JCIM JOURNAL OF CHEMICAL INFORMATION AND MODELING

pubs.acs.org/jcim Article

Acceleration of Graph Neural Network-Based Prediction Models in Chemistry via Co-Design Optimization on Intelligence Processing Units

Hatem Helal, Jesun Firoz, Jenna A. Bilbrey, Henry Sprueill, Kristina M. Herman, Mario Michael Krell, Tom Murray, Manuel Lopez Roldan, Mike Kraus, Ang Li, Payel Das, Sotiris S. Xantheas,* and Sutanay Choudhury*

Cite This: https://doi.org/10.1021/acs.jcim.3c01312 Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information

ABSTRACT: Atomic structure prediction and associated property calculations are the bedrock of chemical physics. Since high-fidelity ab initio modeling techniques for computing the structure and properties can be prohibitively expensive, this motivates the development of machine-learning (ML) models that make these predictions more efficiently. Training graph neural networks over large atomistic databases introduces unique computational challenges, such as the need to process millions of small graphs with variable size and support communication patterns that are distinct from learning over large graphs, such as social networks. We demonstrate a novel hardware−software codesign approach to scale up the training of atomistic graph neural networks (GNN) for structure and property prediction. First, to eliminate redundant computation and memory associated with alternative padding techniques and to improve throughput via minimizing communication, we formulate the effective coalescing of the batches of variable-size atomistic graphs as the bin packing problem and introduce a hardware-agnostic algorithm to pack these batches. In addition, we propose hardware-specific optimizations, including a planner and vectorization for the gather-scatter operations targeted for Graphcore's Intelligence Processing Unit (IPU), as well as model-specific optimizations such as merged communication collectives and optimized softplus. Putting these all together, we demonstrate the effectiveness of the proposed codesign approach by providing an implementation of a well-established atomistic GNN on the Graphcore IPUs. We evaluate the training performance on multiple atomistic graph databases with varying degrees of graph counts, sizes, and sparsity. We demonstrate that such a codesign approach can reduce the training time of atomistic GNNs and can improve their performance by up to 1.5× compared to the baseline implementation of the model on the IPUs. Additionally, we compare our IPU implementation with a Nvidia GPU-based implementation and show that our atomistic GNN implementation on the IPUs can run 1.8× faster on average compared to the execution time on the GPUs.

1. INTRODUCTION

Recent advancements in the field of artificial intelligence (AI) have fueled notable progress in data-driven scientific discovery.[1−5] Neural networks (NNs) provide particular advantages as surrogate models for modeling quantum chemical properties.[6] High-accuracy ab initio methods, which contain no empirical fitting parameters, are prohibitively expensive, with computational costs that can scale as high as $O(N^9)$, where N is the number of atoms in the system.[7] NNs have been shown to be able to reach the same level of accuracy as the ab initio method on which the NN was trained, but with roughly $O(N)$ scaling.[8−10] Atomistic geometries are naturally represented as graphs, which makes Graph Neural Networks (GNNs)[11−13] compelling for building surrogate models to train results obtained by quantum chemistry methods. Initially formulated as a message passing neural network,[14] support of relational geometry and atom-level attributes, such as atom type, has become a key feature of such models.[15] The past few years have witnessed further expansion of this class of models, henceforth referred to as atomistic GNNs, through sophisticated message passing,[16] accounting for multiscale structure in the network,[17] adoption of self-supervised learning,[18,19] and modeling interactions between geometric tensors.[20]

The fast training of these models becomes an increasingly important issue as the scientific community continues to develop massive atomistic databases and embrace GNNs.[21−23] Evidence from the literature suggests that GNNs are expensive

Received: August 16, 2023
Revised: February 2, 2024
Accepted: February 2, 2024

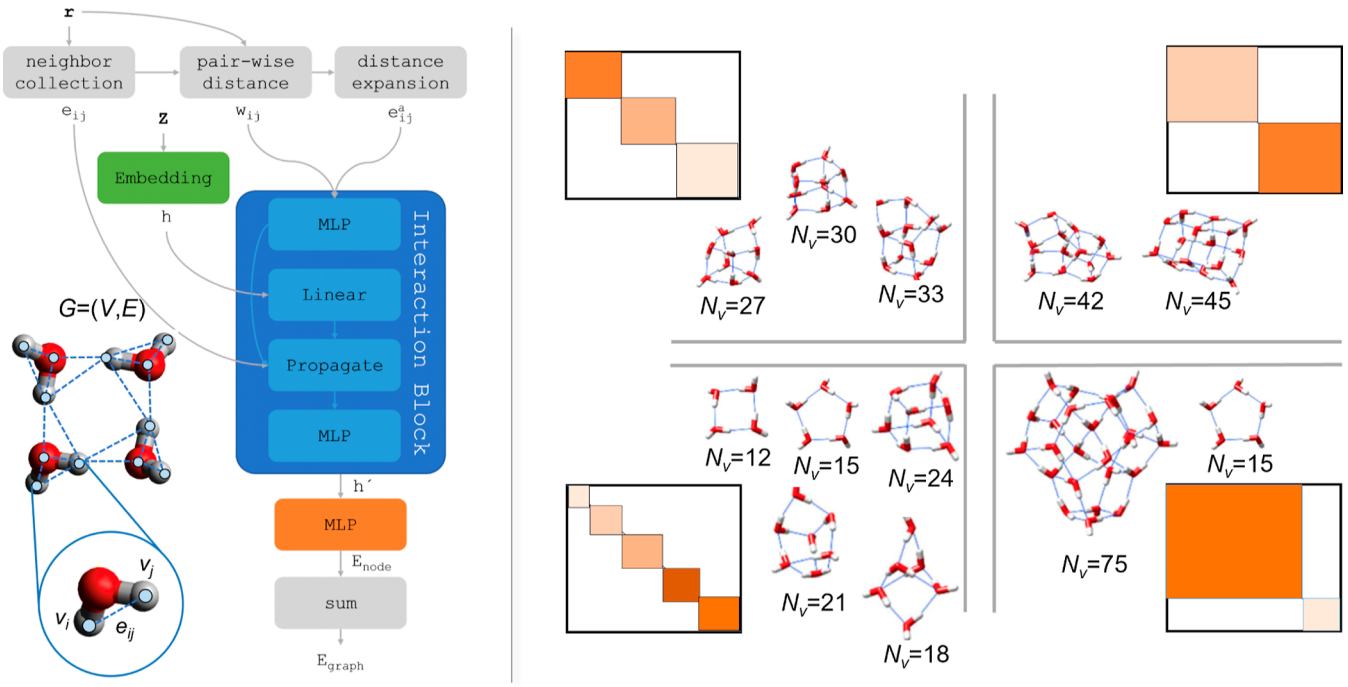


Figure 1. (Left) Illustration of the GNN-based implementation of SchNet. (Right) Visualization of the packing of atomistic graphs, indicating the number of vertices (N_v) in each graph. Processing them requires support for efficient sparse block matrix operations; communications are quite localized.

to train, often spanning days even when scaled over multiple Graphic Processing Units (GPUs) or Tensor Processing Units (TPUs).[18,23,24] However, most of the previous efforts on scaling the training of GNNs have focused on learning over a single massive graph,[25−27] which subsequently steers the design space of software and hardware solutions[28−30] to addressing traditional graph processing challenges such as graph partitioning,[31] irregular memory access,[32] and workload imbalance due to power-law degree distributions.[33] In contrast to training a GNN on a single large graph, atomistic GNNs are trained on an ensemble of small graphs with varying vertex counts. Atomistic graph databases[34−37] are characterized by the comparatively small size of individual graphs (both in terms of the total vertex count and the edge count) and a wide distribution of size and sparsity. Additionally, most of these data sets represent graphs where each vertex is also associated with a 3D coordinate representing the position of an atom, which subsequently requires support for geometric operators such as nearest-neighbor computations.

In a first effort of its kind, we present a hardware−software codesign approach to accelerate the training of message passing GNNs for property prediction that supports data sets composed of millions of small graphs of diverse sizes. In recent years, we have also witnessed the emergence of machine-learning accelerators such as TPUs,[38] reconfigurable data-flow,[39] Graphcore's Intelligence Processing Units (IPUs),[40] etc. The large on-chip high-bandwidth memory, support for fine-grained parallelism, and faster all-to-all communication links make IPUs attractive for atomistic GNN architectures, and as such, they are the primary target of this work.

In addition to focusing on a specific architecture, we also identified a model that can serve as a representative workload. In this study, we focus on the SchNet neural network architecture as an exemplar GNN.[15,41] Given a set of atomic coordinates, SchNet constructs a nearest-neighbor graph around each atom within a radial cutoff distance (Figure 1). Next, the network learns the representation of each atom, accounting for pairwise interactions between atoms based on a number of specified hops, by applying a GNN (interaction block in Figure 1). Finally, a graph-level property, such as the potential energy, is obtained by learning a composition function over each atom embedding. SchNet and its variants have been extensively used for property prediction in diverse domains such as computational chemistry,[23] drug discovery,[42] and materials science.[43] Its pragmatic relevance and its capture of key kernels, which are present in nearly all atomistic GNNs[16,19,20] make SchNet a compelling representative for atomistic GNN-driven codesign.

Though we focus on the SchNet message passing architecture in this paper, many of the advancements detailed herein also apply to more recently published atomistic GNNs. Of particular interest are equivariant message passing GNNs, such as NequIP[20] and MACE.[44] Performance improvements in such models come from the enhanced representation of atomic environments by performing equivariant convolutions over geometric tensors rather than invariant convolutions over scalars. Notably, a SchNet-like model is obtained by omitting higher-order interactions. Therefore, SchNet represents a baseline atomistic GNN useful for the examination of methods for algorithmic acceleration.

To this end, we present a comprehensive performance study using the QM9 small-molecule benchmark data set[45] as well as a recently released water cluster benchmark data set,[35,46] which notably has a high degree of size and sparsity imbalance (Section 2). We next detail the IPU hardware and software (Section 3). Co-design details are discussed in Section 4. In particular, we demonstrate a hardware-agnostic approach, namely batch packing to coalesce small graphs while tracking their original connectivity structure for graph-level property prediction (Section 4.1). We show that batch packing

https://doi.org/10.1021/acs.jcim.3c01312
J. Chem. Inf. Model. XXXX, XXX, XXX−XXX

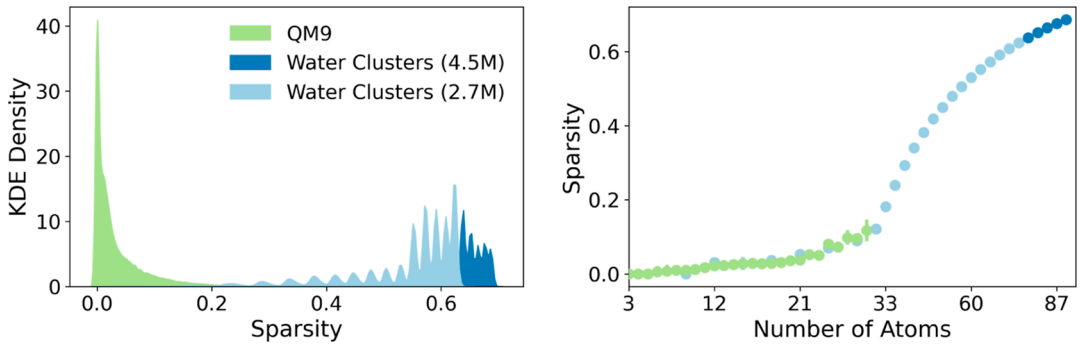


Figure 2. Sparsity of the QM9 and water cluster benchmark data sets plotted by the kernel density estimate (KDE) and against the number of vertices (atoms) per graph (markers represent the mean value with error bars given for 1σ). The water cluster set is split into two sets containing 4.5 and 2.7 M clusters; note that the 4.5 M set contains all clusters in the 2.7 M set and thus overlaps in the plots.

significantly reduces the memory footprint and improves GNN performance for databases with varying size and sparsity by gaining a geometric mean speedup of 1.25× compared to the original padding-based implementation. We also implement an IPU hardware-targeted scatter-gather planner for efficient scheduling of gather and scatter operators (Section 4.2). The planner finds the ideal trade-off between parallelism and device utilization. Additionally, we demonstrate the impact of optimizations such as asynchronous, nonblocking I/O for batch loading, vectorization of scatter operations, and prefetching for GNN workloads. We also propose model-specific optimizations such as merged communication collectives and optimized softplus activation functions (Section 4.3). Section 5 details a comprehensive evaluation of the optimized techniques on the benchmark data sets. Applying all of these techniques results in a speedup of up to 1.5× compared to the baseline IPU implementation. Our effective hardware−software codesign results in a performance improvement of the training of GNNs for atomistic graphs by on average 1.8× on a Bow IPU POD (16 IPUs) compared to a baseline GPU implementation running on an equivalent Nvidia A100 hardware (8 GPUs). Finally, we provide a demonstration of transfer learning across heterogeneous accelerators to illustrate the benefit of faster pretraining of atomistic GNNs on large data sets (Section 6). While transfer learning methods have become the norm in fields such as natural language processing, the implementation of transfer learning approaches in chemistry is still nascent in practice. We demonstrate how a GNN pretrained over 2.7 million water clusters using 64 IPUs (over 92 min) can be leveraged to quickly fine-tune the pretrained model for downstream tasks of molecular dynamics and level-of-theory transfer to a different potential energy surface (PES), taking only 8.3 h and 28 min, respectively, on a single GPU.

2. CHARACTERIZATION OF THE ATOMISTIC GRAPHS

A set of atomic coordinates, whether it be a single molecule or collections of molecules with intermolecular interactions, can be mathematically represented as a graph, with a vertex set and an edge set. Typically, each vertex of the graph represents an atom and is associated with that atom's spatial coordinates and atomic number. Each edge reflects a pairwise interaction between atoms. The property prediction task outlined in the HydroNet benchmark[47] is as follows: given a set of atoms with specified spatial coordinate information, predict the energy of the molecule. To demonstrate the effectiveness of our codesign approach as well as the scalability of the accelerated SchNet model on the IPUs, in this work, we leverage the following two atomistic graph data sets: QM9, which is a commonly used small-molecule data set,[45] and a recently released water cluster data set, which contains intermolecular interactions.[35,46]

The water cluster benchmark data set is made up of 4.5 M water cluster minima computed using the TTM2.1-F ab initio-based interaction potential for water. Each sample (graph) has between 9 and 90 atoms (vertices), and the task is to predict the energy of the cluster. Edges are applied between vertices based on a set spatial distance (here, ≤6 Å). Because physical constraints limit the number of atoms that can be packed into a region of space, water clusters tend to produce sparse graphs. The sparsity of a graph is defined as one minus the ratio of the number of edges N_e with respect to the maximum possible edges (i.e., application of an edge between all pairs of vertices), calculated by $1 - \frac{N_e}{N_v(N_v-1)}$. A sparsity of 0 indicates a complete graph in which all vertex pairs are connected by an edge. As seen in Figure 2, as the number of atoms increases, so does the sparsity of the graph. We examine the full benchmark data set as well as a 2.7 M subset containing clusters of size 9−75 vertices to explore the effect of reduced sparsity.

The second atomistic data set we consider is the widely used QM9 benchmark data set.[45] QM9 provides minimized geometries, along with the corresponding harmonic frequencies, dipole moments, polarizabilities, energies, enthalpies, and free energies of atomization computed at the B3LYP/6-31G(2df,p) level for 134 K stable small organic molecules made up of H, C, N, O, and/or F. As with the water cluster data set, atomistic graphs are generated from the geometries based on proximity, with a cutoff distance of ≤6 Å for comparison to the water cluster benchmark data set. The limited number of atoms in each structure in QM9 leads to reduced sparsity in the corresponding graph compared with structures in the water cluster data set (Figure 2).

3. DETAILS OF IPU HARDWARE AND SOFTWARE

The recently developed Graphcore IPU combines a number of hardware features with software abstractions to create a platform that is exceptionally well suited to the unique computational demands of atomistic GNNs. The main components of a Graphcore IPU are processing elements called tiles. Each tile consists of one computing core and ≈625 kB of local on-chip static random access memory (SRAM) memory, which is sufficient to store a large number of (packed) atomistic graphs (Figure 3a). Compared to GPU shared memory, this near-processor memory on an IPU tile

https://doi.org/10.1021/acs.jcim.3c01312
J. Chem. Inf. Model. XXXX, XXX, XXX−XXX

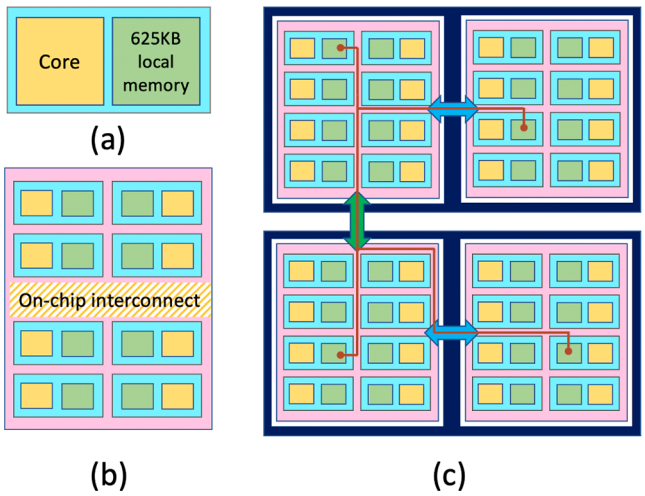


Figure 3. Illustration of tile-based IPU processor architecture and communication topologies used for intercard and inter-IPU communication. Each card has two IPU processors.

also enables faster access to the stored batches of atomistic graphs and avoids the overhead of accessing off-the-chip memory. In addition, IPUs are not susceptible to control flow divergence or memory access divergence since, irrespective of the access pattern, each core works with the locally available SRAM data. GPUs, on the other hand, incur significant overhead if memory access is irregular. The Bow IPU processor consists of a total of 1472 tiles, yielding a total of 900 MB of high-bandwidth (65 TB/s total) distributed SRAM memory available for machine learning workloads to exploit (Figure 3b). Each computing core has 6 hardware threads that are round-robin multiplexed in time. An on-chip interconnect implements a nonblocking, all-to-all pattern to enable high-bandwidth, low-latency communication among the tiles on an IPU. In addition, for interprocessor communication, low-latency, high-throughput IPU links are used (Figure 3c).

The IPU is routinely programmed using high-level Python frameworks such as PyTorch and TensorFlow. We use the PopTorch framework[48] which provides a set of extensions to PyTorch to efficiently target our implementation of SchNet on the IPU architecture. We implemented the interaction (GNN) layer of the SchNet model using the PyTorch-Geometric (PyG) framework.[49] Inside the GNN, an order-invariant gather function is used to aggregate the neighbor's influence on a given vertex via messages. The updated vertex embedding information is then scattered back to the neighbors for the computation of the next set of embeddings in the next interaction layer. These gather-scatter operations within the GNN for atomistic graphs are localized on a single IPU, so their execution is accelerated by utilizing the combined benefit of both the high-bandwidth local memory and the high-bandwidth communication over the on-chip interconnect.

4. CO-DESIGN OPTIMIZATIONS FOR ATOMISTIC GNNS

Effective memory management for the data structures and intermediate computation, optimizing parallel execution, and leveraging efficient all-to-all communication connectivity among the tiles are the key factors that determine the performance of atomistic GNNs. Here, we review a set of optimizations that we have implemented for the SchNet model that can be categorized into three broad categories: (i) input-specific optimization (packing), (ii) hardware-targeted optimizations (planning, prefetching, and compiler passes), and (iii) model-specific optimizations (merging weight updates and optimized softplus activation). In the following, we discuss these optimization techniques to improve the overall performance of the atomistic GNNs.

4.1. Input-Specific Optimization: Batch Packing. As we have discussed in Section 2, atomistic graph structures vary significantly in terms of the vertex counts and the edge counts. Efficiently targeting the high-bandwidth exchange fabric of the IPU is accomplished through static analysis of the communication patterns between tiles for a given application. This ahead-of-time compilation of atomistic GNNs requires knowing the shapes of the input tensors a priori for preparing the batches. To keep the shape sizes, such as the number of vertices and edges, fixed, the naive strategy is to just pad batches up to the maximum number of vertices or edges (Figure 4a). This can lead to a very large proportion of

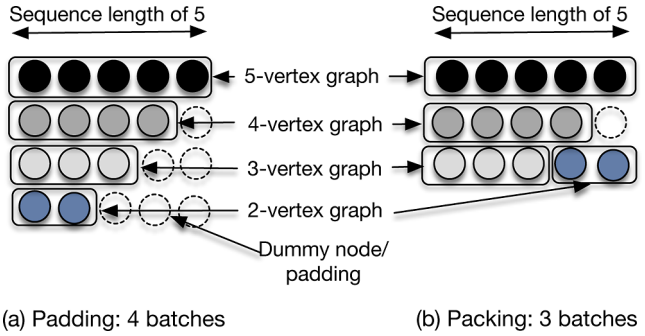


Figure 4. Padding vs batch packing of atomistic graphs.

padding and therefore may waste computation and memory resources. This padding technique is generally applied in well-known GNN libraries such as the PyTorch-Geometric (PyG) framework and Deep Graph Library (DGL)[50] when batches of small graphs are prepared by combining multiple graphs into a single graph of multiple disconnected components (known as "batched graphs" or "advanced mini-batches").

In addition to reducing the extra memory footprint of padding on any hardware, to leverage the faster but limited on-chip IPU SRAM memory, it becomes even more imperative to judiciously allocate memory for the input tensors representing the atomistic graphs. To this end, we first carefully formulate the problem of combining multiple small graphs into mini-batches as the well-known combinatorial optimization problem, namely bin packing:[51] given a set of graphs and a set of equal-sized bins (mini-batch size), the objective is to assign the graphs to the bins by using as few bins as possible. The constraint for this optimization problem is that the total size (total number of vertices) of the graphs assigned to a bin should not exceed the bin's capacity. Bin packing has been known to be an NP-complete problem for decades and is well established.

To solve this problem in the context of efficient construction of mini-batches for atomistic graphs, in this paper, we adopt an approach previously introduced in natural language processing, where different sequences are concatenated.[52] However, instead of concatenating sequences/sentences, we combine graphs, and instead of the number of words as the sequence length, we measure length by the number of vertices. We call

https://doi.org/10.1021/acs.jcim.3c01312
J. Chem. Inf. Model. XXXX, XXX, XXX−XXX

this strategy batch packing. We apply our packing strategy prior to the construction of the batched graphs or advanced mini-batches.

In practice, numerous heuristic-based algorithms for packing exist[53−56] that could be applied in this case. All of these approaches are in the range of $O(n \log n)$ time complexity, where n is the number of graphs/samples. Hence, we use the faster approach from ref 52, which is called longest-pack-first histogram-packing (LPFHP). The algorithm is derived from the best-fit method, in which an item gets added to the bin that will leave the least space after the item is added. The algorithm first computes the histogram of graph sizes (in terms of the total number of vertices in each graph) and the respective sample counts. The algorithm also takes as input the potential maximum sequence lengths in a pack. Next, it iterates from the largest to the smallest graph and tries to add it to existing combined graphs/packs (bin). The trick here is that the algorithm works on histogram bins and counts and not on single graphs. If a graph can fit with multiple different already combined graphs, we chose the one with the least space (best fit). For example, if we need to add a graph with a size of 10 and our maximum sequence length is 100, we would prefer to combine it with a graph of 90 vertices to get a perfect match instead of combining it with a pack or graph of size 11, which is the smallest setting that can occur. The pseudocode of the LPFHP heuristic-based batch packing algorithm can be found in the Supporting Information.

4.2. Hardware-Targeted Optimizations. For accelerating the training of atomistic GNN models on the IPUs, we leverage both a fully static compilation and a gather-scatter planner (Section 4.2.1). With static compilation, access to the computation graph provides the opportunity to reduce the maximum memory usage by changing the order of operations (called "rescheduling") in a full compute graph, thus decreasing the maximum live memory at any given point in the program. This may allow for both a full compute graph to fit into the SRAM and an increase in the amount of available memory at a given point in the program. Additionally, increased SRAM availability can allow some operations to make better use of high-bandwidth local memory to accelerate operations and assist the planner in deciding the optimal data layout. At the compiler level, two IPU-specific features are implemented for optimizing the execution of the atomistic GNNs: the planning of the scatter/gather operators for targeting the IPU and the scatter vectorization.

4.2.1. Scatter/Gather Planner. The message-passing layer that forms part of the Interaction Block/GNN layer of the SchNet model performs two important operations: gather and scatter. These operations' runtimes are dominated by memory accesses. In one representative sample, we found that approximately 25% of the execution time of a single training step is directly spent on the evaluation of the scatter/gather operators. The overall performance of a GNN model relies heavily on the efficient execution of these two operations. The gather step fetches the embeddings for atoms in the neighborhood of each atom, as well as the generated "continuous filter" values corresponding to each edge in the graph, for the aggregation phase of the GNNs. Once messages have been calculated, they are scattered so that messages for each destination atom are aggregated together, which gives the output embeddings. Gather provides batch reads, and scatter provides batch read-modify-writes of dynamically indexed chunks of memory. Hence, optimized implementations of gather and scatter for IPU hardware are required to coordinate dynamically indexed memory access across the distributed tile memories and maximize memory bandwidth utilization.

To this end, a scatter/gather planner has been employed. While varying the chunk size of different dimensions of the input tensors for the gather-scatter operations, the planner estimates the total number of machine cycles it takes to execute a gather and a scatter on a single tile in terms of the SRAM load/store bandwidth. In doing so, the planner tries to find an optimal partition of the tensors for the gather-scatter operations while minimizing data communication between tiles and the number of systemwide reduction operations required once per tile computation is done. Once the partitions are determined, both the gather and the scatter operations are executed by using a divide-and-conquer strategy. In particular, the full gather or scatter operations are divided evenly into smaller gather or scatter operations that are locally executed on each tile. The results are then exchanged and optionally reduced to give the final result. The cost function used for planning is one that estimates the maximum number machine cycles required to complete the operation on any one tile, and a minimum is found by an exhaustive search of valid implementation parameter settings. A detailed discussion of the cost function estimation can be found in the Supporting Information.

4.2.2. Vectorization of Gather/Scatter Operations. One of the benefits of the ahead-of-time compilation of a PyTorch model is the ability to write specialized passes that can perform operator fusion to efficiently target the IPU hardware. These passes can apply simple pattern matching to the captured computation graph and replace or fuse multiple operations into new patterns that are known to be more efficient to evaluate. In the context of GNNs, we have developed a fusion pass that finds a sequence where a vector of indices is broadcast ahead of aggregation. Our pass removes the redundant broadcasting step and ensures that the more efficient vectorized form of the scatter operation is targeted by the scatter/gather planner.

4.2.3. Host-Device I/O Optimizations. 4.2.3.1. Asynchronous, Nonblocking I/O. The preparation of mini-batches can be expensive as it involves the random access of irregular-sized atomistic graphs, followed by the collation process to combine multiple graphs into one large disconnected graph. We can improve the time to access a single graph by using a two-level caching strategy:

- The atomistic graphs are stored on hard drives in an efficient compressed serialized binary representation for multidimensional tensor data.
- The fully materialized graph data structure is cached in memory on first-time access which helps reduce redundant hard drive I/O.

We apply this caching strategy within multiple asynchronous workers for preparing mini-batches on the host, which effectively overlaps the time for preparing mini-batches with the computation on the IPU.

4.2.3.2. Prefetching. To reduce the waiting time between host-device transfers, a prefetch depth can be specified for the data stream transfer. The depth dictates the number of prefetched batches and enables the host-to-device transfer of the next mini-batch simultaneously, while the device continues evaluation of the training loop on the current mini-batch.

4.3. Model-Specific Optimizations. *4.3.1. Merged Communication Collectives.* During data parallel training, each

https://doi.org/10.1021/acs.jcim.3c01312
J. Chem. Inf. Model. XXXX, XXX, XXX−XXX

replica (IPU) computes a local gradient for the mini-batch. These gradients are then combined across all replicas ahead of performing the weight update. To reduce the latency in performing the weight update, allreduce collectives are merged together to communicate all variables at once.

4.3.2. Optimized Softplus. The default implementation of the softplus activation is defined in PyTorch as

$$\text{softplus}(x) = \begin{cases} \frac{1}{\beta}\log(1 + \exp(\beta x)), & \text{if } \beta x \le \tau \\ x, & \text{if } \beta x > \tau \end{cases} \quad (1)$$

where τ is a threshold value for which the activation saturates the linear function. This conditional expression is used for numerical stability, but for the default values of $\tau = 20$ and $\beta = 1$, we can more efficiently evaluate the stable softplus activation equivalently as follows

$$\text{softplus}(x) = \log(1 + \exp(-|x|)) + \max(x, 0) \quad (2)$$

which is used in our optimized implementation of the SchNet model as this simplifies expression compiles down to a more efficient compute vertex than the original formulation in eq 1 and is numerically stable without additional parameters.

5. EVALUATION

We performed a comprehensive set of experiments to evaluate the effect of our proposed optimizations for the SchNet model on predicting properties on a large-scale IPU system. In addition, we conducted strong scaling experiments and benchmarked our holistic codesigned approach with a SchNet implementation modified to run on multiple GPUs. The results are discussed in the following subsections.

5.1. Details of the Experimental Setup. *5.1.1. Hardware Configuration.* The GPU experiments were conducted on Nvidia Ampere A100 GPUs. This system consists of 8 A100 GPUs with 40 GB of main memory attached to each GPU. All IPU experiments were conducted on a Bow Pod64 IPU system. We used this system to study the scaling behavior for data parallel training from a single IPU to 64 IPUs. Since 8 A100 GPUs and 16 IPUs are closely equivalent in terms of their compute capabilities, power, and cost, we consider these two configurations of the hardware accelerators equivalent when comparing the execution time of the SchNet model on these two hardware accelerators (Section 5.4 and Table 1).

5.1.2. Hyperparameter Setup. We used an up-to-date SchNet model implementation in PyTorch Geometric for all experiments reported here. Except where otherwise noted, we used 4 interaction blocks, a hidden feature size of 100, and a uniform grid of 25 Gaussians for the basis function expansion of the interatomic distances. This model was compiled for the IPUs using PopTorch version 3.0.0, which uses PyTorch version 1.10.1 implicitly. We used the Adam optimizer with a learning rate of 0.001 and collected throughput and speedup numbers by executing 25 epochs of a standard training loop.

5.1.3. Batch Packing Setup. In our experiments, the batch packing strategy is expressed as an optimization task that is solved once as a preprocessing step in our training pipeline and is used to guide how mini-batches are sampled from the data set to ultimately build batches of fixed size. Our implementation randomizes both the ordering of the packs but also the order within the packs to ensure that fictitious correlations between packs and within packs are not learned by the network. Except in Section 5.5, for the full Hydronet benchmark data set, we used a pack size of five atomistic graphs with a total number of atoms of 192.

5.1.4. Batch Size. We have conducted experiments with various batch sizes on both the GPUs and the IPUs, and we picked the batch size that maximizes the throughput without sacrificing accuracy. For the reported experimental results on the IPUs, the batch size was set to 224, and on the GPUs, the batch size was set to 256. Batch sizes of 448 for the IPUs and 512 for the GPUs yield similar results.

5.2. Strong Scaling Results. We report the strong scaling training performance of the SchNet model on the IPUs in Figure 5. Here, we keep the data set size constant while increasing the number of IPUs. The reported performance metric, average throughput, is calculated as the number of graphs processed per second (similar to the spirit of evaluating the number of sequences processed per second in training language models). As can be observed from the figure, in most cases, increasing the number of IPUs increases the average throughput. With smaller data sets such as QM9, however, the throughput is maximized with 32 IPUs (with packing). Beyond 32 IPUs, the performance deteriorates due to not having enough work to sustain higher throughput. With the 2.7 and 4.5 M water cluster data sets and with the packing technique, the average throughput continues to increase up through 64 IPUs, as there is sufficient available work to keep all the independent processors busy.

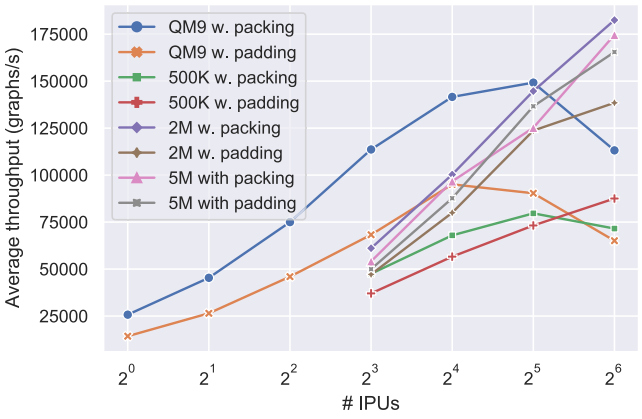


Figure 5. Strong scaling results of the optimized SchNet model on the IPUs with packing and padding techniques. All additional optimizations discussed in Section 4 are also applied in both cases.

Table 1. Average per Epoch Execution Time in Seconds with Different Number of IPUs and GPUs[a]

data set	8 IPUs	16 IPUs	8 GPUs	32 IPUs	64 IPUs
QM9	0.91	0.72	1.86	0.68	0.9
500k	8.39	5.36	6.87	5.0	5.57
2.7 M	35.07	21.37	34.36	14.81	11.74
4.5 M	62.56	35.0	60	27.03	19.38

[a]8 A100 GPUs and 16 IPUs are closely equivalent in terms of compute capability, power, and cost. Hence, we compare the average per-epoch execution time with 8 A100 GPUs and 16 IPUs.

5.3. Average Per Epoch Execution Time with Different Number of IPUs. We also report the average per-epoch execution time in seconds as we increase the number of IPUs in Table 1. With the QM9 data set, the SchNet model achieves

https://doi.org/10.1021/acs.jcim.3c01312
J. Chem. Inf. Model. XXXX, XXX, XXX−XXX

the best performance with 32 IPUs, closely followed by performance while running on 16 IPUs. With the QM9 data set on 64 IPUs, the overhead of communication starts to dominate the execution time instead of the available computation. With the 2.7 and 4.5 M water cluster data sets, due to their larger sizes, increasing the number of IPUs helps achieve better performance as there are more local computations available.

5.4. Comparison with a Different Hardware Accelerator (GPUs). We compare the performance of training our optimized SchNet model for atomistic GNNs on the IPUs with a modified GPU implementation from the PyTorch Geometric (PyG) library. Specifically, for running on multiple GPUs, we modified the existing single-GPU SchNet implementation in PyG to leverage the Distributed Data Parallel (DDP) model of the PyTorch library without introducing any further optimizations. We conduct the experiments with different data sets for 25 epochs and report the results in Table 1. Here, we report and compare the average per epoch execution time with 8 A100 GPUs and 16 IPUs due to their equivalence in computational capabilities, power, and cost. As can be observed from Table 1, the SchNet model runs faster on the 16 IPUs compared with 8 A100 GPUs. In particular, with QM9, 500k water cluster, 2.7 M water cluster, and 4.5 M water cluster data sets, the SchNet model running on the 16 IPUs achieves a speedup of 2.58×, 1.28×, 1.6×, and 1.71×, compared to the 8 A100 GPUs, respectively. The speedup is computed by taking the average of the execution time for 25 epochs for 16 IPUs and 8 GPUs, then taking the ratio of the average runtime on the GPUs and on the IPUs. Additional hardware-agnostic optimization discussed in the paper is certainly a direction for future improvement of the GPU implementation.

5.5. Evaluation of Different Optimization Techniques. *5.5.1. Evaluation of the LPFHP Packing Technique.* We first evaluate the performance of the IPU SchNet model when the batch packing technique with the LPFHP heuristic (Section 4.1) is applied instead of the padding technique. The experimental results are reported in Figure 6.

The speedup achieved with batch packing is calculated by taking the ratio of the average model training time for padding and packing. As can be observed from Figure 6, our LPFHP packing technique may improve the performance of the SchNet model by up to 25% compared to the baseline IPU implementation that applies the padding technique.

We also evaluate the efficiency of our LPFHP batch packing algorithm compared with the padding technique (Figure 7). Efficiency is defined as the percentage of padding reduced by applying the LPFHP batch packing algorithm. With the QM9 data set, padding may result in 38% wastage of memory, where the maximum number of vertices in a graph is 29 (Figures 2 and 7). To determine the optimal maximum sequence length for packing, it is to be noted from the histogram of the number of vertices in graphs (Figure 2) that sometimes the mode of the distribution is larger than half of the maximum number of vertices, as is the case for QM9 as well as the water cluster minima database applied in the HydroNet benchmark task. In this case, if the maximum sequence length, i.e., maximum number of vertices in a pack, is set to a smaller value, a substantial amount of padding would still be required (30% on QM9 compared to 38% with normal padding). Thus, it is beneficial to increase the allowed maximum number of vertices in a pack of graphs that gets processed jointly. For QM9, this

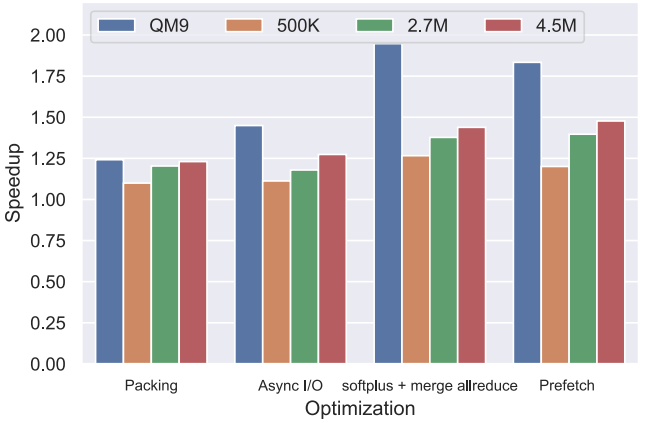


Figure 6. Speedup of the IPU SchNet model with different optimizations w.r.t. a baseline IPU implementation applying only the padding technique. The optimizations specified on the horizontal axis of the plot are added to the baseline IPU implementation progressively from the left to the right. For example, the legend Async/IO implies the application of the packing technique with asynchronous, nonblocking/IO. Note that applying additional optimizations (such as prefetching) may result in slower execution (for example, for QM9, due to the smaller data set size). The experiments were conducted on 16 IPUs.

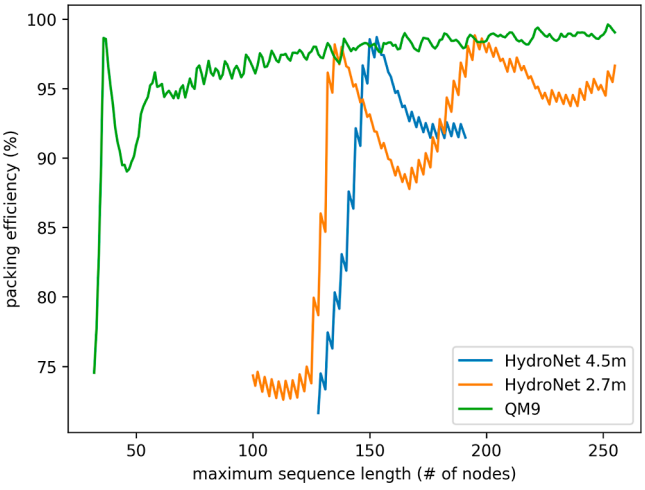
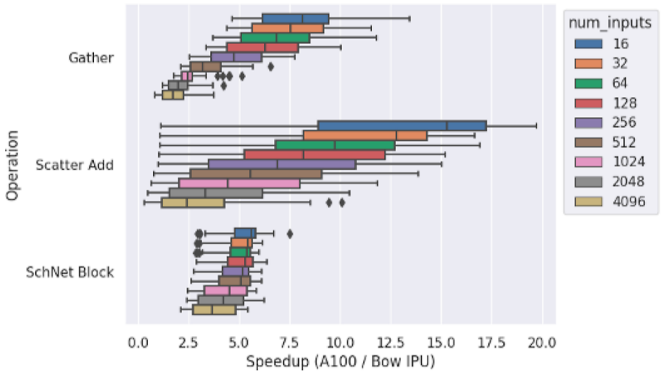


Figure 7. Effect of the maximum number of vertices in a pack (sequence length) on packing efficiency, i.e., on the amount of padding reduced.

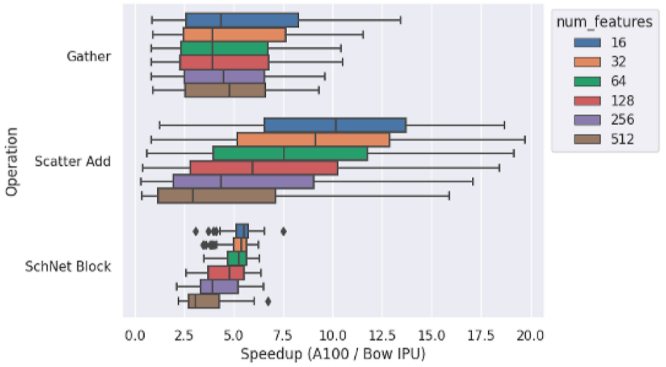
can reduce the padding by less than 2% (Figure 7). Similar observations can be made for the two water cluster minima databases. In all cases, the curves are not smooth and contain several systematic spikes. This can be explained by the histograms displayed in Figure 2. Only certain numbers of vertices occur in the data set, and there are several gaps, and even for the existing numbers, the distributions are not smooth. This directly translates into packing results.

5.5.2. Other Packing Heuristics. In addition to the Longest Pack First (LPFHP) heuristic for batch packing, additionally, we considered non-negative least squares (NNLS),[57] where the packing problem is formulated as a weighted non-negative least-squares problem, and Shortest Pack First (SPF) Histogram Packings (HP) heuristics. With the 4.5 M water cluster minima database, the packing step with the LPFHP or the SPFHP heuristic takes only milliseconds. In contrast, the

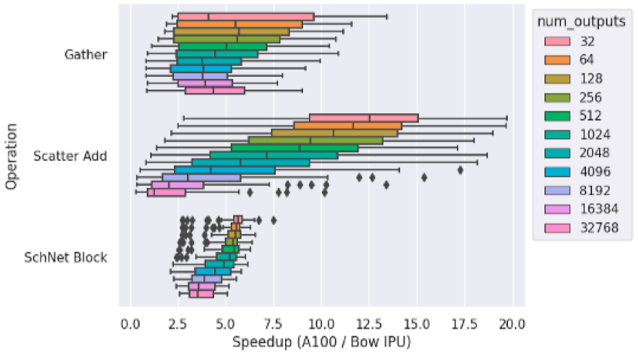
https://doi.org/10.1021/acs.jcim.3c01312
J. Chem. Inf. Model. XXXX, XXX, XXX−XXX



(a) Varying the number of inputs.



(b) Varying the number of features.



(c) Varying the number of outputs.

Figure 8. Effect on the performance of the gather, scatter-add, and SchNet interaction block microbenchmarks while varying the input size, number of features, and output sizes.

Table 2. Test-Set Accuracy, Reported as Mean Absolute Error (MAE) in E_{H_2O} (kcal/mol/water), F_{mag} (kcal/mol/Å), and F_{ang}[a]

variation	hardware	data set	N_{train}	E_{H_2O}	F_{mag}	F_{ang}
Pretrained	IPU	minima	2.7 M	0.0018	21.89	0.501
Scratch	GPU	nonminima	12 K	0.3548	18.76	0.238
Fine-tuned	GPU	nonminima	12 K	0.1316	8.94	0.098

[a]Test sets were derived from the specified dataset.

NNLSHP heuristic takes around 30 s. In all cases, the packing efficiencies are comparable.

5.5.3. Impact of Packing on Accuracy. Ideally, a mini-batch should be formed by randomly sampling from the data set, but the packing algorithm eliminates this random process. We randomly shuffle the data set once before packing and then shuffle the packed minibatches every epoch. Training accuracy is unaffected by this process due to the combined randomness of how graphs are assigned to packages, the size of the packages, and the model batch size.

5.5.4. Impact of Asynchronous I/O. In addition to applying the packing technique, enabling asynchronous I/O improves the performance of the SchNet model, as reported in Figure 6. We observe that, with these two techniques, the QM9 data set achieves better improvement in performance.

5.5.5. Impact of the Optimized Softplus, Merging Allreduce and Prefetching. Adding a cycle-optimized implementation of the softplus activation and merging the allreduce communication collective improves the performance of training the SchNet model using IPUs for all the data sets (Figure 6). While prefetching improves performance with the 4.5 M water cluster data set, it negatively impacts the performance of the QM data set. Here, the prefetch depth is set to 4.

5.5.6. Overall Impact of the IPU Hardware and Software Optimizations on Performance. Notably, by considering the results reported in Figure 6 and Table 1, it can be observed that the majority of the performance improvement of the SchNet model on the IPU can be attributed to software optimizations rather than the unique features of the IPU hardware. For example, with the QM9 data set, software optimizations yield a speedup of 1.9× on the IPUs, and compared to the GPU the speedup is 2.58×. Similarly, with the 4.5 M water cluster data set, the speedups are 1.45× and 1.7×, respectively.

5.6. Microbenchmark Performance. To further analyze the speedups reported in Section 5.4, we have also performed kernel-level benchmarks of gather, scatter, and the SchNet interaction block. In each case, we observe the impact on performance while varying the input tensor dimensions, the number of features, and the lookup tensor (target output) size (i.e., the size of the `index` tensor to be gathered/scattered or

https://doi.org/10.1021/acs.jcim.3c01312
J. Chem. Inf. Model. XXXX, XXX, XXX−XXX

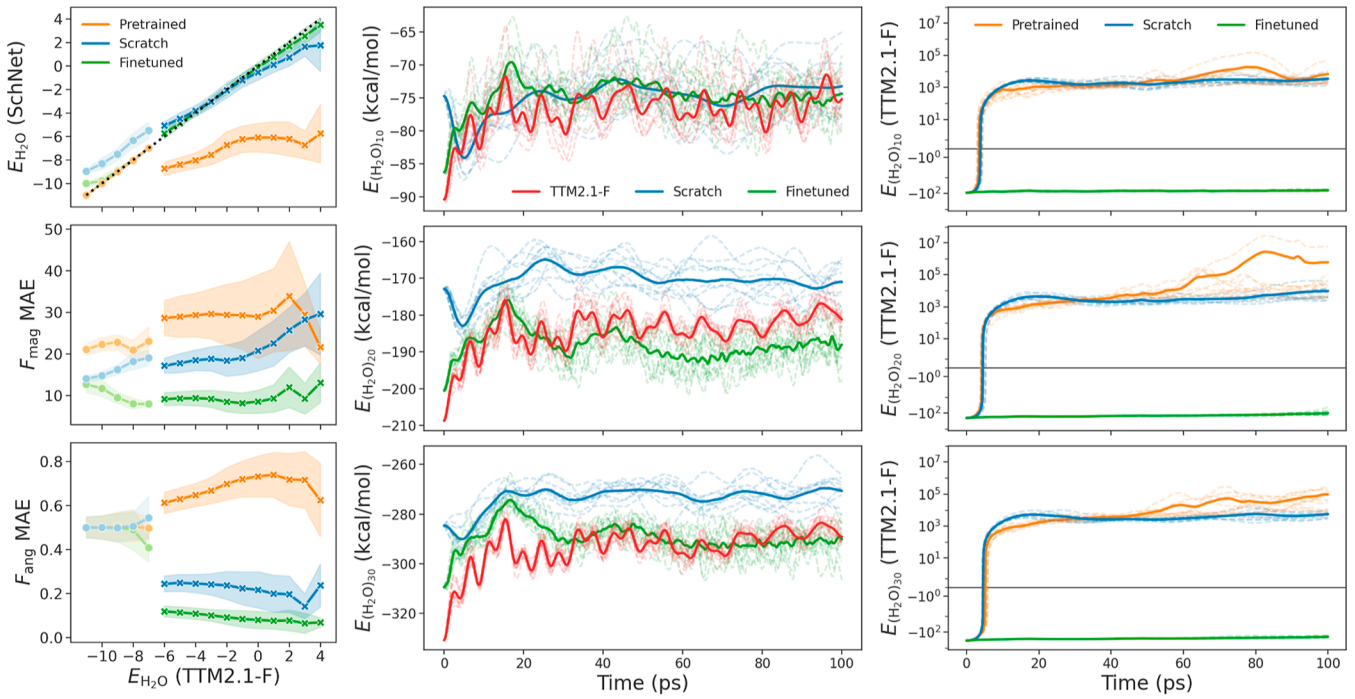


Figure 9. (Left) Static predictions of E_{H_2O}, F_{mag}, and F_{ang} on minima (•) and nonminima (×) test sets. (Center) MD using the TTM2.1-F potential and trained atomic GNNs for water clusters of size N = 10, 20, and 30. (Right) Static TTM2.1-F predictions on MD trajectories generated by the NNPs.

the number of edges in the interaction block). The evaluation was conducted on a Bow POD (16 IPUs) as well as on an A100 (8 GPUs) system. The experimental results are reported in Figure 8. For these benchmarks, on average, a Bow POD achieves a 5× speedup over A100. In general, if the dimensions of the tensors are relatively small, IPUs achieve better performance over GPUs due to their capability of using faster local on-chip SRAM.

6. APPLICATION TO TRANSFER LEARNING

Exploiting the chemical space learned from training on large atomistic databases can improve generalization to new applications by fine-tuning the pretrained model on a new data set.[58] Such so-called transfer learning is common in the domains of computer vision and natural language processing (NLP), where models pretrained on millions of data points are readily available. While advancements from GPT-3 and other pretrained NLP models have led to large-scale chemical language models,[59−61] exploration of comparable foundation models for interatomic potentials is still nascent.[62−64]

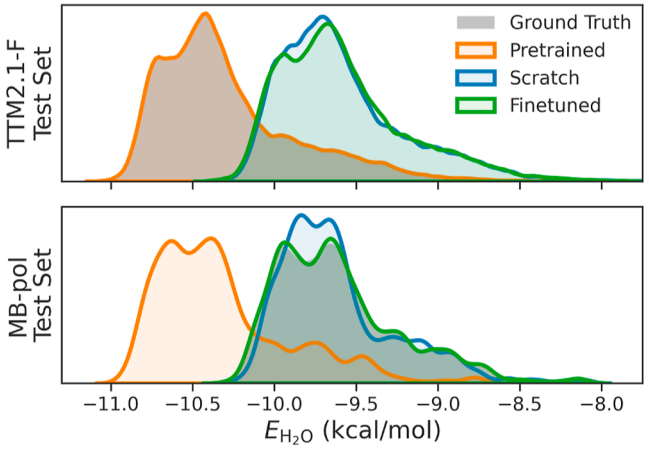


Figure 10. Predicted E_{H_2O} distributions on test sets computed with the TTM2.1-F (top) and MB-pol (bottom) potentials.

In this section, we show how a pretrained atomistic GNN can be transferred from (1) the task of property prediction to molecular dynamics (MD) and (2) to a new level of theory. We start with the SchNet model trained on 2.7 M water clusters by using Graphcore IPUs. We then fine-tune the pretrained model using orders-of-magnitude smaller data sets on a single V100 GPU. We present performance between three model variations: SchNet trained on 2.7 M water cluster minima (Pretrained), the pretrained model further fine-tuned using a much smaller data set (Fine-tuned), and a SchNet model trained from scratch using only the smaller data set (Scratch)—on two downstream tasks—data space expansion and PES transfer.

6.1. Task Transfer. To transfer the model from the task of property prediction to MD, we collected a small data set of nonminima from MD simulations performed at 260 and 300 K. To drive MD simulations using atomistic GNNs, atomic forces **F** are obtained from the negative of the GNN gradient. During training, a force term is typically added to the loss function to improve prediction accuracy.[65] Training without the force term, as was done for the pretrained model, leads to poor predictions of **F**, even though the accuracy in E predictions on minima configurations is high. Moreover, the pretrained model produces poor E predictions on nonminima, necessitating the need for the data space covered by the model to be expanded. Fine-tuning the pretrained model on a much smaller subset of nonminima (>1% of the minima data set) and including a force term in the loss improves predictions of nonminima **F** and E. Because of the roughly three-order-of-magnitude

https://doi.org/10.1021/acs.jcim.3c01312
J. Chem. Inf. Model. XXXX, XXX, XXX−XXX

Table 3. Test-Set Errors in E_{H_2O} (kcal/mol/water)

			TTM2.1-F test set		MB-pol test set	
variation	hardware	train set	MAE	RMSE	MAE	RMSE
Pretrained	IPU	TTM2.1-F	0.0018	0.0032	0.7071	0.7112
Scratch	GPU	MB-pol	0.6973	0.7009	0.0719	0.0924
Fine-tuned	GPU	MB-pol	0.7033	0.7071	0.0122	0.0158

decrease in the size of the training set when using the pretrained model, training was accomplished on a single NVIDIA V100 GPU in 8.3 h.

Table 2 shows the test-set accuracy of the pretrained model and models trained on nonminima with and without pretraining. The error in E is very low for the pretrained model (0.0018 kcal/mol/water), in part due to the restriction of geometries to minima. Fine-tuning on the pretrained model increases the E error to 0.1316 kcal/mol/water due to the expansion of the geometries and the range of energies to be predicted. Training a model from scratch on the data set used for fine-tuning led to further increased errors in E of 0.3548 kcal/mol/water. Following Chmiela et al.,[65] we quantify the topological accuracy of atomic force predictions by the magnitude error $F_{mag} = \|\hat{\mathbf{F}}\| - \|\mathbf{F}\|$, which describes the extent to which the slope of the predicted and reference PES differ, and the angular error $F_{ang} = \cos^{-1}(\hat{\mathbf{F}}/\|\hat{\mathbf{F}}\| \cdot \mathbf{F}/\|\mathbf{F}\|)/\pi$, which describes the orientation of the predicted force direction relative to the reference force direction and ranges between 0 (aligned) and 1 (inverted). Prior to fine-tuning, both F_{mag} and F_{ang} are very high, 21.89 and 0.501, respectively. After fine-tuning the pretrained model, F_{mag} reduces to 8.94 and F_{ang} reduces to 0.098. Notably, when the model is trained from scratch, F_{mag} and F_{ang} are more than twice as high: 18.76 and 0.238, respectively.

We then performed MD simulations driven by the trained models (Figure 9). Berendsen NVT dynamics of three clusters (N = 10, 20, and 30) at 300 K were simulated 10 times each with different random seeds. The mean (bold lines) and individual simulations (dashed lines) are shown for TTM2.1-F and the GNNs trained on nonminima. MD simulations using the pretrained model before fine-tuning are not shown, as the predicted energies were several orders of magnitude below those from TTM2.1-F. The mean E of the fine-tuned model aligns for all cluster sizes, while that of the model without pretraining aligns only for N = 10. We then calculated the TTM2.1-F E for each point in the GNN-generated trajectories to validate the resulting geometries. Notably, only the fine-tuned model produced dynamics within a valid energy range (E < 0 kcal/mol). Though the model without pretraining predicted E of a similar magnitude to TTM2.1-F, the generated structures were calculated by TTM2.1-F to be highly unstable. In fact, the model trained from scratch did not outperform even the pretrained model in generating stable dynamics.

6.2. Level-of-Theory Transfer. The PES of a physical system will have a different representation, depending on the method of calculation, each of which can have different associated computational costs. The PES approximated by a model can be amended by updating a model trained on a large data set collected by a specific method (here, using the TTM2.1-F potential) using a small data set collected by an alternative method (here, the MB-pol potential). We perform such level-of-theory transfer by fine-tuning the pretrained model on 5000 water cluster minima with E computed with the MB-pol potential.[66,67] Because of the drastically reduced size of the data set, the model was trained in under 28 min on 1 NVIDIA V100 GPU. Figure 10 shows E_{H_2O} distributions for the TTM2.1-F and MB-pol test sets, with errors shown in Table 3. Though the comparable test-set errors in E_{H_2O} after training on the MB-pol data do not reach the low error of the pretrained model on its test set (0.0018 kcal/mol/water), fine-tuning from the pretrained model gave a 6-fold decrease in error compared with training on MB-pol data from scratch and was able to more accurately reproduce the E_{H_2O} distribution.

The codebase used to fine-tune the models and analyze the results can be found at https://github.com/pnnl/downstream_mol_gnn. Accompanying data sets and MD trajectories produced by the GNN potentials can be obtained at https://data.pnnl.gov/group/nodes/dataset/33283.

7. CONCLUSIONS

The long history of science is full of examples where new technology has led to a deeper understanding of natural phenomena. More recently, progress in machine learning applied to computer vision and natural language processing has previously been driven by the repurposing of GPU accelerators for the compute workloads of these deep neural networks.[68,69] We present here the first application of IPUs to atomistic GNNs and demonstrate acceleration compared with a baseline GPU implementation. Beyond demonstrating this emerging capability, this paper proposes a set of optimization techniques for executing GNNs with atomistic graphs: packing for efficient representation of the batches, planning for well-balanced work partitioning for optimized scatter-gather execution, overlapping computation and communication techniques for reducing the I/O bottleneck, and model-specific optimizations. We demonstrate that, altogether, these optimizations can take advantage of the high-bandwidth local memory and faster interconnect of the IPU architecture to significantly improve the training time of the atomistic GNNs. As illustrated by the transfer learning application, we believe that accelerators developed using a codesign principle have the potential to introduce novel applications of machine learning for chemistry and materials science.

Use of the PyTorch-Geometric framework allows atomistic GNNs to be easily transferred among IPUs, GPUs, and CPUs. As we showed in this report, an atomistic GNN pretrained on the large water cluster database using IPUs could be fine-tuned on a single GPU using orders-of-magnitude less data. The reduced hardware and data requirements afforded by employing a pretrained model permit experimentation by a larger number of researchers while also decreasing energy resources consumed during experimentation. Such cross-hardware transfer learning clears a path for atomistic foundational models pretrained on hardware accelerators but useable and updatable by those with limited computational resources.

https://doi.org/10.1021/acs.jcim.3c01312
J. Chem. Inf. Model. XXXX, XXX, XXX−XXX

■ ASSOCIATED CONTENT

Data Availability Statement

The full database of water cluster minima computed with the TTM2.1-F potential is available for download at https://data.pnnl.gov/group/nodes/dataset/33224. The preprocessed databases for training, including the database of nonminima computed with the TTM2.1-F potential and the database of minima computed with the MB-pol potential, data set split files, trained model state dictionaries, ASE databases used for MD simulations, and results of data space expansion and PES transfer analyses, are available for download at https://data.pnnl.gov/group/nodes/dataset/33283. The codebase used to train molecular GNNs on IPUs is available at https://github.com/graphcore-research/hydronet-gnn. The codebase used to fine-tune IPU-trained models using GPUs or CPUs is available at https://github.com/pnnl/downstream_mol_gnn.

Supporting Information

The Supporting Information is available free of charge at https://pubs.acs.org/doi/10.1021/acs.jcim.3c01312.

Batch packing algorithm, planner cost estimation, impact of embedding size and number of interaction layers, and MSE loss (PDF)

■ AUTHOR INFORMATION

Corresponding Authors

Sotiris S. Xantheas − *Advanced Computing, Mathematics and Data Division, Pacific Northwest National Laboratory, Richland, Washington 99352, United States; Department of Chemistry, University of Washington, Seattle, Washington 98185, United States;* orcid.org/0000-0002-6303-1037; Email: sotiris.xantheas@pnnl.gov

Sutanay Choudhury − *Advanced Computing, Mathematics and Data Division, Pacific Northwest National Laboratory, Richland, Washington 99352, United States;* Email: sutanay.choudhury@pnnl.gov

Authors

Hatem Helal − *Graphcore, Cambridge CB1 2JH, U.K.*

Jesun Firoz − *Advanced Computing, Mathematics and Data Division, Pacific Northwest National Laboratory, Seattle, Washington 98109, United States*

Jenna A. Bilbrey − *Artificial Intelligence and Data Analytics Division, Pacific Northwest National Laboratory, Richland, Washington 99352, United States*

Henry Sprueill − *Artificial Intelligence and Data Analytics Division, Pacific Northwest National Laboratory, Richland, Washington 99352, United States*

Kristina M. Herman − *Department of Chemistry, University of Washington, Seattle, Washington 98185, United States;* orcid.org/0000-0003-4923-5727

Mario Michael Krell − *Graphcore, Cambridge CB1 2JH, U.K.*

Tom Murray − *Graphcore, Cambridge CB1 2JH, U.K.*

Manuel Lopez Roldan − *Graphcore, Cambridge CB1 2JH, U.K.*

Mike Kraus − *Graphcore, Cambridge CB1 2JH, U.K.*

Ang Li − *Advanced Computing, Mathematics and Data Division, Pacific Northwest National Laboratory, Richland, Washington 99352, United States*

Payel Das − *IBM Research, Yorktown Heights, New York 10598, United States;* orcid.org/0000-0002-7288-0516

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jcim.3c01312

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

The work was partially supported by the ExaLearn Co-design Center within the Exascale Computing Project (17-SC-20-SC), a collaborative effort between the U.S. Department of Energy's Office of Science and the National Nuclear Security Administration. K.M.H. and S.S.X. acknowledge support from the Center for Scalable Predictive Methods for Excitations and Correlated Phenomena (SPEC), which is funded by the U.S. Department of Energy, Office of Science, Basic Energy Sciences, Chemical Sciences, Geosciences and Biosciences Division, as part of the Computational Chemical Sciences (CCS) program under FWP 70942 at Pacific Northwest National Laboratory (PNNL), a multiprogram national laboratory operated by Battelle for the U.S. Department of Energy under Contract DE-AC05-76RL01830. This research was partially supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 66150: "CENATE - Center for Advanced Architecture Evaluation."

■ REFERENCES

(1) Krenn, M.; Pollice, R.; Guo, S. Y.; Aldeghi, M.; Cervera-Lierta, A.; Friederich, P.; dos Passos Gomes, G.; Häse, F.; Jinich, A.; Nigam, A.; et al. On scientific understanding with artificial intelligence. *Nat. Rev. Phys.* **2022**, *4*, 761−769.

(2) Moosavi, S. M.; Jablonka, K. M.; Smit, B. The role of machine learning in the understanding and design of materials. *J. Am. Chem. Soc.* **2020**, *142*, 20273−20287.

(3) Arús-Pous, J.; Blaschke, T.; Ulander, S.; Reymond, J.-L.; Chen, H.; Engkvist, O. Exploring the GDB-13 chemical space using deep generative models. *J. Cheminf.* **2019**, *11*, 20.

(4) Das, P.; Sercu, T.; Wadhawan, K.; Padhi, I.; Gehrmann, S.; Cipcigan, F.; Chenthamarakshan, V.; Strobelt, H.; Dos Santos, C.; Chen, P.-Y.; et al. Accelerated antimicrobial discovery via deep generative models and molecular dynamics simulations. *Nat. Biomed. Eng.* **2021**, *5*, 613−623.

(5) Qiao, Z.; Christensen, A. S.; Welborn, M.; Manby, F. R.; Anandkumar, A.; Miller, T. F., III Informing geometric deep learning with electronic interactions to accelerate quantum chemistry. *Proc. Natl. Acad. Sci. U.S.A.* **2022**, *119*, No. e2205221119.

(6) Fiedler, L.; Shah, K.; Bussmann, M.; Cangi, A. Deep dive into machine learning density functional theory for materials science and chemistry. *Phys. Rev. Mater.* **2022**, *6*, 040301.

(7) Kulichenko, M.; Smith, J. S.; Nebgen, B.; Li, Y. W.; Fedik, N.; Boldyrev, A. I.; Lubbers, N.; Barros, K.; Tretiak, S. The rise of neural networks for materials and chemical dynamics. *J. Phys. Chem. Lett.* **2021**, *12*, 6227−6243.

(8) Behler, J. First principles neural network potentials for reactive simulations of large molecular and condensed systems. *Angew. Chem., Int. Ed.* **2017**, *56*, 12828−12840.

(9) Smith, J. S.; Isayev, O.; Roitberg, A. E. ANI-1: an extensible neural network potential with DFT accuracy at force field computational cost. *Chem. Sci.* **2017**, *8*, 3192−3203.

(10) Lubbers, N.; Smith, J. S.; Barros, K. Hierarchical modeling of molecular energies using a deep neural network. *J. Chem. Phys.* **2018**, *148*, 241715.

(11) Kipf, T. N.; Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

(12) Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; Bengio, Y. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*, 2018.

(13) Hamilton, W.; Ying, Z.; Leskovec, J. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems (NeurIPS)*; Curran Associates, Inc., 2017.
(14) Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; Dahl, G. E. Neural message passing for quantum chemistry. *arXiv* **2017**, arXiv:1704.01212.
(15) Schütt, K. T.; Sauceda, H. E.; Kindermans, P.-J.; Tkatchenko, A.; Müller, K. R. SchNet − A deep learning architecture for molecules and materials. *J. Chem. Phys.* **2018**, *148*, 241722.
(16) Klicpera, J.; Groß, J.; Günnemann, S. Directional message passing for molecular graphs. *arXiv* **2020**, arXiv:2003.03123.
(17) Zhang, S.; Liu, Y.; Xie, L. Molecular mechanics-driven graph neural network with multiplex graph for molecular structures. *arXiv* **2020**, arXiv:2011.07457.
(18) Wang, Y.; Wang, J.; Cao, Z.; Barati Farimani, A. Molecular contrastive learning of representations via graph neural networks. *Nat. Mach. Intell.* **2022**, *4*, 279−287.
(19) Schwaller, P.; Laino, T.; Gaudin, T.; Bolgar, P.; Hunter, C. A.; Bekas, C.; Lee, A. A. Molecular transformer: a model for uncertainty-calibrated chemical reaction prediction. *ACS Cent. Sci.* **2019**, *5*, 1572−1583.
(20) Batzner, S.; Musaelian, A.; Sun, L.; Geiger, M.; Mailoa, J. P.; Kornbluth, M.; Molinari, N.; Smidt, T. E.; Kozinsky, B. E. (3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nat. Commun.* **2022**, *13*, 2453.
(21) Schmidt, J.; Marques, M. R.; Botti, S.; Marques, M. A. Recent advances and applications of machine learning in solid-state materials science. *npj Comput. Mater.* **2019**, *5*, 83.
(22) Lim, J.; Ryu, S.; Park, K.; Choe, Y. J.; Ham, J.; Kim, W. Y. Predicting drug−target interaction using a novel graph neural network with 3D structure-embedded graph representation. *J. Chem. Inf. Model.* **2019**, *59*, 3981−3988.
(23) Bilbrey, J. A.; Heindel, J. P.; Schram, M.; Bandyopadhyay, P.; Xantheas, S. S.; Choudhury, S. A look inside the black box: Using graph-theoretical descriptors to interpret a Continuous-Filter Convolutional Neural Network (CF-CNN) trained on the global and local minimum energy structures of neutral water clusters. *J. Chem. Phys.* **2020**, *153*, 024302.
(24) Addanki, R.; Battaglia, P. W.; Budden, D.; Deac, A.; Godwin, J.; Keck, T.; Li, W. L. S.; Sanchez-Gonzalez, A.; Stott, J.; Thakoor, S.; et al. Large-scale graph representation learning with very deep gnns and self-supervision. *arXiv* **2021**, arXiv:2107.09422.
(25) Tripathy, A.; Yelick, K.; Buluç, A. Reducing communication in graph neural network training. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020; pp 1−14.
(26) Wang, Y.; Feng, B.; Li, G.; Li, S.; Deng, L.; Xie, Y.; Ding, Y. GNNAdvisor: An Efficient Runtime System for GNN Acceleration on GPUs. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI'21)*, 2021.
(27) Kaler, T.; Stathas, N.; Ouyang, A.; Iliopoulos, A.-S.; Schardl, T.; Leiserson, C. E.; Chen, J. Accelerating training and inference of graph neural networks with fast sampling and pipelining. In *Proceedings of Machine Learning and Systems*, 2022; Vol. 4, pp 172−189.
(28) Zheng, D.; Song, X.; Ma, C.; Tan, Z.; Ye, Z.; Dong, J.; Xiong, H.; Zhang, Z.; Karypis, G. Dgl-ke: Training knowledge graph embeddings at scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020; pp 739−748.
(29) Gandhi, S.; Iyer, A. P. P3: Distributed deep graph learning at scale. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, 2021; pp 551−568.
(30) Zheng, D.; Song, X.; Yang, C.; LaSalle, D.; Karypis, G. Distributed Hybrid CPU and GPU training for Graph Neural Networks on Billion-Scale Heterogeneous Graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022; pp 4582−4591.
(31) Karypis, G.; Kumar, V. Multilevelk-way partitioning scheme for irregular graphs. *J. Parallel Distr. Comput.* **1998**, *48*, 96−129.
(32) Morari, A.; Tumeo, A.; Chavarría-Miranda, D.; Villa, O.; Valero, M. Scaling irregular applications through data aggregation and software multithreading. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, 2014; pp 1126−1135.
(33) Geng, T.; Li, A.; Shi, R.; Wu, C.; Wang, T.; Li, Y.; Haghi, P.; Tumeo, A.; Che, S.; Reinhardt, S.; et al. AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020; pp 922−936.
(34) Sussman, J. L.; Lin, D.; Jiang, J.; Manning, N. O.; Prilusky, J.; Ritter, O.; Abola, E. E. Protein Data Bank (PDB): database of three-dimensional structural information of biological macromolecules. *Acta Crystallogr., Sect. D: Biol. Crystallogr.* **1998**, *54*, 1078−1084.
(35) Rakshit, A.; Bandyopadhyay, P.; Heindel, J. P.; Xantheas, S. S. Atlas of putative minima and low-lying energy networks of water clusters n= 3−25. *J. Chem. Phys.* **2019**, *151*, 214307.
(36) Pinheiro, G. A.; Mucelini, J.; Soares, M. D.; Prati, R. C.; Da Silva, J. L.; Quiles, M. G. Machine learning prediction of nine molecular properties based on the SMILES representation of the QM9 quantum-chemistry dataset. *J. Phys. Chem. A* **2020**, *124*, 9854−9866.
(37) Chanussot, L.; Das, A.; Goyal, S.; Lavril, T.; Shuaibi, M.; Riviere, M.; Tran, K.; Heras-Domingo, J.; Ho, C.; Hu, W.; et al. Open catalyst 2020 (OC20) dataset and community challenges. *ACS Catal.* **2021**, *11*, 6059−6072.
(38) Jouppi, N. P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, 2017; pp 1−12.
(39) Emani, M.; Vishwanath, V.; Adams, C.; Papka, M. E.; Stevens, R.; Florescu, L.; Jairath, S.; Liu, W.; Nama, T.; Sujeeth, A. Accelerating scientific applications with sambanova reconfigurable dataflow architecture. *Comput. Sci. Eng.* **2021**, *23*, 114−119.
(40) Jia, Z.; Tillman, B.; Maggioni, M.; Scarpazza, D. P. Dissecting the graphcore ipu architecture via microbenchmarking. *arXiv* **2019**, arXiv:1912.03413.
(41) Schütt, K. T.; Kessel, P.; Gastegger, M.; Nicoli, K. A.; Tkatchenko, A.; Müller, K. R. SchNetPack: A Deep Learning Toolbox For Atomistic Systems. *J. Chem. Theory Comput.* **2019**, *15*, 448−455.
(42) Joshi, R. P.; Gebauer, N. W.; Bontha, M.; Khazaieli, M.; James, R. M.; Brown, J. B.; Kumar, N. 3D-Scaffold: A deep learning framework to generate 3d coordinates of drug-like molecules with desired scaffolds. *J. Phys. Chem. B* **2021**, *125*, 12166−12176.
(43) Jha, D.; Gupta, V.; Ward, L.; Yang, Z.; Wolverton, C.; Foster, I.; Liao, W.-k.; Choudhary, A.; Agrawal, A. Enabling deeper learning on big data for materials informatics applications. *Sci. Rep.* **2021**, *11*, 4244.
(44) Batatia, I.; Kovacs, D. P.; Simm, G.; Ortner, C.; Csányi, G. MACE: Higher order equivariant message passing neural networks for fast and accurate force fields. *Advances in Neural Information Processing Systems*; Curran Associates, Inc., 2022; Vol. 35, pp 11423−11436.
(45) Ramakrishnan, R.; Dral, P. O.; Rupp, M.; von Lilienfeld, O. A. Quantum chemistry structures and properties of 134 kilo molecules. *Sci. Data* **2014**, *1*, 140022.
(46) Heindel, J. P.; Xantheas, S. S. *Database of Water Cluster Minima*, 2019. https://sites.uw.edu/wdbase/.
(47) Choudhury, S.; Pope, J.; Ward, L.; Foster, I.; Schwarting, M.; Blaiszik, B.; Heindel, J.; Xantheas, S. HydroNet: Benchmark Tasks for Preserving Structural Motifs and Long-range Interactions in Predictive and Generative Models. *arXiv* **2020**, arXiv:2012.00131.
(48) PopTorch, G. *PopTorch: PyTorch Integration for the Graphcore IPU*, 2022. https://github.com/graphcore/poptorch.
(49) Fey, M.; Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. *arXiv* **2019**, arXiv:1903.02428.
(50) Wang, M.; et al. Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds* 2019.

https://doi.org/10.1021/acs.jcim.3c01312
J. Chem. Inf. Model. XXXX, XXX, XXX−XXX

(51) Korte, B.; Vygen, J. *Combinatorial Optimization; Algorithms and Combinatorics*; Springer Berlin Heidelberg: Berlin, Heidelberg, 2012; Vol. *21*.
(52) Krell, M. M.; Kosec, M.; Perez, S. P.; Fitzgibbon, A. Efficient Sequence Packing without Cross-contamination: Accelerating Large Language Models without Impacting Performance. *arXiv* **2021**, arXiv:2107.02027.
(53) Johnson, D. S. *Near-optimal Bin Packing Algorithms*. Ph.D. Thesis, Massachusetts Institute of Technology, 1973.
(54) Lee, C. C.; Lee, D. T. A Simple On-Line Bin-Packing Algorithm. *J. ACM* **1985**, *32*, 562−572.
(55) Johnson, D. S.; Garey, M. R. A 7160 theorem for bin packing. *J. Complex* **1985**, *1*, 65−106.
(56) Yue, M.; Zhang, L. A simple proof of the inequality $MFFD(L) \leq 71/60OPT(L) + 1$, L for the MFFD bin-packing algorithm. *Acta Math. Appl. Sin.* **1995**, *11*, 318−330.
(57) Bro, R.; De Jong, S. A fast non-negativity-constrained least squares algorithm. *J. Chemom.* **1997**, *11*, 393−401.
(58) Hu, W.; Liu, B.; Gomes, J.; Zitnik, M.; Liang, P.; Pande, V.; Leskovec, J. Strategies for Pre-training Graph Neural Networks. In *International Conference on Learning Representations*, 2020. arXiv:1905.12265.
(59) Ahmad, W.; Simon, E.; Chithrananda, S.; Grand, G.; Ramsundar, B. ChemBERTa-2: Towards Chemical Foundation Models. *arXiv* **2022**, arXiv:2209.01712.
(60) Horawalavithana, S.; Ayton, E.; Sharma, S.; Howland, S.; Subramanian, M.; Vasquez, S.; Cosbey, R.; Glenski, M.; Volkova, S. Foundation models of scientific knowledge for chemistry: Opportunities, challenges and lessons learned. In *Proceedings of BigScience Episode# 5−Workshop on Challenges & Perspectives in Creating Large Language Models*, 2022, pp 160−172.
(61) Kuenneth, C.; Ramprasad, R. polyBERT: a chemical language model to enable fully machine-driven ultrafast polymer informatics. *Nat. Commun.* **2023**, *14*, 4099.
(62) Chen, C.; Ong, S. P. A universal graph deep learning interatomic potential for the periodic table. *Nat. Comput. Sci.* **2022**, *2*, 718−728.
(63) Takamoto, S.; Shinagawa, C.; Motoki, D.; Nakago, K.; Li, W.; Kurata, I.; Watanabe, T.; Yayama, Y.; Iriguchi, H.; Asano, Y.; et al. Towards universal neural network potential for material discovery applicable to arbitrary combination of 45 elements. *Nat. Commun.* **2022**, *13*, 2991.
(64) Deng, B.; Zhong, P.; Jun, K.; Riebesell, J.; Han, K.; Bartel, C. J.; Ceder, G. CHGNet as a pretrained universal neural network potential for charge-informed atomistic modelling. *Nat. Mach. Intell.* **2023**, *5*, 1031−1041.
(65) Chmiela, S.; Tkatchenko, A.; Sauceda, H. E.; Poltavsky, I.; Schütt, K. T.; Müller, K. R. Machine learning of accurate energy-conserving molecular force fields. *Sci. Adv.* **2017**, *3*, No. e1603015.
(66) Babin, V.; Leforestier, C.; Paesani, F. Development of a "First Principles" Water Potential with Flexible Monomers: Dimer Potential Energy Surface, VRT Spectrum, and Second Virial Coefficient. *J. Chem. Theory Comput.* **2013**, *9*, 5395−5403.
(67) Babin, V.; Medders, G. R.; Paesani, F. Development of a "First Principles" Water Potential with Flexible Monomers. II: Trimer Potential Energy Surface, Third Virial Coefficient, and Small Clusters. *J. Chem. Theory Comput.* **2014**, *10*, 1599−1607.
(68) Krizhevsky, A.; Sutskever, I.; Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*; Curran Associates, Inc., 2012.
(69) Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; Polosukhin, I. Attention Is All You Need. *arXiv* **2017**, arXiv:1706.03762.

https://doi.org/10.1021/acs.jcim.3c01312
J. Chem. Inf. Model. XXXX, XXX, XXX−XXX