Original Research Article

Exploring the Transfer of Neural Network Potentials with Tight Binding Models of Conjugated Systems

Mia Moser Gitter[1] and David Yaron[2]

[1]*Le Lycée Français de Los Angeles, Los Angeles, California, United States;*
[2]*Department of Chemistry, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, United States*

ABSTRACT

Neural network potentials (NNPs) have achieved impressive accuracy on local, short-range chemical environments, yet their ability to capture long-range electronic effects remains unclear. We investigate this question using a controlled testbed for electron delocalization: 1D tight-binding chains with 2–10 atoms. We generated synthetic datasets by drawing nearest-neighbor coupling values from a narrow normal distribution and determining the resulting ground-state energies using Hamiltonian diagonalization. We benchmark two architectures: (1) a "full chain" feed-forward MLP that ingests the entire coupling vector and directly predicts the total energy, and (2) a Behler–Parrinello–style "atom environment" model that sums per-atom energies from local windows (sizes 1–3). Across chain lengths, the full-chain model consistently attains lower error and cleaner predicted-vs-true alignments, though accuracy degrades with system size for all models. For 10-atom chains, mean errors are 0.0279 (full-chain) versus 0.0613, 0.0524, and 0.0417 for window sizes 1, 2, and 3, respectively. Loss curves show stable optimization with limited overfitting on small and mid-sized systems. These results indicate that even simple global-feature MLPs can learn delocalization effects that challenge purely local descriptors. Key limitations include the idealized 1D tight-binding ground truth, nearest-neighbor couplings only, and the absence of symmetry/equivariance constraints, which limit transferability to realistic 3D chemistry. Nonetheless, the findings motivate combining global receptive fields (or longer-range descriptors) with physics-aware architectures to improve generalization in delocalized electronic systems.

Keywords: Neural Network Potential (NNP); Molecular Energy Prediction; Tight-Binding Model (TB); Feedforward Neural Network (FNN); Potential Energy Surface (PES); Transferability; Data- Driven Quantum Chemistry

INTRODUCTION

Although the original mathematical model of a neural network (NN) was first outlined in the early 1940s, its first applications in chemistry occurred in 1995 with the study by Blank *et al.* (1). Since then, as news of its efficacy and accuracy spread, the frequency of its use within the scientific field has only increased.

Corresponding author: David Yaron, E-mail: yaron@cmu.edu.

Accepted October 23, 2025
https://doi.org/10.70251/HYJR2348.366990

This swift growth in popularity can be attributed to Behler and Parrinello's 2007 letter outlining their newer, faster, and more cost effective version of reproducing density-functional theory (DFT) potential-energy surfaces, highlighting the true potential of these prediction methods (2). However, a substantial limitation of their study was that it was applied only to bulk silicon, something that did not show the versatility of these potentials. Ten years later, Isayev, Smith, and Roitberg extended the work of Behler and Parrinello to the ANI-1 potential, which made succesful predictions for organic molecules (3). Since their respective publications, the vast majority of the work in this field has been related to at least one of them, a fact that shows how pivotal both studies are.

While the capabilities of existing potential cannot be understated, there are glaring limitations when it comes to their application, specifically in chemistry. For instance, the use of DFT data has been shown to have questionable accuracy in some of its predictions. Another challenge is the use of this approach to describe systems with extensive electron delocalization. In this context, transferability refers to a model's ability to generalize beyond the systems included in its training set to predict energies accurately across different chain lengths, atomic compositions, or degrees of electron delocalization. The features input to the neural network are short range, so it is not clear how well they can handle systems with long-range electronic effects.

In this project, we explore the extent to which neural network models can capture effects from electron delocalization. We first generate synthetic data using a simple tight binding model on a 1D atomic chain, creating an efficient test system for electron delocalization. We then fit this data using a neural network potential, allowing for systematic tests of the limitations of these potentials on delocalized systems. The models we explore include one that uses a feature vector that describes the entire molecule. We then compare this to a model that, like a Behler-Parinello network, using a local atom environment vector as its input features. This allows us to explore the extent to which local features can capture electron delocalization that goes beyond the length of the input feature.

LITERATURE REVIEW

One of the earliest and most influential studies surrounding neural network potentials and their applications to chemistry is that of Behler and Parrinello. While neural nets had been used before on potential energy surfaces (PESs), they came with many limitations, one of which being the amount of training data needed at the time (1). Behler and Parrinello's paper managed to overcome these challenges, creating a new model to reproduce DFT data at a much lower cost (2). Their work was limited, however, in the sense that they only applied their model to silicon. In solid form, this element includes relatively short-ranged bonds and localized electrons, meaning the model was not able to sustain systems with energy delocalization. A few years later, Artrith and his colleagues applied this method to zinc oxide, demonstrating the fact that, even though they required further steps, neural networks could make predictions in multi-component systems, emphasizing its versatility (4). This, however, led to there being higher costs to set up the neural network. Their work also discusses the idea that these systems had limited transferability to structures different from those present in the training set, meaning that the model performs well on the training data but struggles with more complex systems. This severely restricts the capabilities of the model, leading to an inability to reliably predict results for larger molecules, different atom types, unseen coupling patterns, and quantum chemical systems.

In 2011, Behler wrote a paper analyzing the (then) present status of neural network potentials in chemistry. Here, he provides a list of advantages, as well as disadvantages, that come from working with neural networks. In terms of the former, his list includes several important factors that highlight their efficacy: accuracy, improbability, generality, reactivity, dimensionality, human effort, transferability, efficiency, computational cost, and derivatives (5). These points all emphasize either the precision of the model or the ease of its use. Regarding the disadvantages, he outlines the demanding nature of the construction, the at times low quality results, and the complicated systems. This gives further insight into the complexity of neural networks as a tool. If used correctly, they have the potential to be very accurate and reliable predictors for energy prediction. If used incorrectly, they can be extremely unreliable, wasting precious time and resources.

Two years after the publication of that last paper, in 2013, Bart´ok and his colleagues wrote an article testing the performance of various types of various descriptors to analyze the accuracy and effectiveness of preexisting methods. In the end, their findings highlight even more limitations for neural networks, with the

most glaring one being that out of all the models they tested (including that of Behler and Parrinello), none could make accurate predictions for Si clusters with more than 13 atoms, bringing the inaccuracy of these systems to the forefront (6). While the findings of this study highlight the limitations of these models, it also shows that these hurdles can be overcome. The authors themselves created an atomic environment feature they call Smooth Overlap of Atomic Positions (SOAP), aiding in creating more accurate and more robust PESs. That same year, Pirdashti *et al*. wrote a comprehensive review of neural networks' applications to chemical engineering (7). The authors outline the use of these systems within this branch of engineering, discussing different methods and applications. Their work underlines the diverse ways in which these systems can be used in chemistry and its related fields of study, emphasizing their importance, effectiveness, and practicality.

In 2014, Behler published a review discussing the representation of potential energy surfaces with high-dimensional NNPs (8). In it, he highlights the issues these models have when forming a prediction on systems that contain larger amounts of different chemical elements. Behler also provides various mathematical formulas to illustrate the process of a neural network. Overall, this paper serves as a comprehensive overview of representing PESs in high-dimensional NNPs. He continues on this path, publishing another review a year later that is meant to serve as a tutorial for the development of these systems (9).

In 2017, Smith, Isayev and Roitberg introduce their ANI-1, a new NNP based on the one developed by Behler and Parrinello and designed specifically for transferability (3). After running their code, the authors found that their potential was far more accurate than anything running off DFT (i.e., the Behler Parrinello model), highlighting its potential in the field. The next year, Gossett, Isayev, and their colleagues developed AFLOW-ML to simplify the abstraction on top of predictive models (10). Essentially, when added to any code base, it reduces the amount of code and recourse necessary to run a successful, neural network potential, allowing for facilitated use of machine learning models. In doing this, their hopes were to accelerate the widespread adoption of these prediction methods within materials science, engineering, and development.

Since then, machine learning has infiltrated scientific research, being used for everything from drug discovery to nanoparticle identificatio (11, 12). Within the field of chemistry alone, this surge has resulted in the expansion and development of these NNPs, with different models being altered to accommodate increased complexity. However, despite countless improvements in performance and usability, most models still struggle with electron delocalization. This is due to the fact that most architectures rely on smaller local atomic environments, leading to the model to be unable to capture nonlocal quantum effects that arise in these cases. Therefore, questions remain about how well these simple, local models can generalize in such cases. Our work contributes to this conversation by testing a minimal local model on systems with increasing electron delocalization, evaluating its performance, and comparing it to that of a preexisting, popular model.

METHODS AND MATERIALS

Generation of Synthetic Data

In this project, we construct one-dimensional tight-binding (TB) chains with open boundary conditions, containing 2 to 10 atoms, to generate synthetic datasets for neural network training. To generate the synthetic data needed for this research, we create code which resulted in a dataset of input-output pairs: the input is a vector of coupling values between adjacent atoms, while the output is the total energy of the system calculated using a tight-binding model. Each system is described by a Hamiltonian H of dimension $N_{\text{atom}} \times N_{\text{atom}}$, where $H_{i,i+1} = H_{i+1,i} = \beta_i$ and all other elements are zero. These β_i values represent nearest-neighbor couplings between atoms. We use open boundary conditions, meaning no coupling connects the first and last atom. Here, we utilize a for loop that works with a chain of a given number of atoms (i.e., 2 to 10), ensuring that each one is represented in the results. For each chain, we also outline "num couplings" as the connections between neighboring atoms and "num electrons" as the total number of valence electrons. For the latter, we kept it simple, giving each atom 1 electron. The number of electrons N_e equals the number of atoms ($N_e = N_{\text{atom}}$). The ground-state energy E_0 is obtained by diagonalizing the Hamiltonian with numpy.linalg.eigvalsh, sorting its eigenvalues in ascending order, and filling the lowest-energy orbitals. If N_e is even, the lowest $N_e/2$ orbitals are doubly occupied; if odd, the next orbital is singly occupied. The total energy is computed as the sim of these occupied eigenvalues.

Next, we generate random input vectors (coupling values) using 10,000 samples per chain length, each

drawn from a normal distribution with mean –1.0 and standard deviation 0.05. This distribution simulates small variations in electronic bond strengths. They are meant to represent the strength of electronic couplings between neighboring atoms, simulating a realistic but slightly disordered physical system.

After this, we build a Hamiltonian matrix for each coupling vector c. The matrix is square, with its size being the square of a given number of atoms. Within its values, each nonzero element represents a coupling between atoms. Additionally, the matrix is symmetric and only has non-zero values on the first-off diagonals. The following step is to compute the total energy of the systems for each Hamiltonian. This diagonalizes energy and assures the electrons occupy the lowest-energy orbitals. The steps in this part are different for even and odd numbers respectively: in the former, the lowest N/2 orbitals are doubly occupied; in the latter, the next orbital is singly occupied. Both have to be taken into consideration in the code for risk of it not working properly for certain chain lengths. Lastly, we sum the eigenvalues accordingly to get the value of the total ground state energy.

The final step in the generation of synthetic data is to normalize the target energy. To do so, we use standardization, transforming each value in the target vector. This is done to stabilize training, helping the optimizer behave more smoothly, and to ensure the loss magnitude is consistent, easier to track, and more meaningful. It also allows for generalization across systems, something that makes the model outputs comparable. Normalization is performed separately for each chain length, using the mean and standard deviation of that subset. This ensures consistent scaling across systems of different sizes and stabilizes the optimizer's behavior.

All random seeds were fixed for reproducibility (numpy.random.seed(0) and torch.manual_seed(0)). Each dataset contains 10,000 randomly generated coupling configurations per chain size.

Form of the neural network potential models

We implemented two neural-network potentials (NNPs) in PyTorch 2.0 to test how feature locality affects energy prediction accuracy: a global full-chain model and a local atom-environment model analogous to the Behler-Parrinello architecture.

The full-chain model is a fully connected feed-forward multilayer perceptron (MLP) that receives the entire coupling vector $\beta \in \mathbb{R}^{N\text{atom-1}}$ as input and predicts a single scalar energy. Its architecture is: Linear($N_{\text{atom}} - 1$, 64) → ReLU → Linear(64, 64) → ReLU → Linear(64, 1). We use PyTorch's default weight initialization and the ReLU activation function for both hidden layers

To begin, we define and initialize a simple feed-forward MLP that predicts the total energy given the list of couplings between atoms. We call this the *full-chain model* because its predictions are based on features that describe the entire chain. The first step is to define a custom model class, EnergyNN. This class specifies fixed architecture by inheriting PyTorch's base module nn.Module, setting up a sequential network of layers, and storing the network. We then define how input data flows through the model: an input tensor x, consisting of coupling vectors, is passed through the network, and the final output is a single scalar prediction per sample. At initialization, the model accepts input but contains randomly assigned weights that are updated during training. The next step is to create an actual instance of the model, specifying how many input features it should expect.

The model parameters are optimized with the Adam optimizer (learning rate = 1×10^{-3}) using the mean-squared error (MSE) loss function. Training is performed for 500 epochs on the full training ser (batch = all samples, no mini-batching). Random seeds are fixed (NumPy = 0, PyTorch = 0) for reproducibility.

We also construct an *atom-environment* model in PyTorch, analogous to a potential of the Behler-Parrinello network. The window size, N_{window}, specifies how far away couplings are included. Each atom's environment is represented by the couplings within a local 'window' of width w, meaning the w nearest couplings to the left and right are included. For example, when w = 2, the input vector for an atom has four values (two on each side). Atoms near chain ends are zero-padded to maintain constant input size. The atomic sub-network has the same structure as the full-chain model (Linear(2w, 64) → ReLU → Linear(64, 64) → ReLU → Linear(64, 1)) and outputs one energy per atom. The total molecular energy is obtained by summing these atomic contributions. We test three window sizes (w = 1, 2, 3) to analyze how increasing local context affects accuracy.

In this approach, the model predicts the energy of each atom in the chain and then sums these values to get the total energy. Unlike models such as ANI-1, the environment of an atom here is described using the couplings surrounding that atom as inputs to a simple feed-forward neural network. The window size, N_{window},

specifies how far away couplings are included. For a chain with N_{atom} atoms, the couplings are given by a list of $N_{atom} - 1$ values indicating the bonds between adjacent atoms. When $N_{window} = 2$, the input to the neural network for a given atom has four values: the two couplings to the left and the two couplings to the right. To handle atoms near the ends of the chain, missing couplings are replaced with zeros (i.e., the input vector is padded with zeros).

Both models are trained independently for each chain length. Data are randomly split 80% train / 20% test using train_test_split(random_state = 0). The loss at each epoch is computed on both splits to monitor over- or under-fitting.

Training the neural network potential

After defining both neural-network architectures and generating the datasets, we train each model independently for every chain length using consistent hyperparameters and evaluation criteria. The loss function is the Mean Squared Error (MSE), defined as:

$$MSE = \frac{1}{N}\sum_i (\hat{y}_i - y_i)^2$$

where $\hat{y}_i$ and y_i are the predicted and true standardized energies. We compute this loss on both the training and test sets at every epoch to monitor convergence and detect overfitting.

We use the Adam optimizer (learning rate = 1×10^{-3}) implemented in PyTorch 2.0. Training runs for 500 epochs with full-batch gradient descent (the entire training set per iteration). No regularization (dropout or weight decay) is applied. Model parameters are initialized with PyTorch's default Xavier uniform scheme. Each dataset is split into 80% training and 20% testing subsets using train_test_split(random_state = 0) to ensure reproducibility."

During each epoch, the training phase performs a forward pass, computes the MSE loss, back-propagates gradients, and updates parameters with Adam. The evaluation phase computes the same loss on the held-out test set (without gradient updates) to track generalization. All loss values are stored for later visualization of learning curves. The training phase itself is comprised of two parts, the first of which being the forward pass on training data. Here, the model takes input, processes it through the network, and makes a prediction. We also compute the MSE between predictions and true energy values, producing a scalar loss value representing the model's average squared error. The next step is back propagation, in which we calculate the gradient loss with respect to every parameter in the model. The Adam optimizer can now update the weights using the gradients from the back propagation, acting as the learning step.

The evaluation phase assesses generalization by computing the test-set MSE and monitoring divergence between train and test losses. After this step, we can make the final predictions, which we plot against the true energy to see how well the model learned the mapping.

After training, we evaluate model performance using the Root-Mean-Square Error (RMSE) between de-standardized predicted and true energies:

$$RMSE = \sqrt{\frac{1}{N}\sum_i (\hat{E}_{0,i} - E_{0,i})^2}$$

This metric is reported for each chain length to compare accuracy across system sizes and window settings.

Graphing the results

We visualize training behavior and model accuracy using Matplotlib 3.8, producing consistent figures for each atom-chain length and model type. We generate figures directly from arrays recorded during training. Each figure uses the stored lists of training and test losses per epoch and the final predicted and true (de-standardized) energies from the test set. We make it so that, for each atom system, there would be two graphs: one for training and test loss, the other for the predicted and true energy values.

The first panel in each figure shows the training and test Mean-Squared-Error (MSE) versus epoch (0–500 epochs). Blue lines represent training loss; orange lines represent test loss. We include gridlines and axis labels for consistency. These curves illustrate convergence rate and potential over- or under-fitting.

The second panel plots predicted energy ($\hat{E}$) versus true energy (E) for the test set. Points are shown as semi-transparent blue scatter dots to indicate density; the red dashed line $y = x$ marks perfect prediction. The spread around this line visualizes error magnitude, complementing the numerical RMSE reported in each figure.

A third summary plot displays the per-chain Root-Mean-Square-Error (RMSE) values computed on de-standardized energies. Each bar (or labeled point) corresponds to one chain length (2–10 atoms). Error values are expected to increase with system size, highlighting the growing difficulty of modeling long-range delocalization.

For the atom-environment model, we produce separate composite figures for window sizes 1, 2, and 3. Each composite figure contains nine subplots (2–10 atoms), showing predicted vs true energy scatter plots annotated with their RMSE values. This grid layout enables quick visual comparison of accuracy across both chain length and window size.

All plots are generated using Matplotlib's default style to ensure reproducibility. Figure sizes, color palettes, and axis ranges are standardized across models to facilitate direct comparison. Random seeds and data splits are identical to those used during training.

RESULTS

Full Chain Model

For the full chain model, we obtain both of the aforementioned graphs. When discussing them both side by side, we choose to go up to an 8-atom chain for simplicity. For the system made up of 2 atoms, the two functions appear to overlap throughout most of the graph (Figure 1). The exception to that occurs from around 0 to 50 epoch. Both have a steep decline from 0 to around 150 epoch, where the MSE loss for both goes from around 1 to 0. From that point on, the functions plateau. In the second graph, the scatter plot is heavily clustered around the diagonal. In terms of the training and test loss curves, the results for a 3-atom system appear similar to that of a 2-atom system. The train and test functions overlap throughout the graph, almost more so than the other system. The decent for both functions is less steep: the MSE only becomes 0 around 250 epochs. There are some slight differences present in the second graph. While the scatter plot is heavily clustered around the diagonal, the position of certain data points appears more dispersed, especially as the true energy and the predicted energy grow higher. Both graphs in 4 atom system more closely resemble those of the 2-atom system. In the first, the test and train functions overlap for most of the epoch values, both having relatively steep declines until the MSE value reaches 0 at around 250. In the second graph, the scatter plot looks like a better way to cluster around the diagonal better than in the 3-atom system. The results for the 5-atom system differ slightly from those of the previous chains. In the first graph, the gap between the two functions is larger than in the other systems and lasts until around 150 epochs. The rest of the two functions overlap. In the second graph, the scatter plot resembles that of a 3-atom system. Although the majority of the data points also follow the dashed line closely, there are still some outliers.

In the first graph of the 6-atom system, the test function and the train function overlap at around 230

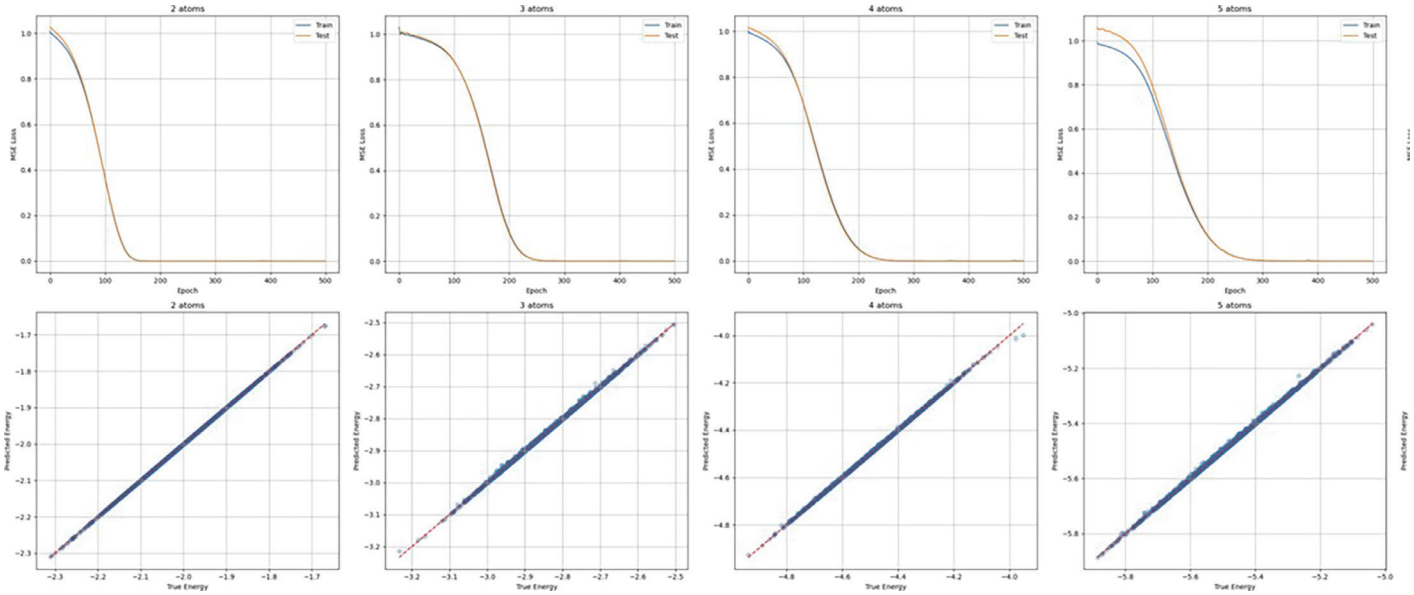

Figure 1. Training and prediction results for 2–5 atom chains using the Full Chain Model. Each plot shows the training and test loss (top row) and predicted vs. true en- ergy values (bottom row) for atom chains of increasing length. The close overlap between predicted and true energy lines and the low loss values indicate that the model accurately learns the ground-state energy for smaller systems with minimal overfitting.

epoch (Figure 2). For both functions, there are some slight fluctuations at around 250 epoch and 310 epoch. The second graph shows that, like for most of the other systems, the scatter plot is clustered around the dashed line. For the first graph in the 7-atom chain, the two functions are much more aligned than they were for the last two systems. The two functions overlap at around 150 epoch, both descending relatively constantly but not as steep as for previous systems. The second graph shows that the scatter plot is essentially aligned with the diagonal, with some minor discrepancies. The first graph in the 8-atom chain sees the two functions overlap at around 200 epoch. Unlike the other systems, however, at around 330 epoch and 420 epoch, there are two very slight spikes in both functions. In the second graph, the data points in the scatter plot seem to be clustered around the middle to lower part of the dashed line, leaving a space in the upper right corner.

In terms of the error numbers for each atom chain, we went up to a 10-atom chain to better illustrate the changes that occurred (Figure 3). For almost all the graphs depicted in the figure shown above, the error numbers for the results of the full chain model seem to increase with the atom chain. The only exceptions to the trend are the error numbers of the 4 and 5 atom chains: the number of the 4-atom chain (0.0024) is almost equal to that of the 3-atom chain (0.0025), while the number of the 5-atom chain (0.0054) is bigger than that of the 6-atom chain (0.0048).

Atom Environment model

In terms of the atom environment model, we studied the predicted and true energy graph, along with the error number. The latter is what we will use to compare this model with the full chain model. Since it is one of the major elements within the atom environment model, we will also be looking at the results for window sizes of 1 through 3.

For the smallest window size, the error numbers of the 2, 3, and 4 atom chains appear to increase with the number of atoms present (0.0004, 0.0026, and 0.0055 respectively) (Figure 4). This trend continues for the

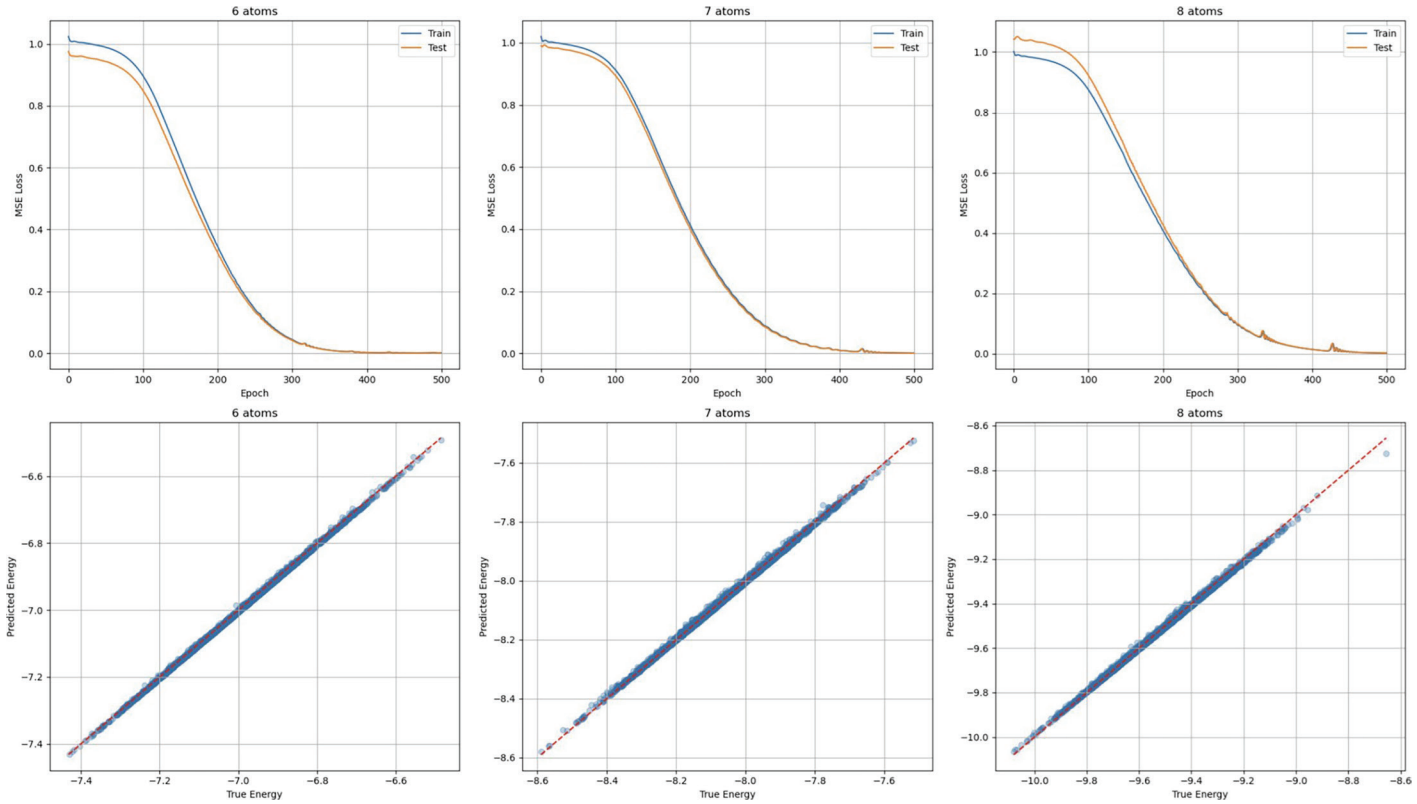

Figure 2. Training and prediction results for 6–8 atom chains using the Full Chain Model. Similar to Figure 1, these plots display the loss curves (top) and predicted versus true energy correlations (bottom). As chain length increases, the error values rise slightly, suggesting that larger systems are more challenging for the model to learn while maintaining stability in predictions.

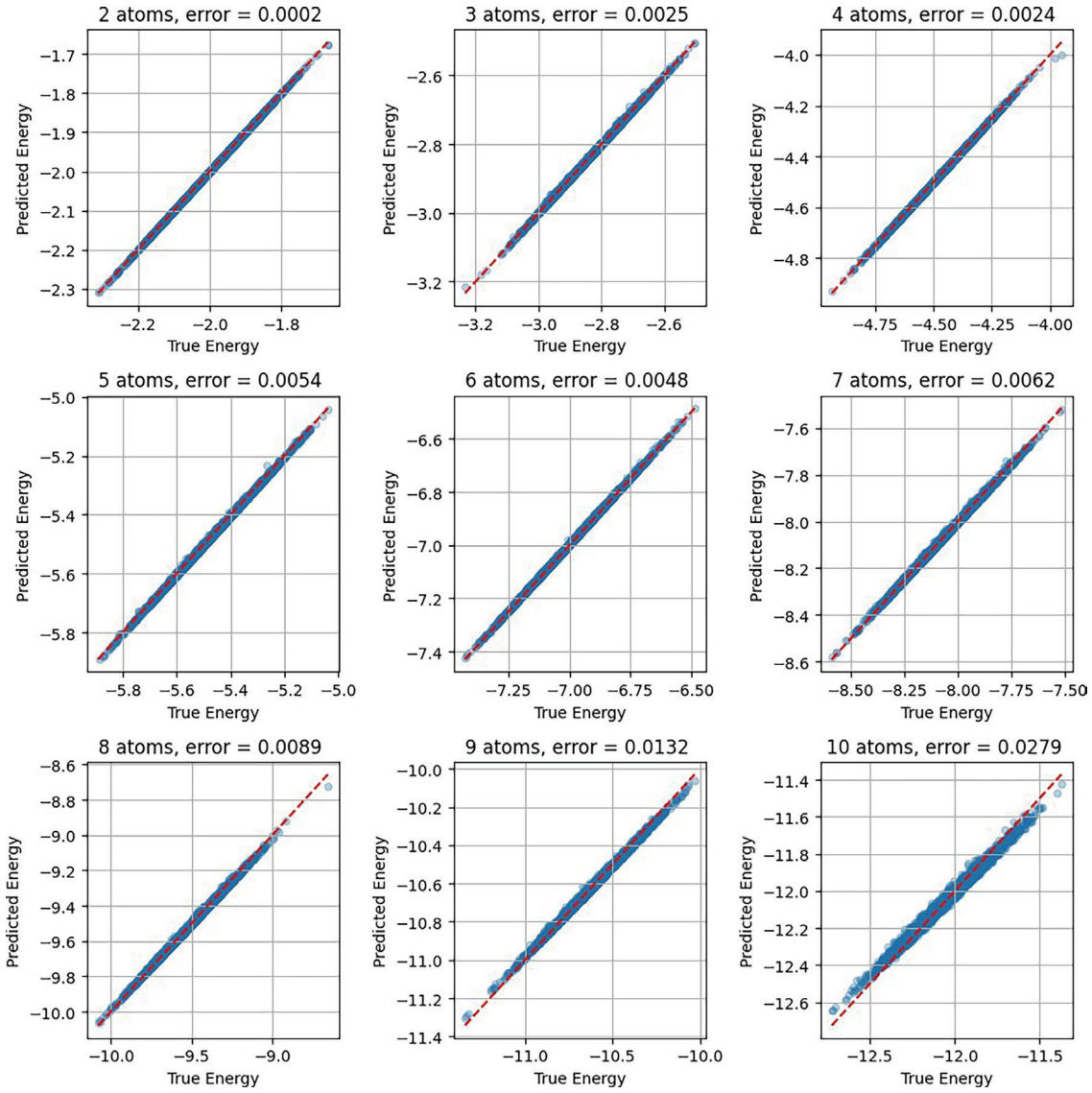

Figure 3. Predicted versus true energy values for the Full Chain Model across 2–10 atom chains. Each panel compares predicted energy (y-axis) to true energy (x-axis) for a specific chain size, with the red dashed line representing the ideal 1:1 correlation. The model maintains strong predictive accuracy, though errors increase modestly for larger atom systems, consistent with delocalization effects.

next three graphs, representing chains of 5, 6, and 7 atoms (0.0103, 0.0255, and 0.0112 respectively). The error numbers for the systems consisting of 8, 9, and 10 atoms also increase as the chains grow larger (0.0489, 0.0164, and 0.0613 respectively).

When the window is set to 2, the results of the first 6 graphs (2 to 7 atom chains) are relatively similar, with little variation between the error numbers of neighboring graphs (Figure 5). For instance, the difference of the error numbers of the 3 and the 3-atom chain is only 0.0007 (= 0.0021 - 0.0014). The number does, however, still increase as the number of atoms grows, with the system seeming to have some slight trouble with the 5-atom chain, with it having the highest error number for the first 6 graphs (0.0102). The final three graphs, representing the 8, 9, and 10 atom chains, differ greatly from the others in the figure due to their high error numbers (0.0349, 0.0227, 0.0524 respectively).

Like in the two previous windows, when the window equals three, the error numbers increase along with the number of atoms for each given system (Figure 6). The results for the first four graphs (representing the 2 to

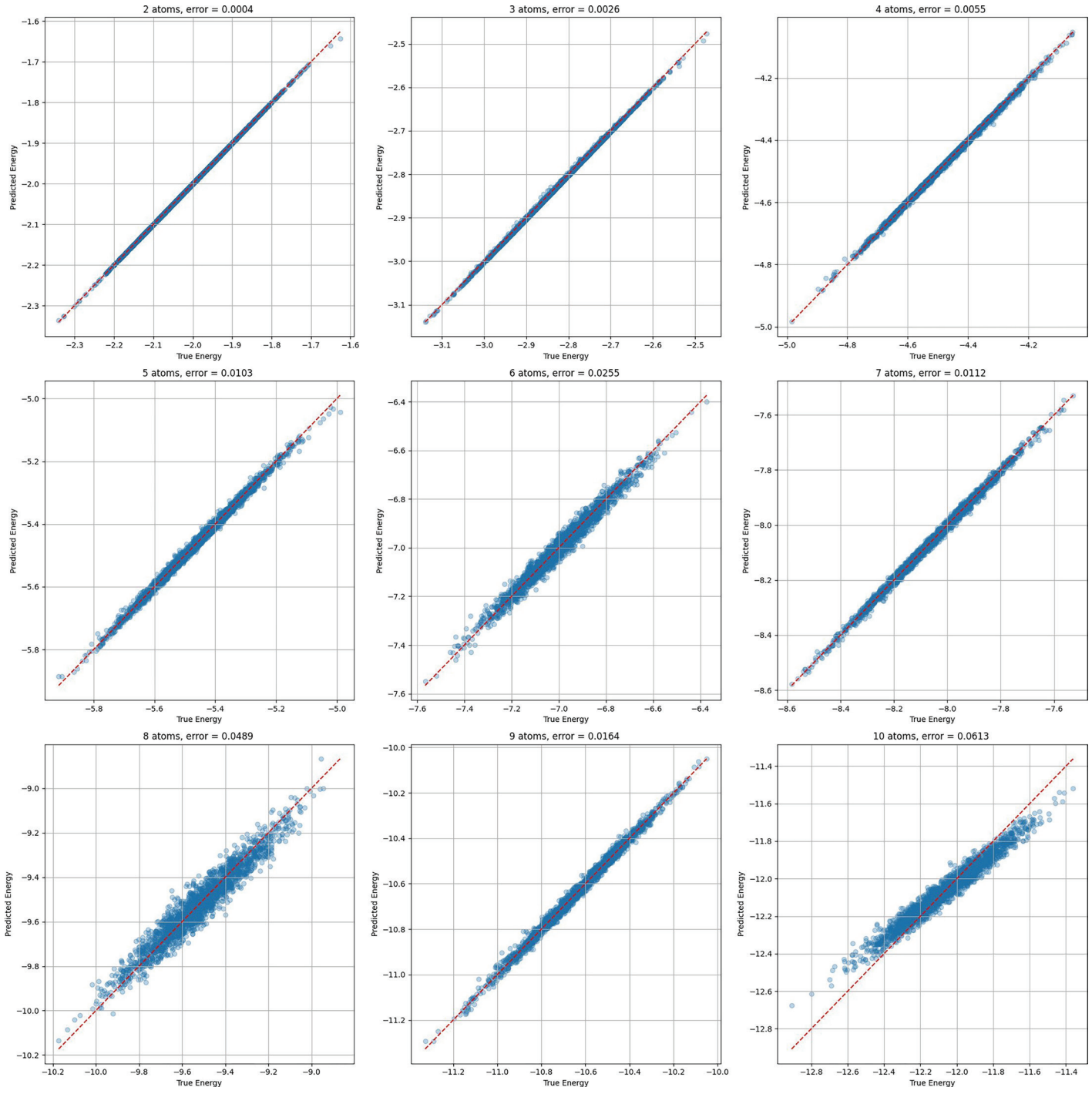

Figure 4. Predicted versus true energy values for the Atom Environment Model for when window size = 1. The scatter plots show predicted and true energies for atom chains from 2 to 10 atoms. Smaller window sizes limit the model's ability to account for neighboring atomic effects, leading to higher prediction errors as system size increases.

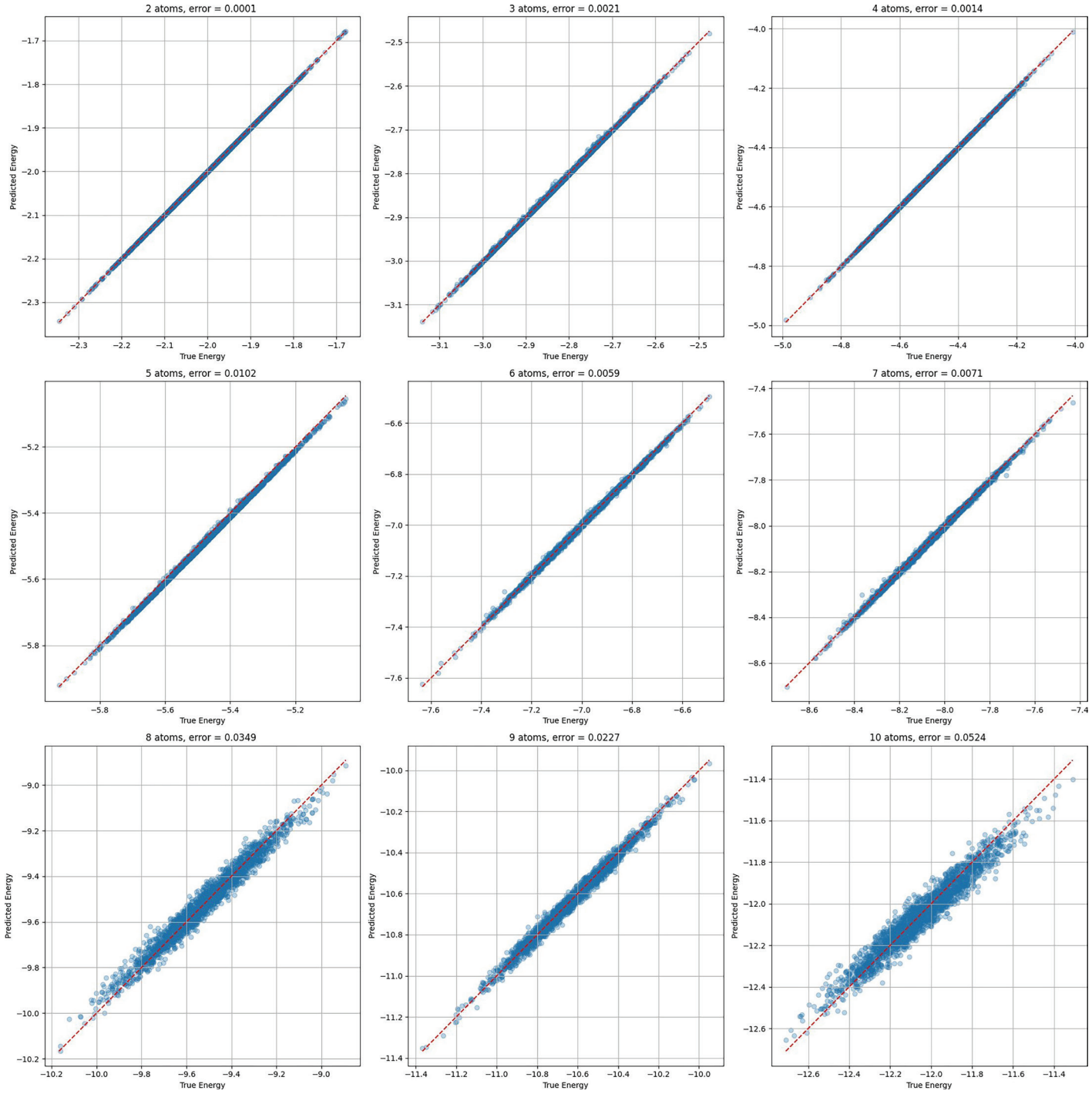

Figure 5. Predicted versus true energy values for the Atom Environment Model for when window size = 2. As the window size increases, the model incorporates more neighboring atomic information, improving accuracy for smaller chains. However, the pre- dictive performance still declines for systems beyond seven atoms, reflecting the limits of localized context in delocalized systems.

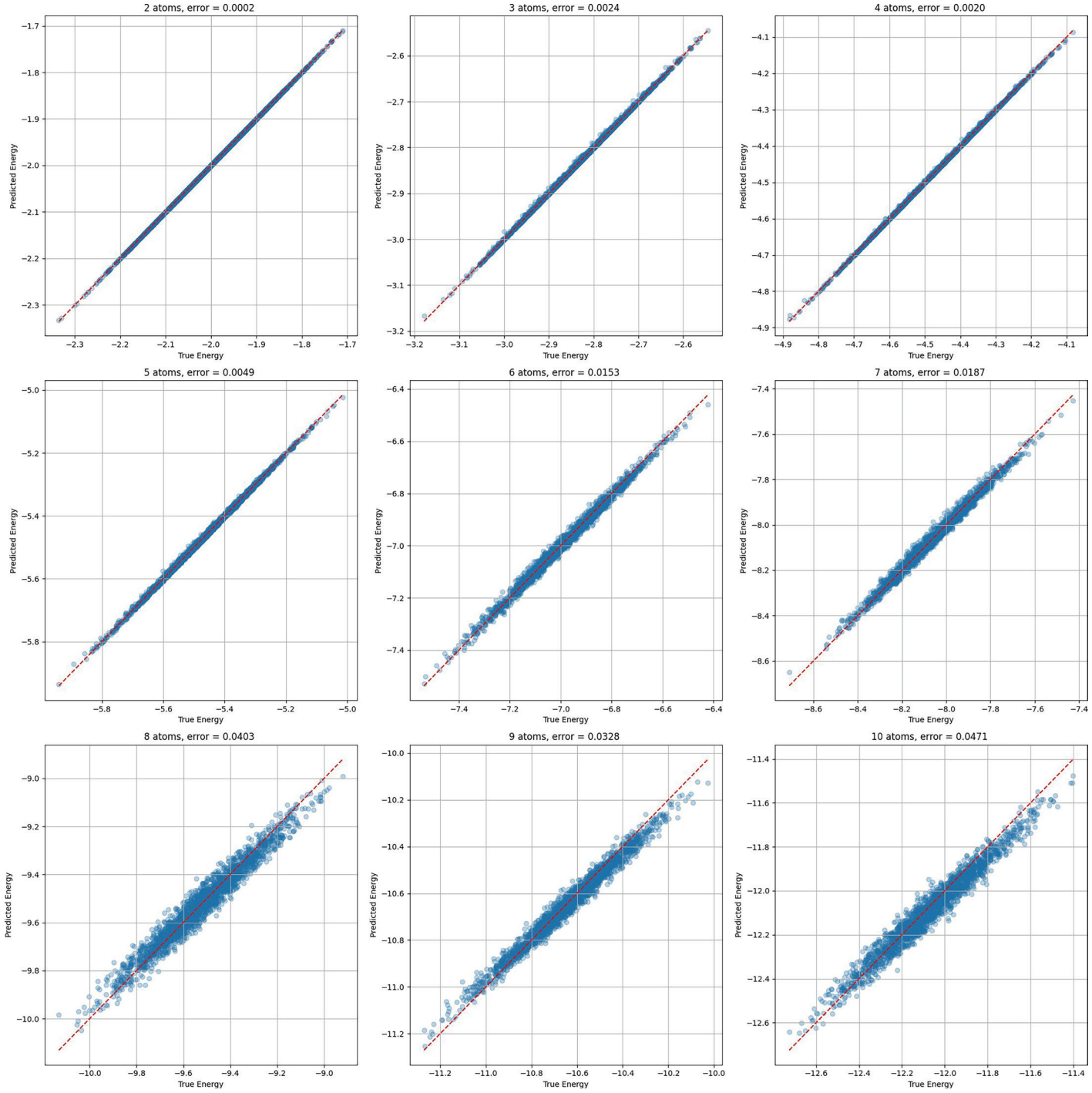

Figure 6. Predicted versus true energy values for the Atom Environment Model for when window size = 3. Including a larger local environment improves prediction ac- curacy for some systems but introduces more variability in larger atom chains. This suggests that increasing the window size does not uniformly enhance performance, highlighting the complexity of electron delocalization effects.

5 atom chains) appear similar, with none of the error numbers exceeding 0.0050 (0.0002, 0.0024, 0.0020, and 0.0049 respectively). The error numbers of the 6 and 7 atom chains, however, are both larger than 0.0100 (0.0153 and 0.0187 respectively), marking a notable increase in values. This jump occurs again for the error numbers in the 8, 9, and 10 chains, with their values being 0.0403, 0.0328, and 0.0471 respectively.

DISCUSSION

The results of the full chain model indicate that it can accurately make predictions in systems with delocalization. The training and test loss graphs for the first 7 systems (chain sizes of 2 to 8 atoms) show that there is little to no overfitting taking place, showing the model is working well with the amount of data that we gave it. Both the predicted/true energy graphs and the error numbers for each atom chain also show that this model could make relatively accurate predictions. The one downside to this model seems to be the fact that, as the system grows larger and consists of more atoms, it has a harder time learning and making predictions. This can be seen through the fact that the error number increases along with the system size. Additionally, in the training/test loss graph for the 8-atom chain, the slight spikes could indicate possible unstable training or overfitting, showing the minor difficulty in making predictions. This is to be expected, however, considering that this occurrence is common among most existing models. While it is by no means perfect, this model does appear to make accurate predictions and learn given the conditions we placed it in.

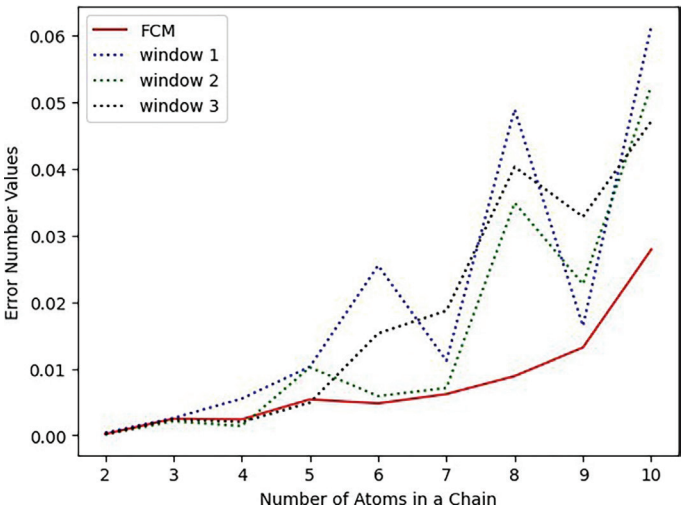

Figure 7. Comparison of prediction error across models and window sizes. The plot summarizes average error values for each atom chain size across the Full Chain Model (red) and Atom Environment Models with window sizes 1–3 (dotted lines). While all models show increasing error with system size, the Full Chain Model consistently achieves the lowest overall error, demonstrating superior scalability.

The Behler-Paranillo Model, or the atom environment model, has slightly different results. When analyzing all of the error numbers in each figure, it becomes clear that the model is able to make more accurate predictions when the window size increases. For instance, the error number for the 10-atom chain is equal to 0.0613 when the window size is equal to 1 while, when the window size is equal to 3, the error number is equal to 0.0417. Like the full chain model, the atom environment model has a harder time making predictions in larger systems, something that is evident for all three window sizes tested. Thus, this model can make relatively accurate predictions, even though there are still some considerable error numbers for larger systems.

Figure 7 illustrates how prediction accuracy varies with chain length across all models (Figure 7). As chain length increases, the task becomes significantly harder, consistent with the idea that electron delocalization is inherently difficult for these models to capture. The full chain model consistently outperforms the atom environment models, but even there, accuracy decreases as systems grow longer. For the atom environment models, one might expect that larger window sizes would steadily improve performance. However, even with a window size of three on a ten-atom chain, the results oscillate enough that it is not obvious which window size is better. Overall, these findings emphasize that electron delocalization remains a fundamental challenge for neural network potential.

Implications

This research demonstrates that neural networks can learn to predict energies of delocalized systems without one having to model long-range interactions. At times, these predictions can be more accurate than well-known and widely used models (i.e., Behler-Paranillo). The consistency of the full chain model's performance across the different atom chain lengths could suggest that even simple feed-forward architecture may be enough to make these predictions in a relatively accurate way. A successful learning model, however, still depends heavily on having access to large, high-quality datasets

in order to better extract relevant patterns from local input features. These findings support the idea that, if trained carefully, ML potential can be extended beyond purely local systems.

Limitations

One of the main flaws of our research was that caused by our choice of architecture. The neural network we utilize is a standard feed-forward model, one of the most popular type of networks. This widespread use can be attributed to its low computational costs, as well as how easy it is to be implemented. This model, however, does not take the spacial structure or symmetries of a molecular system into account. In both chemistry and physics, these characteristics are crucial: ignoring them can lead to unrealistic results and may require much more data for accurate prediction. Different, more chemically intuitive models, such as equivariant networks, do exist and might have managed to better capture the topology and structure of the data.

The input representation of our research is also limited. This is important as the neural net cannot learn a structure it cannot see. For simplicity's sake, we only feed the neural network raw hopping values without leveraging any alternative encoding, ignoring key characteristics like symmetry and physical position. The model we use also doesn't explicitly enforce physical invariance. These issues could cause increased sensitivity to irrelevant input permutations and poor generalization to differently ordered inputs.

Our training strategy also has a few constraints. Our use of a fixed training setup without implementing hyperparameter regularization. Normally, this is included to prevent overfitting and improve convergence. Omitting it may restrict the neural network's ability to generalize or converge optimally, something most apparent for the larger atom chains. Additionally, the lack of tuning may explain some of the inconsistencies present across different atom counts, the most notable being the slight wobbles in the loss curves.

About both the full chain model and the atom environment model, we only analyze smaller chains and window sizes. While this is done in order to not complicate the process and the results, it may lead to a rudimentary conclusion. As noted previously, the results of the atom environment model grow more and more accurate as the window size increases. Studying larger window sizes may be beneficial as it could further prove, or even disprove, our conclusion. The same goes for studying larger chains. As the number of atoms present in a system increases, so does the inaccuracy of the model's predictions. Therefore, it may be interesting to see whether the results of the two models even out at one point.

Future Directions

In future research, it could be interesting to introduce longer chains, and odd electron counts to further test the model capacity to learn delocalization. Delocalization begins to increase and behave differently in larger systems, introducing non-local effects and depending on interactions across distant atoms. To integrate this into a neural network, the model would have to learn how to handle unpaired electrons and non-integer occupancies, adding complexity to the entire process. This would also make predictions more realistic, thanks to the better representation of molecules with specific characteristics.

One could also explore different regularization techniques such as dropout, weight decay, or batch normalization to further improve generalization. These types of methods are most effective when training on small datasets or on high dimensional inputs, both of which are used in molecular systems. Future research could involve comparing a model with regularization to one without it across the same atom counts, almost in the same manner as this study. This could lead to reduced overfitting and improved performance across test sets.

Finally, future research could investigate how model performance changes with different types of coupling distributions, such as uniform or bimodal distribution. This should be done for numerous reasons, including but not limited to mimicking different types of bond types or reproducing molecular diversity. Using different distribution methods could also test how flexible and reliable the model truly is, providing further insight into its functionality.

CONCLUSION

Ultimately, the main goal of this paper is to explore whether a general neural network potential could successfully learn and generalize in systems with delocalized electrons and compare its functionality to that of the Behler-Parrinello Model. In the end, across the atom chains, the full chain model consistently reduces both the training and the test error, indicating that it is able to learn how to approximate the ground-state energy. Additionally, the scatter plots depicting the predicted energy against the true energy show that the points were mostly clustered around the optimal

predictions line, with relatively small error numbers compared to those present in each of the results of the atom environment models. This is especially pertinent in the larger chains, where the highest ones are located. For example, the error number for a 10-atom chain in the full chain model is equal to 0.0279, while, for the same sized system each of the three windows in the atom environment models, they are equal to 0.0613, 0.0524, and 0.0417 respectively. Therefore, we conclude that the full chain model is able to learn better and make more accurate predictions than the Behler-Paranillo Model.

This study demonstrates that neural networks can effectively learn to predict the energies of delocalized systems without explicitly modeling long-range interactions. In many cases, their predictions rival or even surpass those of established models such as the Behler–Parrinello model. The consistent performance of the full chain model across varying atom chain lengths suggests that even relatively simple feed-forward architecture can capture key electronic behaviors in delocalized systems. However, the success of such models still relies heavily on access to large, high-quality datasets that allow them to extract meaningful patterns from local inputs. Overall, these results indicate that, with sufficient data and careful training, machine learning potentials hold strong promise for extending predictive modeling beyond purely local electronic systems.

While this research produces telling results, there are still some limitations presents. Firstly, we utilize a very simple 1D tight-binding model in generating ground truth energies. We use this type of model as it comes with relatively fast running time and allows for controlled testing. While this does prove to be effective and even useful in terms of testing learning capacity, this model assumes non-interacting electrons, along with a highly idealized structure. Real molecular systems are, however, much more complex and may contain characteristics that where not taken into account (i.e., orbital overlap, specific geometries). Therefore, the use of this type of model may result in a lack of real chemical complexity, leading to unrealistic electronic structure and limited transferability to a 3D system.

The second limitation is that, for simplicity, the energy target is based on an ideal- ized Hamiltonian. The value used in this study is a relatively simple matrix with only nearest-neighbor couplings being taken into account. We use this, like other factors, for computational simplicity. Unfortunately, these values disregard many physical properties and realities that can strongly influence electronic properties in real systems (i.e., environ- mental effects, next-nearest neighbor hopping). This leads to problems when interpreting a prediction as certain electronic properties like conductance or delocalization are sensitive to factors that were not added to the model.

To address the limitations, present in our work, as well as to expand upon the knowledge in the relevant field, future researchers could incorporate more realistic chemical data. As previously stated, the dataset we use was limited in that it was too simple to represent realistic values. While its accuracy is questionable, the DFT method, used in the Behler-Paranillo model, serves as one example of a more realistic data source. Future work could entail applying methods such as this to the full chain model as to better reflect actual molecular behavior and structures.

CONFLICT OF INTERESTS

The authors declare that there are no conflicts of interest related to this work.

REFERENCES

1. Blank T, Brown SS, Calhoun A, Doren DJ. Neural Network Models of Potential Energy Surfaces. *Journal of Chemical Physics.* 1995; 103: 4129-4137. https://doi.org/10.1063/1.469597
2. Behler J, Parrinello M. Generalized Neural-Network Representation of High- Dimensional Potential-Energy Surfaces. *Physical Review Letters*. 2007; 98. https://doi.org/10.1103/PhysRevLett.98.146401
3. Smith JS, Isayev O, Roitberg AE. ANI-1: an extensible neural network potential with DFT accuracy at force field computational cost. *Chemical Science.* 2017; 8: 3192- 3203. https://doi.org/10.1039/C6SC05720A
4. Artrith N, Morawietz T, Behler J. High-dimensional neural-network Potentials for Multicomponent systems: Applications to Zinc Oxide. *Physical Review B*. 2011; 83. https://doi.org/10.1103/PhysRevB.83.153101
5. Behler J. Atom-centered Symmetry Functions for Constructing high-dimensional Neu- ral Network Potentials. *The Journal of Chemical Physics*. 2011; 134: 074106. https://doi.org/10.1063/1.3553717
6. Bart'ok AP, Kondor R, Csa'nyi G. On Representing Chemical Environments. *Phys- ical Review B*. 2013; 87. https://doi.org/10.1103/PhysRevB.87.184115
7. Pirdashti M, Curteanu S, Kamangar MH, Hassim MH, Khatami MA. Artificial Neural networks: Applications in Chemical Engineering. *Reviews in Chemical Engineering.* 2013; 24 (9): 205-239. https://

doi.org/10.1515/revce-2013-0013
8. Behler J. Representing Potential Energy Surfaces by high-dimensional Neural Network Potentials. *Journal of Physics: Condensed Matter.* 2014; 26: 183001. https://doi.org/10.1088/0953-8984/26/18/183001
9. Behler J. Constructing high-dimensional Neural Network potentials: a Tutorial Re- view. *International Journal of Quantum Chemistry.* 2015; 115: 1032-1050. https://doi.org/10.1002/qua.24890
10. Gossett E, Toher C, Oses C, Isayev O, *et al*. AFLOW-ML: a RESTful API for machine- learning Predictions of Materials Properties. *Computational Materials Science.* 2018; 152: 134-145. https://doi.org/10.1016/j.commatsci.2018.03.075
11. Muratov EN, *et al*. A Critical Overview of Computational Approaches Employed for COVID-19 Drug Discovery. *Chemical Society Reviews.* 2021; 50. https://doi.org/10.1039/D0CS01065K
12. Li B, Raji IO, Gordon AGR, Sun L, *et al*. Accelerating Ionizable Lipid Discovery for mRNA Delivery Using Machine Learning and Combinatorial Chemistry. *Nature Materials*. 2024; 23: 1002-1008. https://doi.org/10.1038/s41563-024-01867-3
13. Paszke A, *et al*. PyTorch: An Imperative Style, High-Performance Deep Learning Library. 2019; https://arxiv.org/abs/1912.01703.

APPENDIX: Code for the Full Chain Model

```
import numpy as np import torch
import torch.nn as nn import torch.optim as optim
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split


#Set random seeds for reproducibility np.random.seed(0)
torch.manual_seed(0)

#Utility  Functions
def make_hamiltonian(couplings, num_atoms): H = np.zeros((num_atoms,
    num_atoms)) for i in range(len(couplings)):
        H[i, i + 1] = H[i + 1, i] = couplings[i] return H

def compute_energy(H, num_electrons): eigs =
    np.sort(np.linalg.eigvalsh(H)) energy = 0.0
    full_pairs = num_electrons // 2
    energy += 2 * np.sum(eigs[:full_pairs]) if num_electrons % 2 == 1:
        energy += eigs[full_pairs]      # singly occupy next orbital
    return energy


#Main Loop
atom_counts = list(range(2, 9))         # even and odd
summary_data = []

for num_atoms in atom_counts:
    num_couplings = num_atoms - 1
    num_electrons  =  num_atoms    # 1 electron per atom


        #Generate dataset N
        = 10000
        X  =  np.random.normal(loc=-1.0,  scale=0.05,  size=(N,  num_couplings))
        y = np.array([compute_energy(make_hamiltonian(c, num_atoms), num_electrons) for c in


        #Normalize energy values
        y_mean = y.mean()
        y_std = y.std()
        y = (y - y_mean) / y_std


        #Split into train/test sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat X_train,
        X_test = map(torch.tensor, (X_train, X_test))
        y_train, y_test = map(lambda t: torch.tensor(t).view(-1, 1), (y_train, y_test)) X_train =
        X_train.float()
```

```
X_test = X_test.float() y_train
= y_train.float()
y_test = y_test.float()


#Define neural network model class
EnergyNN(nn.Module):
    def _init_(self, input_size):
        super()._init_()
        self.net =
            nn.Sequential( nn.Linear(input
            _size, 64), nn.ReLU(),
            nn.Linear(64, 64),
            nn.ReLU(),
            nn.Linear(64, 1)
        )
    def forward(self, x):
        return self.net(x)

model = EnergyNN(num_couplings) loss_fn
= nn.MSELoss()
opt = optim.Adam(model.parameters(), lr=0.001)


#Train model
train_losses = []
test_losses = []
epochs = 500

for epoch in range(epochs):
    model.train()
    pred = model(X_train)
    loss = loss_fn(pred, y_train)


    opt.zero_grad()
    loss.backward()
    opt.step()

    model.eval()
    with  torch.no_grad():  test_pred =
        model(X_test)
        test_loss  =  loss_fn(test_pred,  y_test).item()


    train_losses.append(loss.item())
    test_losses.append(test_loss)

#Save predictions for plotting with
torch.no_grad():
```

```
            y_pred = model(X_test).squeeze().numpy() y_true
            = y_test.squeeze().numpy()

        #Denormalize  for  plotting
        y_pred = y_pred * y_std + y_mean
        y_true  =  y_true  *  y_std  +  y_mean

        summary_data.append({ 'atoms':
            num_atoms,
            'train_losses':
            train_losses, 'test_losses':
            test_losses, 'y_true': y_true,
            'y_pred':  y_pred
        })


#Plotting Section
fig, axs = plt.subplots(2, len(atom_counts), figsize=(7 * len(atom_counts), 12))


#1. Plot Training vs Test Loss
for i, data in enumerate(summary_data):
    axs[0, i].plot(data['train_losses'], label='Train') axs[0,
    i].plot(data['test_losses'], label='Test') axs[0,
    i].set_title(f'{data["atoms"]} atoms') axs[0,
    i].set_xlabel('Epoch')
    axs[0, i].set_ylabel('MSE Loss')
    axs[0, i].legend()
    axs[0, i].grid(True)


#2. Plot Predicted vs Actual Energy
for i, data in enumerate(summary_data):
    axs[1, i].scatter(data['y_true'], data['y_pred'], alpha=0.3) min_val =
    min(data['y_true'].min(), data['y_pred'].min())
    max_val  =  max(data['y_true'].max(),  data['y_pred'].max())
    axs[1, i].plot([min_val, max_val], [min_val, max_val], 'r--') axs[1,
    i].set_xlabel('True Energy')
    axs[1, i].set_ylabel('Predicted Energy')
    axs[1, i].set_title(f'{data["atoms"]} atoms') axs[1,
    i].grid(True)

plt.tight_layout()
plt.show()
```

A. Code for the Atom Environment Model

```
import numpy as np import
torch
import torch.nn as nn import
torch.optim as optim
import torch.nn.functional as F import
matplotlib.pyplot as plt

from  sklearn.model\_selection  import  train\_test\_split


#Utility  Functions
def make_hamiltonian(couplings, num_atoms): H =
    np.zeros((num\_atoms, num\_atoms)) for i in
    range(len(couplings)):
        H[i, i + 1] = H[i + 1, i] = couplings[i]
    return H

def compute_energy(H, num_electrons): eigs =
    np.sort(np.linalg.eigvalsh(H)) energy = 0.0
    full_pairs = num_electrons // 2
    energy += 2 * np.sum(eigs[:full_pairs]) if
    num_electrons % 2 == 1:
        energy += eigs[full_pairs] #singly occupy next orbital return energy

#Main Loop
atom_counts = list(range(2, 11)) #even and odd
summary_data = []

for num_atoms in atom_counts:
    num_couplings = num_atoms - 1
    num_electrons = num_atoms #1 electron per atom


    #Generate dataset N
    = 10000
    X  =  np.random.normal(loc=-1.0,  scale=0.05,  size=(N,  num_couplings))
    y = np.array([compute_energy(make_hamiltonian(c, num_atoms), num_electrons) for c in


    #Normalize energy values
    y_mean = y.mean()
    y_std = y.std()
    y = (y - y_mean) / y_std


    # Split into train/test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat X_train,
    X_test = map(torch.tensor, (X_train, X_test))
```

```
    y_train, y_test = map(lambda t: torch.tensor(t).view(-1, 1), (y_train, y_test)) X_train =
    X_train.float()
    X_test = X_test.float() y_train
    = y_train.float() y_test =
    y_test.float()

    #Define Behler-Parrinello-style model
    class AtomNN(nn.Module):
        def _init_(self, window_size):
            super()._init_() input_size =
            2 * window_size self.net =
            nn.Sequential(
                nn.Linear(input_size, 64),
                nn.ReLU(),
                nn.Linear(64, 64),
                nn.ReLU(),
                nn.Linear(64, 1)
            )


        def forward(self, x):
            return self.net(x)

    class ChainEnergyModel(nn.Module): def _
        init_(self, window_size):
            super()._init_() self.window_size =
            window_size self.atom_nn =
            AtomNN(window_size)

        def forward(self, beta):
            batch_size, num_couplings = beta.shape
            pad = self.window_sizenatoms = num_couplings + 1


            beta_padded = F.pad(beta, (pad, pad), mode='constant', value=0.0) #(batch_si
            beta_windows = beta_padded.unfold(dimension=1, size=2 * pad, step=1) #(batch
            beta_windows_flat = beta_windows.reshape(-1, 2 * pad) # (batch_size * natoms


            atom_energies = self.atom_nn(beta_windows_flat) #(batch_size * natoms, 1) atom_energies
            = atom_energies.view(batch_size, natoms)
            total_energy = atom_energies.sum(dim=1) #(batch_size,) return
            total_energy.view(-1, 1)

#Initialize model
window_size = 1 #How far to look on each side; changed to 2 and 3 model =
ChainEnergyModel(window_size)
loss_fn  =  nn.MSELoss()
opt = optim.Adam(model.parameters(), lr=0.001)
```

```
#Train model
train_losses = []
test_losses = [] epochs
= 500

for epoch in range(epochs):
    model.train()
    pred = model(X_train)
    loss = loss_fn(pred, y_train)


    opt.zero_grad()
    loss.backward()
    opt.step()
    model.eval()
    with  torch.no_grad():  test_pred =
        model(X_test)
        test_loss  =  loss_fn(test_pred,  y_test).item()


    train_losses.append(loss.item())
    test_losses.append(test_loss)

#Save predictions for plotting with
torch.no_grad():
    y_pred = model(X_test).squeeze().numpy() y_true =
    y_test.squeeze().numpy()

#Denormalize for plotting
y_pred = y_pred * y_std + y_mean y_true  =
y_true  *  y_std  +  y_mean

summary_data.append({ 'atoms':
    num_atoms,
    'train_losses':
    train_losses, 'test_losses':
    test_losses, 'y_true': y_true,
    'y_pred':  y_pred
})


#Plotting Section
fig, axs = plt.subplots(3, 3, figsize=(20, 20))
axs = axs.flatten()
for i, data in enumerate(summary_data): axs[i].scatter(data['y_true'],
    data['y_pred'], alpha=0.3, s=20) min_val = min(data['y_true'].min(),
    data['y_pred'].min())
    max_val  =  max(data['y_true'].max(),  data['y_pred'].max())
    axs[i].plot([min_val, max_val], [min_val, max_val], 'r--')
```

```
    axs[i].set_xlabel('True Energy') axs[i].set_ylabel('Predicted
    Energy')
    y_true = data['y_true']
    y_pred = data['y_pred']
    rms_error = np.sqrt(np.mean((y_true - y_pred) ** 2))
    axs[i].set_title(f'{data["atoms"]} atoms, error = {rms_error:.4f} ') axs[i].grid(True)

plt.tight_layout()
plt.show()
```