Learning* [2, 5]. While scientific machine learning has made great strides in advancing the frontiers of scientific discovery and continues to show potential to advance the state-of-the-art across a variety of different disciplines, there are significant obstacles to be overcome before the full potential of scientific machine learning can be reached; including characterizing epistemic and aleatoric uncertainty arising from machine learning pipelines, developing methods to identify "hallucinations" in large-scale models, reducing the size of the data sets needed to train machine learning pipelines, improving the training efficiency of specializing foundation models, reducing the overall energy usage needed for the training and implementation of large-scale machine learning models, and enabling the incorporation of extant knowledge into machine learning pipelines. In this effort, we focused on addressing the latter two challenges within the context of graph neural networks; in particular, we demonstrate the utility of the inner product Laplacian as a domain-informed graph convolutional kernel and investigate the methods by which the energy (and computational) costs of evaluated a dense graphical kernel, such as those given by the inner product Laplacian, can be reduced. As a motivating problem for our efforts, we will consider the efficient approximation of the potential energy associated with a molecular configuration. This provides a natural test case for our approach on multiple fronts; there is a wealth of high-quality data readily accessible (in the form of density-functional theory calculations of molecular potentials) which will facilitate investigation of the training procedure in a variety of use cases, neural network approximations for the potential often form the inner-loop of more computationally intensive workload (such as, molecular dynamics simulations) and so improvement in any

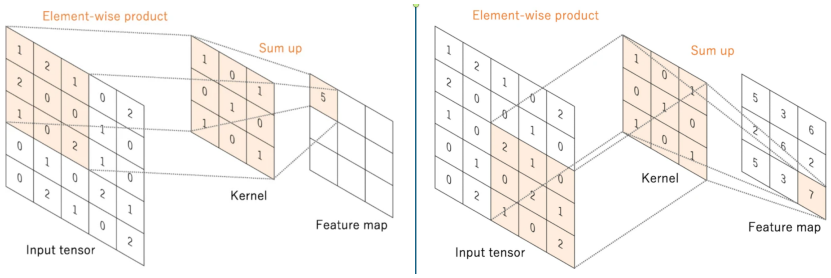


FIGURE 1. A 3×3 convolutional kernel applied to a 5×5 input tensor. Image and example courtesy of [26].

of the accuracy, computational speed, or energy usage will provide an outsized benefit, and finally there is wealth of readily accessible chemical knowledge which can be incorporated into our framework.

1.1. **Graph Neural Networks.** Before proceeding to the technical results of this effort, we review the notion of a *Graph Neural Network* (GNN). There are several broad classes of GNNs, motivated by our exemplar problem of of calculating the potential energy of a molecular system, we will focus primarily on the *Convolutional* GNNs. To this end, we first give a high level overview of Convolutional Neural Networks (CNNs) to introduce convolutional GNNs by analogy. For a survey of CNNs, see [21] and for a survey on GNNs see [25].

1.1.1. *Convolutional Neural Networks.* CNNs are a type of feed forward neural network that uses local convolution kernels as a method for averaging data. The archetypical use-case of a CNN is to classify images by considering average pixel values in small subgrids across a larger image, which is typically represented as a tensors with two dimensions corresponding to location of a pixel and the third dimension expressing the activation of various "channels" (e.g., cyan, magenta, and yellow or red, green, and blue). An example of how a single convolution layer works is illustrated in Figure 1. The 3×3 matrix in the middle represents a particular convolutional kernel applied to the input tensor with fixed (but learned) parameters. In this case, the learned parameters consist of an "X" of 5 ones with zeros in the other 4 entries. To apply the convolutional kernel to an input tensor, the Hadamard product of the convolutional kernel and with a selected subtensor of the input tensor is taken and then the entries are summed. The results are recorded in a (typically) smaller tensor with entries corresponding to distinct blocks in the input tensor (in this case the output is a 3×3 tensor corresponding to the collection of distinct 3×3 subtensors of the input tensor.

CNNs provide a valuable tool for coalescing data when data which is close (in the euclidean sense) is expected to be similar. In the motivating example of image processing, pixels are largely near similarly colored pixels. Within the larger feed forward network, the convolution layers are used to generate a feature map which is then used as an input to a feed forward network which is reduced in size while capturing the general feature layout of the data. The architecture of a simple CNN in which several convolutional layers are used as input to a short fully connected neural network is included in Figure 2.

While CNNs are incredibly powerful tool when locally close data is related, there are a number of use-cases in which physical proximity does not imply a quantitative similarity. In Section 1.1.2, we describe how the relational structure of a *graph* can inform a more general convolutional framework for data which is related in non-euclidean ways.

1.1.2. *Graph Neural Networks.* In general, graph neural network have a similar structure as convolutional neural networks except the convolutional layers are replaced with computational layers which are governed by a graph structure. Recall that, formally, a *graph* $G = (V, E)$ is a relational structure where:

- V is a set of objects (called *vertices*). The size of V is commonly denoted by n.
- E is a set of (unordred) pairs of vertices (called *edges*). The size of E is commonly denoted by m.

In practice, graphs are useful models of object of interest (represented by V) when pairwise relationships between those objects are important. A canonical example of a graph model is that of a social network, where V represents users of the social network and E represents when two members of the network are linked. In

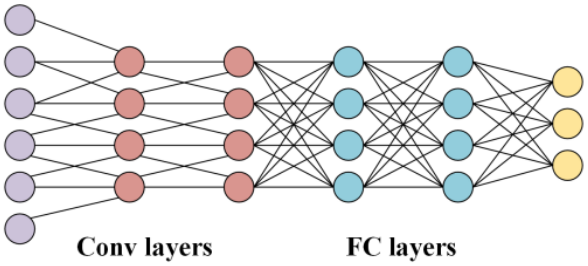


FIGURE 2. The architecture of a simple CNN. The violet layer corresponds to the input to the neural network, the mauve layers correspond to two convolutional layers being applied to the input, the teal layers constitute a fully connected (FC) neural network, and the yellow nodes correspond to the output layer. Image and example courtesy of [21].

our application, we will use a graph to model the relationships between distinct atoms in a molecular system. Each atom in the molecular system is a vertex and the edges represent a relationship between the atoms, either the presence of a bond or (as is the case in section 1.2) physical proximity of the atoms. In the case of the power grid, each vertex could model one of a variety of different pieces of electrical infrastructure (power bus, transformer, generator, switched shunt, etc.) and the edges would model the electrical connections between the various portions of the electrical infrastructure.

Given a data set related by a graphical model, we now describe how to perform convolution as a generalization of the methods described in Section 1.1.1. To this end, we will introduce notation we will use herein. Let $V = \{v_1, v_2, \ldots, v_n\}$ be an enumeration of the vertices of G. For $v \in V$, we denote the *neighborhood* of v by $N(v) := \{v' : \{v, v'\} \in E\}$. That is, the neighborhood of v is the set of vertices which are adjacent to v. In the example of a molecular system, the neighborhood of an atom is the set of other atoms which are bonded with that atom or are within some prescribed physical distance. The neighborhood will be important in describing how to perform convolution in a graph network.

Graph convolutional layers generalize classical convolutional layers by averaging information from vertices which are close, rather than data which is spatially close.[1] Irrespective of the averaging operation used, each vertex v is equipped with a state vector $\mathbf{x}_v$ and associated with each graph convoluational layer is a (learned) weight matrix $\mathbf{W}$ which is compatable with the state vectors. The simplest form of graph convolution combines information from *adjacent* vertices iteratively via the update:

$$\mathbf{x}'_v = \sum_{u \in N(v) \cup \{v\}} \mathbf{W}\mathbf{x}_u$$

where $\mathbf{x}'_v$ is the new state associated with the vertex v. This is often times discussed (and implemented) in terms of a message passing interface. That is, instead of viewing the update operation as each vertex pulling information from their neighborhood, one could image every vertex sending its current state out to its neighbors and each vertex combining the recieved states with their current state. There are a myriad of normalizations and modifications of this general framework [25], the most common being a normalizing factor which controls for the *degree* of a vertex v, where $\deg(v) := |N(v)|$ as follows:

$$\mathbf{x}'_v = \sum_{u \in N(v)} \frac{1}{\sqrt{\deg(u)\deg(v)}} \mathbf{W}\mathbf{x}_u.$$

Here we remark that traditional convolutional neural networks can be cast as graph convolution on grid graphs, illustrated in Figure 3.

1.2. **Neural Network Potentials.** Traditional methods for simulating atomic interactions in molecular and materials systems include quantum mechanical approaches like density functional theory (DFT) and empirically derived interatomic potentials (e.g., Lennard-Jones, embedded atom method). While DFT provides high accuracy, it is computationally expensive and scales poorly with system size. Conversely, empirical

[1]The following example is adapted from an informative article on *Towards Data Science* [20].

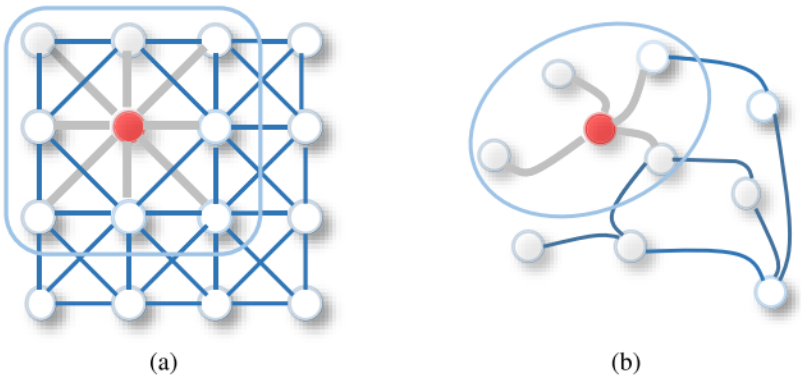


FIGURE 3. (a). Traditional convolution performed on a graph convolutional neural network. (b). General graph convolution. Image and example courtesy of [25].

interatomic potentials are computationally efficient but often lack the accuracy necessary to model complex materials systems. Interatomic potentials derived from neural networks, called neural network potentials (NNPs), provide an intermediate method, giving near-quantum mechanical accuracy while retaining computational efficiency comparable to empirical potentials [19, 17, 10]. NNPs are trained on quantum mechanical data, typically from DFT calculations, to learn the relationship between an atomic configuration and the system's corresponding energy. This relationship is then used in tasks such as molecular dynamics simulations, property prediction, and modeling chemical reactions.

Geometric NNPs are a class of machine learning models that utilize GNNs to predict interatomic interactions [11]. In these geometric NNPs, atoms are represented by nodes and the edges are defined based on the distance between atom pairs. Typically, a maximum distance cutoff is applied to limit the number of edges in the graph. Notably, geometric NNPs inherently respect the underlying symmetries of chemical systems, namely translation, rotation, and permutation invariances.

The typical network architecture of geometric NNPs is composed of three sequential phases: 1) node embedding, 2) node interactions, and 3) property readout. The node embedding step generates an initial internal representation h_i based on atom-level features, such as the atomic number. The interaction layers further refine the internal representation of each node through graph convolutions, as described above. The readout phase is typically composed of a multi-layer perceptron (MLP), which is a series of dense layers that transform the internal node representations to a scalar value. The scalars are then summed to provide system-level properties, in this case the potential energy. Backpropagation over the network with respect to the atomic coordinates provides the forces acting on each atom.

Figure 4 shows the basic architecture of the SchNet geometric NNP [24]. While SchNet considers only pairwise properties, specifically the Euclidean distance between atoms, newer architectures, such as DimeNet++ [13] and GemNet [12], incorporate 3- and 4-atom properties in terms of angles and dihedrals into the interaction block.

1.3. **Inner Product Laplacians.** As noted in Section 1.1.2, the defining features of a graph convolutional layer in a neural network are the graph, which defines a notion of neighborhood, and an averaging (or smoothing) operation over those neighborhoods. This smoothing operation can take many forms depending on the nature of the neural network structure. As noted in 1.1.2, one common smoothing function averages the neighborhood around v by weighting the contribution of the vertex u by $\frac{1}{\sqrt{\deg(v)\deg(u)}}$. If, instead of thinking of this smoothing operation acting on a per-vertex basis, but rather on all vertices simultaneously it can be recast as

$$\mathbf{X}' = D^{-1/2}AD^{-1/2}\mathbf{X}\mathbf{W}^T$$

where D is the diagonal degree matrix for the graph and A is the adjacency matrix of the graph. Those familiar with modern spectral graph theory will immediately notice the similarity of this matrix to the normalized Laplacian of the graph. Indeed, the normalized Laplacian $\mathcal{L}$ is the positive semidefinite matrix

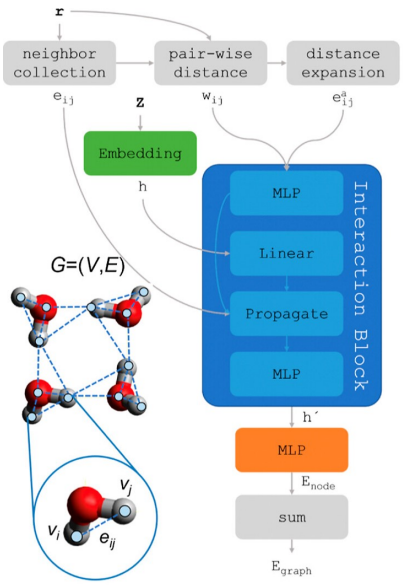


FIGURE 4. The architecture of the SchNet geometric NNP. Image courtesy of [14].

defined by $I - D^{-1/2}AD^{-1/2}$.[2] This choice of aggregation function has several notable advantages; firstly, the spectral (i.e., linear algebraic) properties of $\mathcal{L}$ (and hence $I - \mathcal{L}$) have been shown to be tightly related to variety of underlying combinatorial properties of the graph. See [4] and [8] for a survey of such results. Additionally, since the spectrum of the normalized Laplacian is entirely contained with in $[0, 2]$, the spectrum of $I - \mathcal{L}$ is contained in $[-1, 1]$. In particular, this means that applying $D^{-1/2}AD^{-1/2}$ to the state matrix $\mathbf{X}$ can not increase the overall size of the state matrix, that is, the Frobenious norm of $\mathbf{X}'$ is bounded by the Frobenious norm of $\mathbf{X}$.[3] Informally, the boundedness of the spectral norm of $D^{-1/2}AD^{-1/2}$ may help mitigate the possibility of exploding gradients in the training process by keeping the overall state space bounded. It is worth noting that the spectrally similar, "random-walk Laplacian" $D^{-1}A$ shares several of these advantages while also allowing an interpretation of the graph convolution as a diffusion process. While we are not aware of any results directly addressing this, we believe that $D^{-1/2}AD^{-1/2}$ is preferred over $D^{-1}A$ as the former belongs to the class of the Hermetian matrices which typically have much better numerical stability properties.

GNNs, and in particular graph convoluational layers, have prove to be extremely useful in a variety of neural network contexts. For example, the key feature of the SchNet, DimeNEt++, and GemNet architectures for neural network potential is the use of a graph convolutional layer based on an interaction defined by the locations of molecules. Similar approaches could be used in the context of the power-grid to solve AC Optimal Power Flow equation [9] by exploiting the under graphical structure of the grid [22]. However, despite their successes, there are significant limitations of GNNs and GCLs in particular. Specifically, in contrast to modern physics informed neural networks or neural operator networks, there are a limited methods for incorporate domain knowledge directly into the architecture or training. Indeed, currently the state-of-the-art methods for incorporating domain information into a graph convolutional layer is to use domain information to adjust the edge weights of the interaction graph to reflect domain knowledge. However, for many classes of domain information there is no obvious approach to encode this information into the interaction graph. For instance, for neural network potentials it would seem to be advantageous to directly encode the atom type in the graph convolutional layer as similar atoms should be have similarly electrically. But as various

[2]It is worth noting that other aggregation functions can also be seen as a closely related to other graph Laplacians such as the combinatorial Laplacian, $D - A$.

[3]This observation doesn't imply that the norm of the state associated with a particular vertex is not increasing, just that the sum of the squared-norms over all states is not increasing

atoms can be well-separated in the molecule there is no obvious way to introduce edge weights which capture this similarity.

Recently Aksoy and Young [1, 27] introduced a framework for constructing Laplacians on simplicial complexes (including graphs) which provides an alternative approach to incorporate domain knowledge into a graph convolutional layer. Although their approach generalizes to arbitrary simplicial complexes, we focus on the restriction to graphs (which are 1-dimensional simplicial complexes). In order to build the Laplacian defined Aksoy and Young on a graph $G = (V, E)$ we will require two positive definite similarity matrices M_V and M_E. As these matrices are positive definite, they may be thought of as defining an inner product space on the vector spaces defined by the vertices and edges of G. Because of this, the resulting Laplacian is refered to as an *inner product Laplacian*.

In order to define the inner product Laplacian, it is necessary to note that since M_V and M_E are positive definite there exists positive definite matrices Q_V and Q_E such that $Q_V^2 = Q_V^T Q_V = M_V$ and $Q_E^2 = Q_E^T Q_E = M_E$. The final ingredient in the definition of the inner product Laplacian is a edge-vertex incidence matrix B, defined so that $B_{ev} \in \{-1, 1\}$ if and only if $v \in e$ and is zero otherwise. The choice of signs for B is such that each row has precisely one $+1$ and one -1, thus B may be thought of as describing an arbitrary orientation of the graph G.[4] With these pieces, the inner product Laplacian is defined by Aksoy and Young as $\mathcal{L}^{IP} = Q_V^{-1} B^T M_E B Q_V^{-1}$. Much like the normalized and combinatorial Laplacian the spectral properties of the inner product Laplacian are closely tied to the combinatorial properties of the G, except now the "size" of the sets of edges and vertices are measured in terms of the inner product spaces defined by M_V and M_E. In particular, Aksoy and Young show that there are analogues of common isoperimetric results for the normalized and combinatorial Lapalcian, such as the Cheeger inequality and the expander mixing lemma [1]. Additionally, they show that the spectrum of the inner product Laplacian can be (approximately) bounded as a function of the maximum ratio between the size of the set of edges incident to a vertex and the size of the vertex (as measured by the respective inner product spaces). This observation will be helpful in Section 2.2 when attempting to design inner product spaces which perform well in graph convolutional neural networks.

2. IPLnet

In this section we detail our experimental setup for comparing the inner product graph convolutional layers with standard convolutional layers (as used in the SchNet architecture), specifying the neural network architecture in Section 2.1, the various inner products considered between the molecules and the edges in Section 2.2, the overall training framework in Section 2.3, the molecular data sets 2.4, and concluding with our results in Section 2.5.

2.1. **IPLnet Architecture.** IPLnet follows a similar architecture to that of SchNet architecture, depicted in Figure 4, but with the distance expansion replaced with $\mathcal{L}^{IP}$ and the message propagation layer replaced with a dense convolutional layer. The specific definitions of the vertex and edge inner products are given below. Throughout this work we compare the training of IPLnet with that of SchNet to examine how incorporation of $\mathcal{L}^{IP}$ affects training. Both models have comparable hyperparameters, and so we use the same values to maintain consistency: 6 interaction layers, 128 hidden channels and filters in the interaction layers, and a message size of 50. Edges are placed between nodes if the distance between the two atoms are within 10 Å. Despite using the same hyperparameters when defining the model architecture, IPLnet has 455,909 trainable parameters and SchNet has 594,149 trainable parameters.

2.2. **Molecular Inner Products.** Before describing the similarity matrices investigated in this work, we briefly highlight some of the challenges (and potential approaches) in defining a similarity matrices which are positive definite. Recall that a (real) matrix M is positive definite if it is symmetric and $x^T M x > 0$ for all non-zero vectors x. Alternatively, the condition $x^T M x > 0$ can be rephrased in terms of the strict positivity of all eigenvalues of M. However, both of these conditions are computationally challenging to verify accurately due to numerical issues, especially if M has eigenvalues that are close to 0. An alternative criteria for testing the positive definiteness of symmetric matrices is known as Sylvester's criterion, which

[4] It is worth noting that if M_E is not a diagonal matrix than the exact form (and the spectrum) of $\mathcal{L}^{IP}$ is dependent on the choice of orientation of G.

states that a symmetric matrix is positive definite if for all of the leading principle minors the determinant is positive. While this criteria is relatively easy to verify computationally[5], the challenge with Sylvester's criterion is that it doesn't provide insight on how to define the similarity between elements (i.e., the value of an off-diagonal entry in the similarity matrix) in a way that ensures that the resulting matrix satisfies Sylvester's criterion.

Thus, rather than relying an alternative necessary and sufficient criteria for positive definiteness of the matrices, we will limit the class of similarity matrices somewhat and only consider similarity matrices which can be shown to be positive definite using the Greshgorin disk theorem.

Greshgorin Disk Theorem. *Let $M \in \mathbb{C}^{n\times n}$ and let $r_i = \sum_{j\neq i} |M_{ij}|$. If D_i is closed disk in the complex plain of radius r_i and centered at M_{ii}, then the spectrum of M is contained in $\bigcup D_i$.*

Thus, in order to create a positive definite similarity matrix M it suffices to determine the off-diagonal entries and then fix $M_{ii} = \epsilon + \sum_{j\neq i} |M_{ij}|$ for some $\epsilon > 0$. In what follows, ϵ is chosen to be one for simplicity.

2.2.1. *Vertex Inner Product.* For the vertex inner product we encode the similarity based on the chemical properties of the various atoms in the network. This echoes the behavior of the SchNet, except that for SchNet the similarity encoding must be learned through an encoding process rather than being directly provided by subject matter expertise. More concretely, we specify one of three elemental properties: Pauling electronegativity, which describes the tendency of an atom to attract electrons; valency, which is the number of electrons an atom has in its outermost shell available for bonding; or block membership, which describes an atom's characteristic orbital based on its azimuthal quantum number. The choice of chemical property associates a value x_v to each vertex v which is the used to define the similarity of two distinct vertices u and v as $e^{-|x_u - x_v|}$. The diagonal entries are then determined by the Greshgorin disk theorem as discussed above, resulting in a matrix M. As a final step in determining M_V, a diagonal matrix N is chosen (based on the edge inner product and the observed graph) so that $M_V = N^{-1}MN^{-1}$ is such that the expected spectral radius of $\mathcal{L}^{IP}$ is 2.

2.2.2. *Edge Inner Product.* As the dihedral angles between pairs of molecules is known to have significant impact on the electrical potential (with certain dihedral angles being energetically preferred) we base the edge similarity measure on these angles. Recall that in the SchNet architecture (and hence in the IPLnet architecture) each atom in the molecule is equipped with a position in $\mathbb{R}^3$, denoted $p_1, \ldots, p_n$ and the there is an edge between i and j if $d(p_i, p_j) \leq c$ where c is a tuneable cut-off threshold. In order to calculate the similarity between two edges e and f we first define the dihedral angle, θ_{ef}, between the edges, and then define the similarity as a function of θ_{ef} and a tuneable scaling parameter s If the dihedral angle between two edges is undefined, then we will say the similarity of e and f is 0.

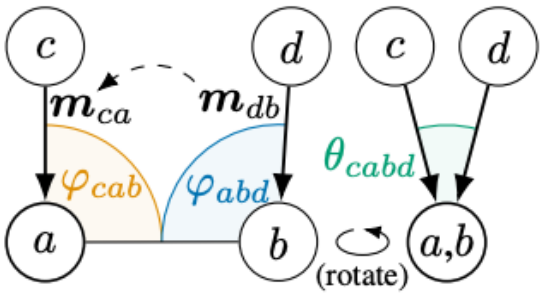


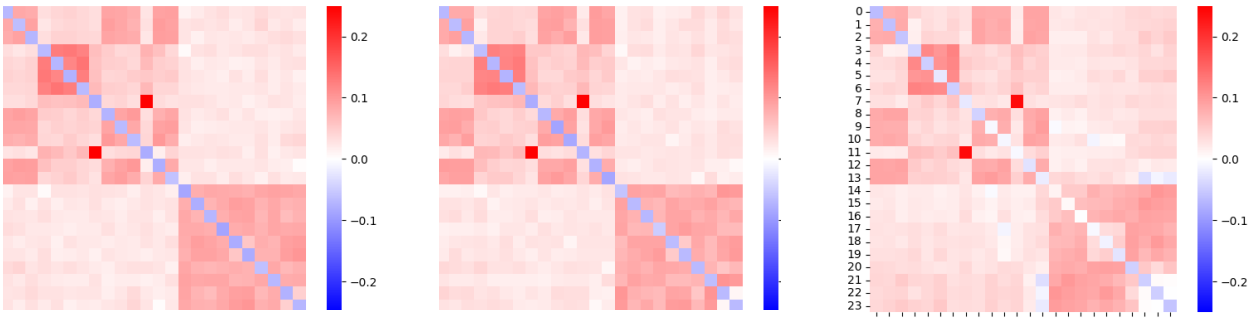
FIGURE 5. The dihedral angle between a quartet of atoms. The molecule is rotated so that the center atoms align, and the dihedral angle is computed. Image courtesy of [18].

The simplest case for the dihedral angle is the case where the vertices of e and f all lie in the same plane, or equivalently, $e \cap f = \{v\}$. In this case, we assume without loss of generality that $e = \{u, v\}$ and $f = \{v, w\}$ and define

$$\theta_{ef} = \arccos\left(\frac{(p_u - p_v)^T(p_w - p_v)}{\|p_u - p_v\|_2 \|p_u - p_w\|_2}\right),$$

that is, the angle between the edge e and f in the unique plane defined by p_u, p_v, p_w. In this case, $M_{ef} = \cos(s\theta_{ef})$.

[5]Numerical issues can still arise from small eigenvalues, however they are less prevalent as the determinant can be evaluated by using algorithms which are generally numerically stable such as Gaussian elimination. Additionally, there exists algorithms to evaluate the determinant which are division-free [3].



(A) Angle scaling factor of 1 (B) Angle scaling factor of 5 (C) Angle scaling factor of 10

FIGURE 7. Heatmap of the entries in convolution layer associated with the inner product Laplacian for a caffeine molecule.

In the case where all the end points do not lie in the same plane, we need first to define a reference plane to measure the dihedral angle in. To this end, suppose that $e = (u, v)$ and $f = (x, y)$ and that

$$d(p_v, p_x) = \min\left\{d(p_v, p_x), d(p_v, p_y), d(p_u, p_x), d(p_u, p_y)\right\}.$$

That is p_v and p_x are the closet pair of vertices with one belong to e and the other belonging to f. If $d(p_v, p_x) > c$, then we will say that the dihedral angle is undefined and the corresponding entry in M_E is 0, while if $d(p_v, p_x) \leq c$ we define θ_{ef} as the angle between the edge e and f when projected on to the plane normal to $p_v - p_x$, see Figure 5. More concretely, we define the normal vector $n = \frac{p_v - p_x}{\|p_v - p_x\|_2}$ and define $v_e = p_u - p_v - (p_u - p_v)^T n$ and $v_f = p_y - p_x - (p_y - p_x)^T n$. The dihedral angle between e and f is then

$$\theta_{ef} = \arccos\left(\frac{v_e^T v_f}{\|v_e\|_2 \|v_f\|_2}\right).$$

Finally, we define the similarity between e and f as $\left(1 - \frac{d(p_v, p_x)}{c}\right)\cos(s\theta_{ef})$. We note that the scaling factor in front of the similarity is chosen to ensure that the edge similarity is a continuous function of the vertex positions.[6] The diagonal entries of M_E are set to be one more than the absolutive value of the row-sum, ensuring that M_E is positive definite by the Greshgorin disk theorem.

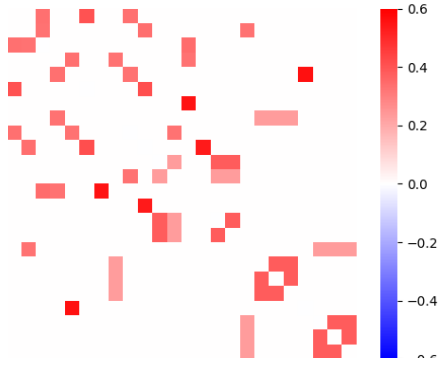


FIGURE 6. Heatmap of the entries in convolution layer associated with the normalized Laplacian for a caffeine molecule.

In order to illustrate the effect of the inner product Laplacian on the GNN, we present in Figures 6 and 7 heatmaps of the matrix associated for to the convoluational layer for a caffeine molecule ($C_8H_{10}N_4O_2$). In Figure 6, the heatmap depicts entries of $I - \mathcal{L}$ which are the entries corresponding to the convolutional layer associated with the caffeine bond graph. As we can see all entries of the convolutional layer are positive and there are relatively few entries. In Figure 7 we present the heatmaps associated with the convolutional layer for the inner product Laplacian ($I - \mathcal{L}^{IP}$) using electronegativity for the vertex inner product and three different scalings (1,5,10) of the dihedral angle for the edge inner product. In contrast to the normalized Laplacian convolutional layer, these matrices are significantly denser and have smaller values overall. While there is only minor variation readily visiable as the scaling value for the dihedral angle changes, we will see in Section 2.5 that changes in this scaling value have a significant effect on the performance of IPLnet.

[6]Note that any continuous function of $\frac{d(p_v, p_x)}{c}$ which is 0 when $d(p_v, p_x) = c$ and 1 when $d(p_v, p_x) = 0$ would suffice.

ID	Hyperparameter	Value
1	VIP feature	**valence**, electronegativity, block
2	Cutoff distance	**3**, 4, 5, 6, 10
3	Optimizer	**Adam**, AdamW, RMSprop
4	Learning rate	**1e-5**, 1e-4, 5e-4, 1e-3, 5e-3
5	Learning rate scheduler	**ReduceLROnPlateau**, StepLR
6	Scheduler patience epochs	0, **10**, 25
7	Learning rate factor	0.1, **0.4**, 0.8
8	Epochs	10, 20, **100**, 500, 1000
9	Batch size	32, **128**
10	Scaling factor of dihedral angle	1, 2, 3, 4, **5**, 6, 7, 8, 9, 10

TABLE 1. Optuna settings for Hyperparameters

Model	MAE (100 epochs)
IPLnet (Angle scaling factor of 1)	14.357485
IPLnet (Angle scaling factor of 5)	**7.231942**
IPLnet (Angle scaling factor of 10)	7.426400
SchNet	13.328036

TABLE 2. Performance of IPNnet with angle scaling factor of 1, 5, and 10

2.3. **Neural Network Training Framework.** We used the PyTorch Geometric (PyG) framework to load graph-based datasets and PyTorch to train the model with GPU accelerator. Optuna was used to search for optimal hyperparameters, while default settings were applied for all other training configurations, as detailed in Table 1. The models were trained on an NVIDIA A100 GPU with PyTorch 2.3, PyG 2.5.3, and Optuna 4.0.0. The Adam optimizer, with a learning rate of 1e-5, was selected to minimize MAE, and the ReduceLROnPlateau scheduler was used with a patience value of 10. A default batch size of 128 was used.

2.4. **Molecular Datasets.** In order to compare the performance of IPLNet and SchNet we trained both models on three different data sets of electrical potential for organic molecules. These standard data sets have varying number of molecules in different (generally static) conformations and the associated electrical potentials.

2.4.1. *QM9.* QM9 is a benchmark dataset composed of 134k organic small molecules made up of H, C, O, N, and/or F atoms that are minima on the potential energy surface (i.e., the forces on all atoms are 0) [23]. Along with atomic coordinates, scalar molecular properties, such as energy, enthalpy, and free energy of atomization, are given. All properties were computed by DFT (B3LYP/6-31G(2df,p)).

2.4.2. *MD17.* The MD17 dataset is composed of drug-like organic small molecules, such as benzene, toluene, naphthalene, ethanol, uracil, and aspirin, generated through ab initio molecular dynamics (AIMD) simulations [6]. A revised subset of the data was re-calculated with improved DFT parameters (PBE/def2-SVP level of theory) [7]. Because the structures were generated through AIMD, all are non-minima, meaning that non-zero forces are present on some or all atoms, and the same molecule is repeated with slightly different geometry and energy, in contrast to QM9 where each structure represents a unique molecule.

2.4.3. *ZINC.* The ZINC database was developed to aid virtual screening for drug discovery and contains structural properties of 728k drug-like small molecules [16]. The AGZ7 subset of the ZINC dataset includes optimized (minima) atomic coordinates and energies computed by DFT (B3LYP/cc-pVTZ) for a subset of 140k molecules [15]. Notably, ZINC and the AGZ7 subset contain a wider variety of atoms than QM9 or MD17: H, B, C, N, O, F, Si, S, P, Cl, Br, I, and Sn.

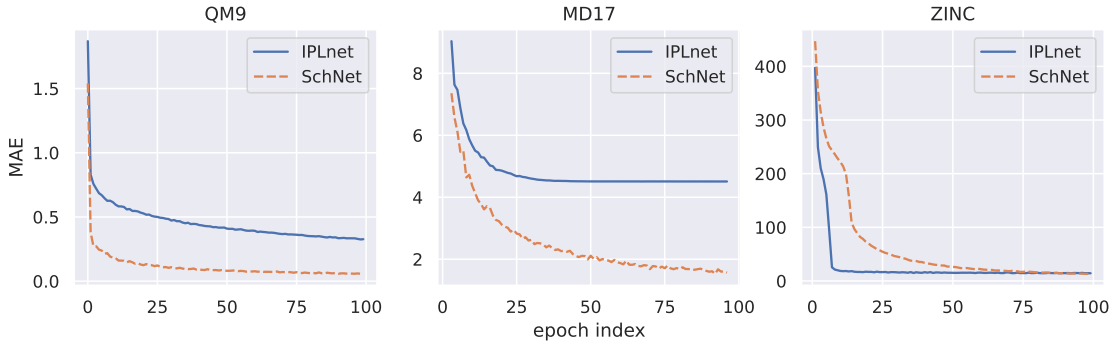


FIGURE 8. Comparison of IPLnet and SchNet on QM9, MD17, and ZINC datasets.

2.5. **Comparison of IPLnet and SchNet.** We conducted comparisons on several datasets, including QM9, MD17, and ZINC. The results show that IPLnet outperforms the baseline SchNet on the ZINC dataset using the angle scaling factor, with improvements observed from the first 100 epochs (which are summarized in Table 2 and Figure 9), while still delivering reasonable performance on the other two datasets. For the QM9 dataset, we compared IPLnet with SchNet for the internal energy property (`U_0`), showing a mean absolute error (MAE) of 0.32 after the first 100 epochs, compared to 0.057 MAE from SchNet. For the MD17 dataset, using a combined set of molecules from a revised subset of data, IPLnet achieved an MAE of 4.5 compared to SchNet's 1.32. In the ZINC dataset, the performance gap between the models was much narrower, with MAEs of 14.35 and 13.32, respectively. We applied different scaling factors to the ZINC dataset, which reduced the MAE by 7.12 and 6.85 when using scaling factors of 5 and 10, respectively. Notably, IPLnet outperformed SchNet by reducing the MAE by up to 45.7% when using these scaling factors. A summary of the comparison results is provided in Figure 8.

Taken together, these results illustrate the significant potential of the IPLNet architecture while highlighting the inherent challenges of this approach. In particular, we first observe that after the first 100 epochs, the MAE error is $\sim 6\times$ and $\sim 2\times$ worse in IPLNet as in SchNet, while with the AGZ7 subset of the ZINC database the best scaled IPLNet instance is has approximately half the MAE as SchNet. While it is challenging to precisely identify the root cause of the difference in the behavior, it is natural to suspect that the richer collection of atoms provides a significant advantage to IPLNet. In particular, QM9 has only 5 atom types while MD17 has only 4 atom types (with N only appearing in one molecule class), in contrast there are 13 different atoms represented in the AGZ7 subset. As a consequence, there is significant more domain information that SchNet has to recover from the data while with the appropriate choice of inner product this can be directly encoded into the IPLNet framework. Indeed, after approximately 10 iterations the MAE error IPLNet is over an order of magnitude smaller than in SchNet. However, this advantage does not hold up for large number of epochs, as after 500 epochs the MAE error for IPLNet is approximately twice that of SchNet, and at 1000 epochs the ratio increases to approximately 3. This change is almost entirely driven by the slow and steady decrease in MAE for SchNet compared to a long plateau in the performance of IPLNet. While we do not currently sufficiently have enough data to justify this conclusion, one may suspect the initial rapid decrease in error may be from the ability of IPLNet to directly exploit the domain knowledge through the explicit representation given by the inner products. In contrast, SchNet in some sense needs to build that representation before attempting the learn the mapping between molecules to energy potential. However, in SchNet as training progresses the representation of individual atoms can be further refined to the task while IPLNet it must be fixed at the beginning of training. This suggests that the behavior of IPLNet could be refined by instead of defining a single inner product for the vertices and edges, a parameterized family is defined that can be tuned to the particular data instance to improve performance. In fact, that is what we see with the edge inner product in Table 2 and Figure 9. The addition of the scaling parameter to the edge inner product allows two-fold decrease in the MAE error. This suggests that future development of IPLNet should focus on designing families of inner products which can be tuned to capture the relevant features for a given task.

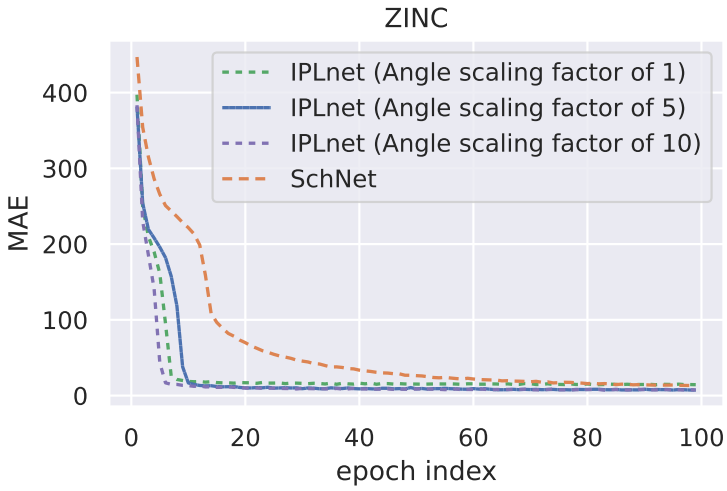


FIGURE 9. IPLnet Performance with angle scaling factors.

3. ENERGY-EFFECIENT IPL FOR IPLNET

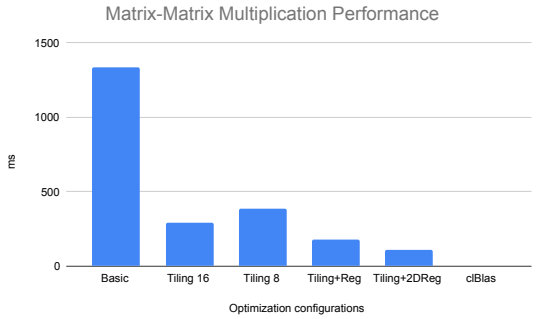


FIGURE 10. clBlas, the linear algebra library for OpenCL, shows significant performance advantage over the manual optimization efforts.

The primary objective of leveraging Graph Neural Networks (GNNs) and Graph Convolutional Neural Networks (GCNNs) to replace computationally intensive first-principles chemistry methods is to expedite scientific discoveries while maintaining satisfactory accuracy and fidelity. In parallel to accelerating chemistry methodologies through AI/ML surrogates, further enhancements can be achieved by optimizing the training and inference processes of GNNs and GCNNs through hardware advancements, algorithmic improvements, and compiler optimizations. This section explores these opportunities in the context of accelerating the Inner Product Laplacian (IPL) model and discusses the lessons learned.

3.1. **IPL Acceleration.** This section outlines our efforts to accelerate the Inner Product Laplacian (IPL) model. We explored multiple approaches, including: (1) Hardware acceleration on GPUs, (2) Algorithmic improvements, and (3) Compiler optimizations.

3.1.1. *Hardware Acceleration.* The majority of the performance gains observed in both scientific and AI domains over the last decade have been driven by offloading key computational tasks to GPUs. Given that most modern supercomputers (such as those powered by NVIDIA, AMD, and Intel GPUs) utilize GPUs as their primary computational resources, it is a logical progression to explore GPU acceleration for IPL. Specifically, we aim to optimize the evaluation of the expression:

$$\mathbf{X}' = (I - \mathcal{L}^{IP})\mathbf{X}\mathbf{W}^T.$$

Two distinct strategies can be employed to achieve this optimization. One option is to develop a custom GPU kernel using CUDA or OpenCL for the entire expression. Alternatively, we can develop linear algebra primitives that interconnect in a dataflow manner. We selected the latter approach, as it allows these primitives to be reused for other problems and also facilitates their application to non-GPU accelerators, such

as FPGAs. The key linear algebra operations to develop include matrix-matrix multiplication, transposition, and matrix inversion, all of which are well-studied with several existing implementations.

To leverage the diverse capabilities of various GPU architectures, we conducted several studies aimed at understanding the impact of specific optimization techniques on the performance of matrix-matrix multiplication. Following these investigations, we compared the results with the performance metrics of commonly used linear algebra libraries. In Figure 10, our findings with clBlas on AMD GPUs are presented. This OpenCL-based library for General Matrix Multiply (GEMM) operations is relatively less explored compared to cuBLAS, the CUDA-based linear algebra acceleration library for NVIDIA GPUs. We meticulously applied several optimization techniques to matrix-matrix multiplication, including tiling for local memory with varying tile sizes, one-dimensional and two-dimensional register blocking, among other methods. Our results indicate that the clBlas library consistently achieves the shortest time-to-solution, underscoring the efficiency of such libraries.

3.1.2. *Algorithmic Improvements.* As previously detailed, the adjacency matrix A is too small to yield performance enhancements on GPUs; the overhead of data transfer to and from the GPU device outweighs the benefits of accelerated computation. However, it is feasible to amalgamate different input matrices into a larger matrix, as illustrated in Figure 11. The resultant matrix is constructed by the juxtaposition of all the adjacency matrices along the diagonal, each representing a distinct problem. A similar approach is employed for solving power grid problems, where matrices representing various scenarios are arranged along the diagonal. As depicted in the figure, the resulting matrix exhibits a block-sparse format. Additionally, each block (i.e., the original adjacency matrix) displays certain sparsity patterns. Nonetheless, as discussed in prior sections, the matrix's size remains insufficient to leverage the advantages of sparse computation.

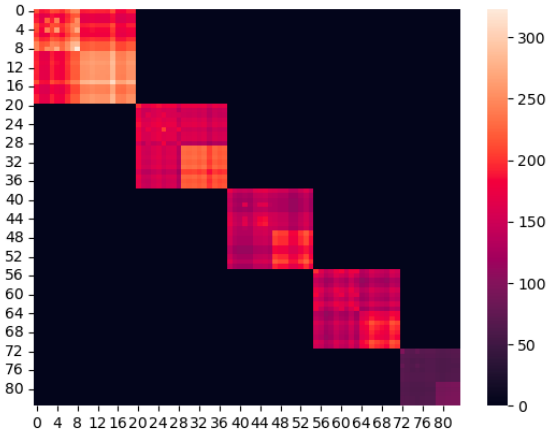


FIGURE 11. By combining multiple adjacency matrices into a larger one, a larger block-sparse matrix is constructed to reduce the relative overhead of data transfer to GPUs.

Figure 12 illustrates that by amalgamating a sufficient number of small matrices into a larger block-sparse matrix and employing sparse linear algebra implementations, significant speedups over dense representations and dense linear algebra operators with the same inputs can be achieved.

Furthermore, a performance comparison of matrix multiplication using different sparse matrix formats is presented. As depicted in Figure 13, the Blocked-ELL format consistently outperforms the CSR format in matrices with hardware-friendly choices of block sizes. This highlights a future challenge of determining the optimal block size based on aggregated graph data and specific hardware characteristics.

3.1.3. *Compiler Optimizations.* The fusion of operators and the implementation of inter-operator optimizations represent promising approaches for reducing the number of temporary variables, minimizing arithmetic operations, and enhancing data locality. The concepts of loop and kernel fusion have been extensively explored in the literature. In this project, we applied kernel fusion for operations involving sparse-matrix-dense matrix multiplication followed by dense-matrix-dense matrix multiplication, specifically,

$$\mathbf{X}' = ((I - \mathcal{L}^{IP})\mathbf{X})\mathbf{W}^T.$$

The COMET compiler incorporates optimizations for matrices stored in compressed-row format (CSR). However, the original adjacency matrices are too small to benefit significantly from such optimizations (since kernel fusion introduces additional complexity, there is a trade-off between the additional code complexity and the reduction of temporaries and arithmetic operations) or from sparse computations in general. Fortunately, the COMET team has been developing support for an adaptive block-sparse storage format, where the blocks

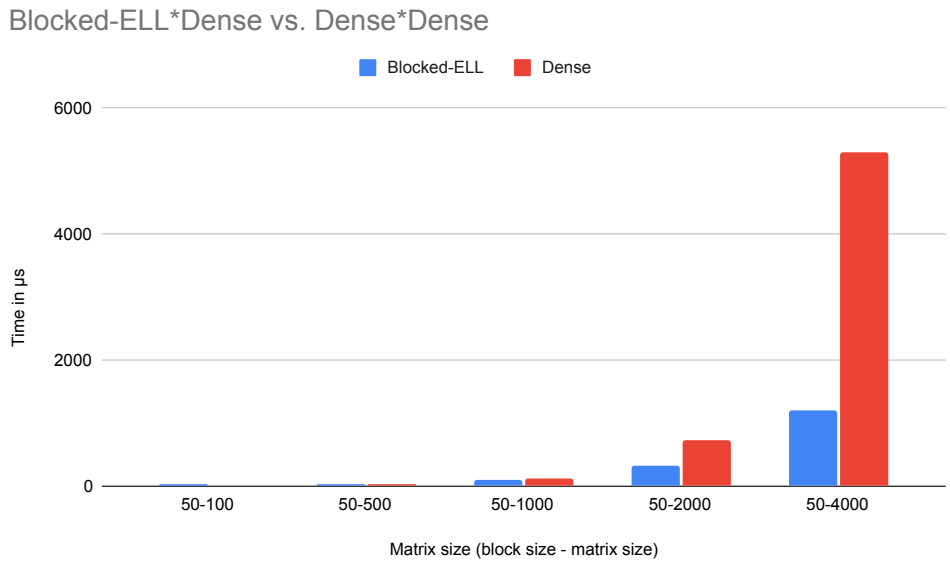


FIGURE 12. Execution time of a SpMM kernel on GPUs as a function of the size of the block-sparse matrix.

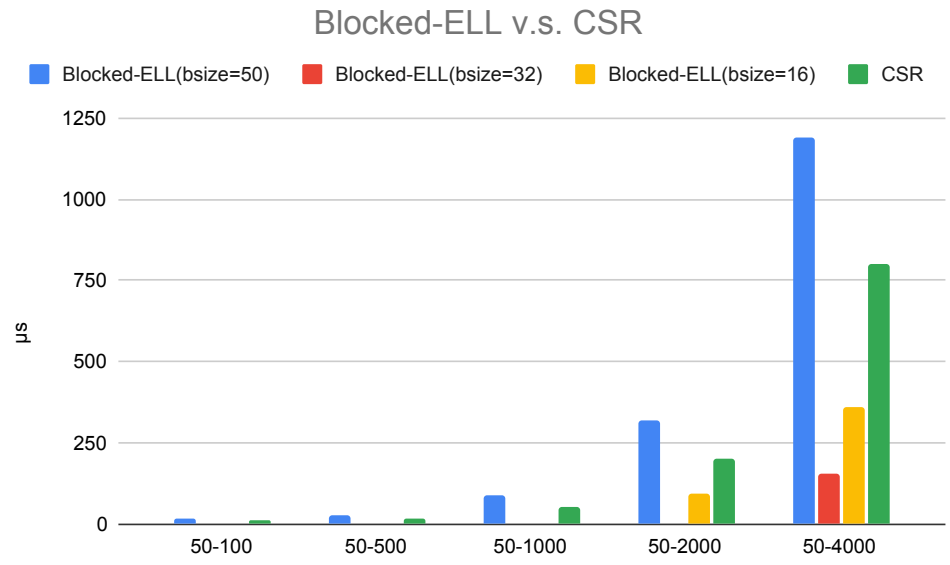


FIGURE 13. Performance comparison of various sparse storage formats.

of the block-sparse matrix are not required to have uniform shapes, and each block can employ a different sparse storage format to leverage the specific sparsity patterns of the data.[7] The exploration of this new feature is left as future work.

[7]This work is currently under submission.

```python
import numpy as np

N = 1024
M = 1024
K = 4096
A = np.ones((M,K), dtype=np.float32)
B = np.ones((K,N), dtype=np.float32)
C = np.zeros((M,N), dtype=np.float32)
Cref = np.dot(A,B)
```

(A) NumPY

```python
import numpy as np
import PyMCL as mcl
import PyMCL.linalg as la
N = 1024
M = 1024
K = 4096
A = np.ones((M,K), dtype=np.float32)
B = np.ones((K,N), dtype=np.float32)
C = np.zeros((M,N), dtype=np.float32)
Cref = la.dot(A,B)
```

(B) PyMCL

FIGURE 14. Standard NumPy code vs PyMCL. Note that, although the code appears very similar, with the only evident difference of using PyMCL operators instead of NumPy operators, under the hood, the MCL runtime can decide to execute the computation on GPUs, FPGAs, or any other available device.

3.2. **PyMCL.** In our experience with the PNNL MARS initiative, we have observed that integrating emerging technologies, such as FPGAs and data-flow architectures, as well as multi-GPU systems, into complex codebases can be challenging. Most system software and programming models developed for these emerging technologies are based on C/C++ or proprietary solutions (e.g., CUDA, SambaFlow, Xilinx HLS) provided by individual vendors. Conversely, modern code authored by computational scientists, particularly in the fields of AI and ML, predominantly utilizes Python (e.g., SchNet).

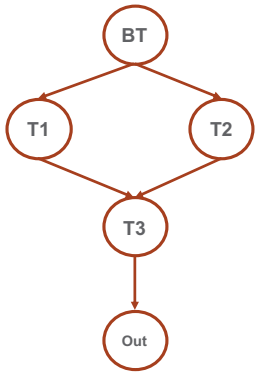


FIGURE 15. Computation directed acyclic graph (DAG) for the PyMCL computation described in Figure 16b. Each node on the DAG represents a computation task, while each edge represents a data dependency.

To bridge this gap and facilitate the integration of emerging technologies and custom accelerators within Python code, we developed PyMCL. PyMCL is a Python package designed to serve as a drop-in replacement for NumPy, enabling the execution of NumPy operations on hardware accelerators. PyMCL is built on the Minos Computing Library (MCL) developed at PNNL, which is an asynchronous task-based system software and programming model for highly heterogeneous systems. Users of PyMCL do not need to interact directly with MCL; instead, they can rely on the familiar NumPy interface while providing additional hints to the MCL runtime if necessary.

Figures 14a and 14b present equivalent code written in standard NumPy and PyMCL, respectively. The main code in both examples appears very similar, with the only significant difference being the use of the `.dot()` method from `PyMCL.linalg` instead of the standard `numpy` package. However, utilizing `PyMCL.linalg.dot()` results in the initiation of an asynchronous MCL task that can be executed on various computing devices, including GPUs, FPGAs, and data-flow accelerators.[8]

Using PyMCL, the code can be rewritten using `mcl.linalg` primitives, as shown in Figure 16b.[9] While the code is relatively straightforward, it conceals two important aspects: first, the algebraic primitives can

[8]The specific device on which the task will execute is determined at runtime by the MCL scheduler, based on the implemented scheduling policy (e.g., locality-aware, power-saving, maximum throughput, balanced, etc.). However, users can specify a class of devices for the MCL to execute the task by using the `dev` parameter. For instance, to execute the matrix multiplication in Figure 14b on a GPU-class device, the code can be modified with the statement: `Cref = la.dot(A, B, dev = mcl.TASK_GPU)`.

[9]Currently, PyMCL does not support custom Python operators, only library-based calls.

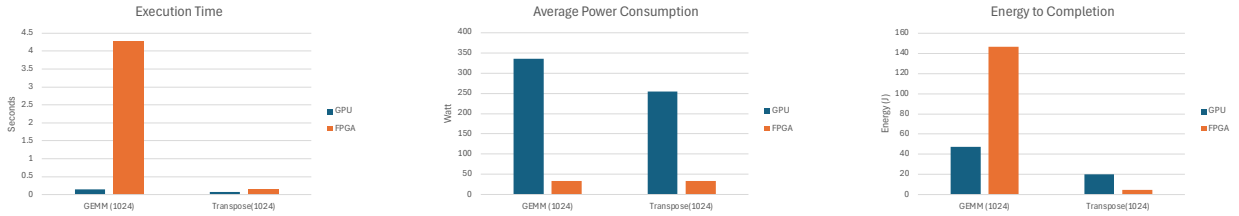
```
out = Q_Vinv @ B @ edge_ip
        @ np.transpose(B) @ Q_Vinv
```

(A) IPL implementation using standard NumPy

```
BT = la.transpose(B, sync=True)
T1 = la.dot(Q_Vinv, B)
T2 = la.dot(edge_ip, BT)
mcl.wait_all()
T3 = la.dot(T1,T2, sync=True)
out = la.dot(T3, Q_Vinv)
```

(B) IPL implementation with PyMCL

FIGURE 16. IPL implementation using standard NumPy and PyMCL.



(A) Execution time. (B) Average power consumption. (C) Total energy to completion.

FIGURE 17. Performance, power, and energy-to-completion comparisons between AMD GPU and Xilinx FPGA for the GEMM and Transpose kernels used in the IPL computation. GPUs are generally highly optimized for GEMM-like computation, yielding superior performance and energy efficiency, albeit with high average power consumption. FPGAs outperform GPUs in terms of energy efficiency for irregular computations. Despite the lower performance of the transpose kernel on FPGA compared to GPU, its lower average power consumption results in greater energy efficiency.

be executed on heterogeneous devices; second, tasks without dependencies can be executed in parallel. For instance, tasks `T1` and `T2` can be executed concurrently. Data dependencies are specified either through the `sync` parameter in library calls or by enforcing a barrier with `mcl.wait()` or `mcl.wait_all()`. Figure 15 illustrates the directed acyclic graph (DAG) for the IPL computation depicted in Figure 16b.

3.3. **Energy Efficiency.** As previously discussed, the input to the IPL computation may be too small to benefit significantly from heterogeneous resources that are "far away" on the I/O bus, such as GPUs connected to the host through the PCIe bus, since the overhead of data transfer outweighs the advantages of accelerated computation. Instead of exclusively targeting performance improvements, we investigated opportunities to enhance energy efficiency. This is a critical consideration: according to current trends in the size and computational and memory requirements of contemporary AI/ML models, progress will stagnate unless we significantly reduce the energy cost of training and inferencing on these models.

3.3.1. *Analysis of Energy-to-Completion.* We compared the performance, power consumption, and energy-to-completion of two fundamental kernels in the IPL computation — the matrix-matrix multiplication and the matrix transpose — on AMD GPUs and Xilinx FPGAs, and present the results in Figure 17. As Figure 17a demonstrates, the GPU outperforms the FPGA in terms of execution time. For matrix-multiplication kernels (GEMM), the speedup is significant: GPUs have been extensively optimized to perform GEMM kernels efficiently over the years, which is a key factor in recent advancements in AI/ML. Despite the GPU consuming much more power (see Figure 17b), the energy-to-completion is still lower than that of the FPGA due to the GPU's rapid execution (see Figure 17c). The transpose kernel also performs better on the GPU than on the FPGA, though the gap is not as wide as for the GEMM kernel. Transposes involve more irregular/large stride memory accesses compared to GEMM, and GPUs are not as well optimized for these patterns. In contrast, FPGAs allow for custom programming of memory access patterns, resulting in better performance. Although the FPGA's execution of the transpose kernel is slower than the GPU's, the margin is narrower than

```
. . .
mcl_start = time.time()
out = roc.ilp(Q_Vinv, edgp_ip, B, dev=mcl.TASK_ILP)
mcl.wait_all()
. . .
```

FIGURE 19. PyMCL implementation of the IPL product using the IPL processor. MCL recognizes a new type of processor with distinct inputs and outputs, as opposed to a conventional FPGA implementing various kernels, resulting in even simpler code than the previous version.

for GEMM. However, the FPGA's significantly lower average power consumption results in a lower overall energy-to-completion compared to the GPU. These results suggest that to achieve lower energy consumption while maintaining acceptable performance, one could map the GEMM kernel to the GPU and the transpose kernel to the FPGA. These scenarios are precisely what MCL is designed for. Moreover, the MCL runtime optimizes data movement among the CPU, GPU, and FPGA. Thanks to PyMCL, this feature is available to domain scientists working in Python environments, and it is what we employed in this project.

3.3.2. *The custom IPL processor.* FPGAs promise higher energy efficiency compared to GPUs, albeit at the cost of reduced performance. However, FPGAs are fully programmable, allowing for the design of a custom processor for the IPL based on the algebraic primitives explored in the previous section. The primary objective of designing an IPL processor is to minimize data movement across the PCIe bus and leverage the dataflow capabilities of FPGAs. We designed and synthesized such a processor on an FPGA based on the fundamental kernel primitives of the IPL. Figure 18 illustrates the interconnection of blocks and the data flow from one block to another. Within the FPGA, data flow between blocks, with only the input and final output being transferred to/from the host, thereby reducing data movement to/from the host.

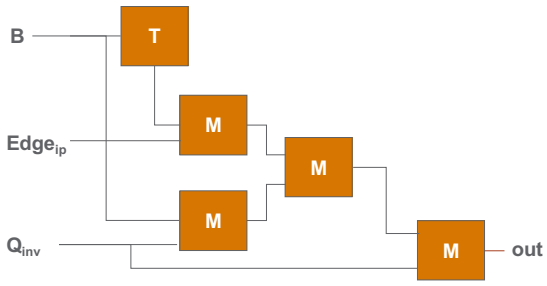


FIGURE 18. Block diagram representing the interconnection of algebraic primitives in the IPL processor implementation on FPGA.

When utilizing the custom IPL processor we designed, users interact directly with the custom processor rather than managing the execution and data dependencies of individual kernels within the IPL processors, as illustrated in Figure 19. The resulting code is even simpler than before, as shown in Figure 16b.

References

[1] S. G. Aksoy and S. J. Young, *Inner product laplacians*. `in preperation`.

[2] N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, K. Willcox, and S. Lee, *Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence*.

[3] R. S. Bird, *A simple division-free algorithm for computing determinants*, Information Processing Letters, 111 (2011), pp. 1072–1074.

[4] A. E. Brouwer and W. H. Haemers, *Spectra of graphs*, Springer Science & Business Media, 2011.

[5] J. Carter, J. Feddema, D. Kothe, R. Neely, J. Pruet, R. Stevens, P. Balaprakash, P. Beckman, I. Foster, K. Iskra, A. Ramanathan, V. Taylor, R. Thakur, D. Agarwal, S. Crivelli, B. de Jong, D. Rouson, M. Sohn, M. Wetter, S. Wild, T. Bremer, M. Goldman, A. Kupresanin, L. Peterson, B. Spears, D. Stevens, B. Van Essen, R. Bent, M. Grosskopf, E. Lawrence, G. Shipman, K. Rose, R. Grout, N. Kouakpaizan, F. Omitaomu, S. Peles, P. Ramuhalli, A. Shankar, D. Womble, G. Zhang, T. Catanach, R. Oldfield, S. Rajamanickam, J. Ray, M. A. Leung, C. Catlett, and E. M. Dietrich, *Advanced research directions on ai for science, energy, and security: Report on summer 2022 workshops*.

[6] S. Chmiela, A. Tkatchenko, H. E. Sauceda, I. Poltavsky, K. T. Schütt, and K.-R. Müller, *Machine learning of accurate energy-conserving molecular force fields*, Science advances, 3 (2017), p. e1603015.

[7] A. S. Christensen and O. A. Von Lilienfeld, *On the role of gradients for machine learning of molecular energies and forces*, Machine Learning: Science and Technology, 1 (2020), p. 045018.

[8] F. R. Chung, *Spectral graph theory*, vol. 92, American Mathematical Soc., 1997.

[9] W. Dong, Z. Xie, G. Kestor, and D. Li, *Smart-pgsim: Using neural network to accelerate ac-opf power grid simulation*, in SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, 2020, pp. 1–15.

[10] T. T. Duignan, *The potential of neural network potentials*, ACS Physical Chemistry Au, 4 (2024), pp. 232–241.

[11] A. Duval, S. V. Mathis, C. K. Joshi, V. Schmidt, S. Miret, F. D. Malliaros, T. Cohen, P. Lio, Y. Bengio, and M. Bronstein, *A hitchhiker's guide to geometric gnns for 3d atomic systems*, arXiv preprint arXiv:2312.07511, (2023).

[12] J. Gasteiger, F. Becker, and S. Günnemann, *Gemnet: Universal directional graph neural networks for molecules*, Advances in Neural Information Processing Systems, 34 (2021), pp. 6790–6802.

[13] J. Gasteiger, J. Gross, and S. Günnemann, *Directional message passing for molecular graphs*, arXiv preprint arXiv:2003.03123, (2020).

[14] H. Helal, J. Firoz, J. A. Bilbrey, H. Sprueill, K. M. Herman, M. M. Krell, T. Murray, M. L. Roldan, M. Kraus, A. Li, et al., *Acceleration of graph neural network-based prediction models in chemistry via co-design optimization on intelligence processing units*, Journal of Chemical Information and Modeling, 64 (2024), pp. 1568–1580.

[15] B. Huang and O. A. von Lilienfeld, *Dictionary of 140k gdb and zinc derived amons*, arXiv preprint arXiv:2008.05260, (2020).

[16] J. J. Irwin and B. K. Shoichet, *Zinc- a free database of commercially available compounds for virtual screening*, Journal of chemical information and modeling, 45 (2005), pp. 177–182.

[17] S. Käser, L. I. Vazquez-Salazar, M. Meuwly, and K. Töpfer, *Neural network potentials for chemistry: concepts, applications and prospects*, Digital Discovery, 2 (2023), pp. 28–58.

[18] J. Klicpera, F. Becker, and S. Günnemann, *Gemnet: Universal directional graph neural networks for molecules*, in Proceedings of the 35th International Conference on Neural Information Processing Systems, 2021, pp. 6790–6802.

[19] E. Kocer, T. W. Ko, and J. Behler, *Neural network potentials: A concise overview of methods*, Annual review of physical chemistry, 73 (2022), pp. 163–186.

[20] M. Labonne, *Graph convolutional networks: Introduction to gnns*. Towards Data Science, 2023.

[21] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, *A survey of convolutional neural networks: analysis, applications, and prospects*, IEEE transactions on neural networks and learning systems, 33 (2021), pp. 6999–7019.

[22] D. Owerko, F. Gama, and A. Ribeiro, *Unsupervised optimal power flow using graph neural networks*, in ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2024, pp. 6885–6889.

[23] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. Von Lilienfeld, *Quantum chemistry structures and properties of 134 kilo molecules*, Scientific data, 1 (2014), pp. 1–7.

[24] K. T. Schütt, H. E. Sauceda, P.-J. Kindermans, A. Tkatchenko, and K.-R. Müller, *Schnet–a deep learning architecture for molecules and materials*, The Journal of Chemical Physics, 148 (2018).

[25] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, *A comprehensive survey on graph neural networks*, IEEE Transactions on Neural Networks and Learning Systems, 32 (2021), pp. 4–24.

[26] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, *Convolutional neural networks: an overview and application in radiology*, Insights into imaging, 9 (2018), pp. 611–629.

[27] S. J. Young, *Reimagning spectral graph theory*. `https://www.youtube.com/watch?v=gWVfGibrgps`, November 2023. Texas A&M Institute of Data Science Seminar.

Pacific Northwest National Laboratory

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99354

1-888-375-PNNL (7665)

www.pnnl.gov