Project Machine Learning
SchNet: A Continuous Filter Convolutional Neural Network for Modeling Quantum Interactions
— Milestone 3: The Final Prediction Method —

Ariane Oesch, Christian Kasim Loan, Trung Duc Ha

February 2, 2024

Abstract

In this concluding milestone, we assess the final prediction method. Our approach involves the development of an alternative confidence estimation method based on normal distributions, with the integration of explainability techniques. To enhance the performance of the model, we introduce a regularization technique. Moreover, we introduce a new architecture containing transformers. Additionally, we explore a hyperparameter grid for SchNet to gain more insights into the hyperparameter space and empirically find the best ones. Our evaluation indicates that SchNet gains major benefits from introducing normalization methods, achieving better results than the original paper, and might benefit from increased complexity.

1 Introduction

We build on the work from the second milestone to implement the final version of our SchNet model. After implementing and evaluating our implementation of the SchNet model against a baseline in the previous two milestones, we try to match the metrics of the original paper by Schütt *et al.* [1].

For this final stage, we will introduce measures for confidence and explainability, explore a hyperparameter grid for SchNet, introduce regularization methods, and a SchNet Transformer architecture. We evaluate the performance of the final models and the interpretability methods in suitable experiments.

2 Methodology

We detail the final implementations of our SchNet models. First, we introduce our chosen confidence measure, which is based on an estimated normal distribution. Second, we discuss our method of hyperparmeter tuning. Following this, we derive a measure for explainability. Finally, we introduce enhancements to the architecture, namely a regularization technique and the transformer architecture. We carried out all experiments and training using the ISO17 data set.

2.1 Confidence Measure

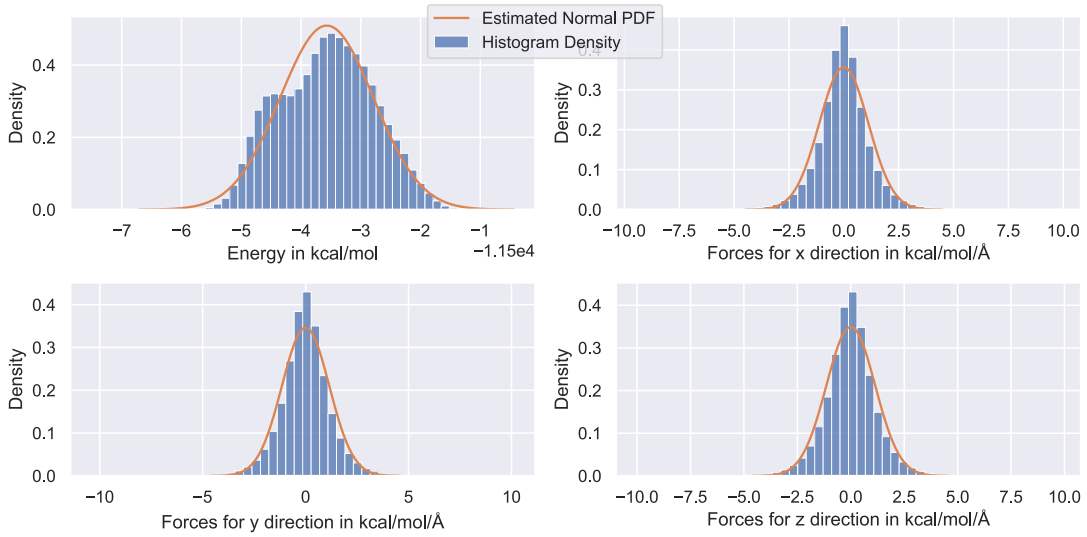
In the previous milestone report, we employed ensemble neural networks to estimate a confidence measure for model predictions[2]. However, this requires training multiple models with an algorithm that is similar to k-fold cross-validation. This is computationally impractical for models that should be ready for production. Thus, we propose an alternative method using estimated normal distributions.

2.1.1 Normal Distribution of Predicted Features

To establish a confidence measure during model training, we use the fact that the predictions of the model appear to be normally distributed [3]. An advantage of assuming the normal distribution is the simplicity of calculating it, requiring only the estimation of the mean μ and standard deviation σ to construct its probability distribution function (PDF).

First, we further analyze the distribution of predicted features for the ISO17 training data. We construct a density using histograms of one hundred bins and use it to estimate the parameters of a normal distribution and subsequently its PDF. Figure 1 shows the histograms of the data and the estimated normal distributions for the energy and separate force directions.

Figure 1: Histograms for Energy and Forces of the ISO17 dataset. The densities of the histogram bins are colored blue and the estimated normal probability distribution function in orange.



We observe that the estimated PDF is close to the original data, with two important differences. First, the distribution for the energies seems to have a subtle kink, indicating that it could be a composition of two normal distributions. Additionally, the peak around the mean for the forces seems to be a higher. This indicates a higher kurtosis than a normal distribution, potentially suggesting a different underlying distribution.

However, the normal distribution seems to fit well, and we justify it using the Kullback–Leibler (KL) Divergence D_{KL}. The KL divergence measures the difference between two distributions, where a lower value represents a better fit. Using this, we compare the histogram distribution to our estimated PDF, presented in Table 1.

Table 1: Estimated mean μ, standard deviation σ of the normal distribution and KL divergence of the estimations. The values are rounded to three decimal places.

Estimated Feature	μ	σ	D_{KL}
Energy	-11,503.569	0.613	0.024
Forces – x	0	1.253	0.034
Forces – y	0	1.328	0.026
Forces – z	0	1.306	0.025

Notably, the KL divergence is very low for the energy and forces. This indicates, that the estimated normal distributions are a good fit for the data and represent it well. Interestingly, we also observe that the parameters of the forces closely resemble a standard normal distribution with mean zero and standard deviation of one.

Estimating normal distribution parameters typically requires the entire dataset. Assuming that we feed new data to the model using stochastic optimization methods, accessing the complete

data set becomes impractical. We propose an online solution that accommodates continuously provided data in Section 2.1.2.

2.1.2 Running Mean and Standard Deviation

A challenge with estimating the mean and variance for a normal distribution is that we need access to the entire data set prior to training the model. This might not be possible or impractical for very large data sets. A more practical approach involves estimating the mean and standard deviation, while data is fed to the model. Welford [4] proposes an algorithm to compute the mean and variance (and, in turn, the standard deviation) incrementally. We add the running parameters as a property in our implementation of SchNet.

For seamless integration with our PyTorch model, we use a publicly available implementation [5]. The model only computes the parameters for the first epoch of every training run. The computed mean and standard deviation can be restored after saving the model. Therefore, we have an efficient way to estimate the parameters of normal distributions.

2.1.3 Confidence Measure using Z-scores

Having estimated a normal distribution for our features, we use the standard score or Z-score to estimate the confidence of the predictions. It measures, how far a point deviates from the mean, relative to the standard deviation and is defined as

$$z = \frac{\hat{x} - \mu}{\sigma}$$

, where $\hat{x}$ is the predicted feature, μ is the *population* mean and σ the *population* standard deviation. We will use the estimated sample mean and standard deviation for these parameters, as we have a sufficient amount of data to approximate the population parameters.

We equip our implementation of the SchNet model with a `predict` function. It produces predictions for the energy and derived forces with their respective confidence scores. We use the mean and standard deviation provided by the method described in Section 2.1.2.

2.2 Hyperparameter Selection

Fine-tuning neural network parameters is a challenging task that requires significant computational resources. To identify optimal parameters for the SchNet architecture, we utilized the Ray library [6] and its ASHAScheduler [7] for efficient exploration of a vast parameter space. Employing 2 GPUs in parallel expedited experimentation with a grid of 324 parameter configurations. The integration of Ray and ASHAScheduler provides a robust platform for hyperparameter tuning, ensuring a streamlined optimization process. Notably, Ray, with its distributed computing tools, enables accelerated experimentation and scalable model training.

Our approach aligns with the principles outlined by the ASHAScheduler. The ASHAScheduler employs an asynchronous hyperparameter optimization strategy, dynamically allocating resources to promising configurations. This adaptability efficiently navigates the intricate landscape of hyperparameter combinations.

We explore a grid for a total of 5 architecture parameters, consisting of the atom embedding dimension, the number of interaction layers (N Interactions), the maximum value for the radial basis functions (RBF Max), the number of RBF filters (N RBF) and the choice of nonlinear activation functions. For each, we explore less and more complex combinations to study the impact of complexity on model performance. For the activation functions, we experiment with the default of SchNet, the Shifted Softplus (SSP) [1] and additionally LeakyRELU and GELU, which have seen popular use [8].

Table 3 presents the values explored in detail, and Figure 7 shows the combinations of the best performing parameters. Section 3.4 delves into improvements in hyperparameter choices.

2.3 Explainability

For interpreting model predictions and achieving explainability, we employ sensitivity analysis [9]. This approach quantifies the local gradient at a given data point x, measuring the *sensitivity* to each feature of x. Although sensitivity analysis may not be fully suited to explain features in the context of class prediction tasks, it is useful for regression problems, as evidenced by its widespread application [9]. In our regression scenario, the gradients of the predicted energy with respect to atom positions represent the forces acting on each atom. These forces contribute significantly to the overall potential molecular energy, making them integral to the interpretative process.

In our case, a data point x corresponds to a molecule, consisting of the atomic numbers $Z = (Z_1, ..., Z_n)$ for each atom and the atom positions $R = (\mathbf{r}_1, ..., \mathbf{r}_n)$, where n is the number of atoms of the molecule. We compute the sensitivity for these two input features.

We adapt the definition by Montavon *et al.* [9]. Let $i \in \{1, 2\}$ and $\mathbf{x}_1 = Z, \mathbf{x}_2 = R$. We define the sensitivity S_i for feature i as $S_i(x) = \left(\frac{\partial \hat{E}}{\partial \mathbf{x}_i}\right)^2$, where i is the current feature and $\hat{E}$ is the output of the network.

A limitation in PyTorch and tensor calculus in general is that gradients cannot be calculated for integer-type tensors such as Z. The gradient of these tensors is not well-defined.

We recall, that SchNet first creates atom type embeddings $(\mathbf{a}_{Z_1}, ..., \mathbf{a}_{Z_n})$ for each atom to represent it inside the network [1]. Instead of calculating the gradient w.r.t. Z, we calculate it w.r.t. $\mathbf{a}_{Z_i}$. However, this will change the output dimension of the sensitivity and will not correspond to Z. Therefore, we norm the gradient w.r.t. each embedded atom. The squared magnitude will then correspond to each atom in Z.

We arrive at the final definition for the sensitivity of each feature:

$$S_1(x) = \left(\left\| \frac{\partial \hat{E}}{\partial \mathbf{a}_{Z_1}} \right\|^2 \quad \cdots \quad \left\| \frac{\partial \hat{E}}{\partial \mathbf{a}_{Z_n}} \right\|^2 \right) \quad \text{and} \quad S_2(x) = \left(\left(\frac{\partial \hat{E}}{\partial \mathbf{r}_1} \right)^2 \quad \cdots \quad \left(\frac{\partial \hat{E}}{\partial \mathbf{r}_n} \right)^2 \right) = \left(\hat{\mathbf{F}}_1^2 \quad \cdots \quad \hat{\mathbf{F}}_n^2 \right)$$

We implement this confidence measure with an `explain` function, which returns the sensitivity of the atomic numbers and the positions of the atoms.

2.4 Regularization

In the second milestone report [2], we described that the proposed architecture for SchNet does not incorporate regularization techniques. We introduce Root Mean Square Layer Normalization (RMSNorm) to the model architecture in an attempt to improve the training behavior and the validation loss.

Zhang *et al.* [10] proposed RMSNorm as an improvement on LayerNorm [11], which stabilizes model training and helps with model convergence by adjusting the statistics of neuron activations. While LayerNorm has found tremendous success in many tasks, it suffers from computational overhead [10]. In contrast, RMSNorm is easier to compute while retaining the advantages of LayerNorm. Notably, large language models have recently employed RMSNorm to stabilize the training of networks with billions of parameters [12].

We apply RMSNorm in SchNet after each linear layer, preceding nonlinear activation, as suggested by the original paper [10].

2.5 SchNet Transformer

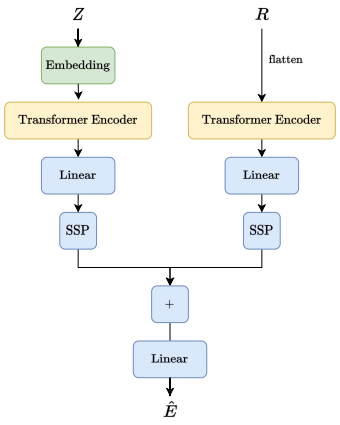
In the recent past, transformers [13] have achieved significant success, especially in the field of natural language processing, audio processing, and computer vision [14]. In the latter, vision transformer architectures (ViT) [15] have begun outperforming conventional counterparts, such as the convolutional neural network.

From this idea we apply the transformer architecture to our problem of predicting energies and forces of molecules. We model the input features similarly to the method used by ViT. It transforms each input into flattened image patches, which are then fed into a transformer

architecture [15]. We apply it similarly, by flattening the positions of the atom positions R. The atomic numbers Z are already one-dimensional and thus do not require further processing. However, we will generate embeddings for each type of atom.

Our approach involves two separate transformers, one R and one for Z, using only the encoder of the transformer architecture [13]. We assume that the encoders will learn the interactions between the atom types and the atom positions. These representations are then suitable for calculating the overall energy level. Figure 2 provides a high-level view of the architecture.

Figure 2: High level SchNet Transformer architecture



To achieve a fair comparison, we match the number of parameters of the SchNet Transformer with the default SchNet configuration. More specifically, we adjust the hyperparameters of the encoders to an internal embedding of 64, with the transformer for Z having 8 attention heads with 3 layers, while for R has 3 attention heads (one for each coordinate) with 2 layers. With this configuration, the transformer version has about 140,000 parameters, which is a 20% increase over the default model.

2.6 Final Model Training

We train a final version of the models as follows. First, we split ISO17, which consists of roughly 500,000 data points, into a train and validation split. The validation set consists of 100,000 points, while the rest is used as the train set. This train set is further divided into a train and test set, with a split of 90%-10%, respectively. We use the test set for early stopping.

We train the models with the training parameters proposed from the paper [1] and as described in our previous milestone report [2]. However, we adjust the training epochs, extending the epochs to 150. The training procedure took 5.5 to 8 hours on an NVIDIA RTX 3090 GPU, depending on the model architecture.

3 Evaluation

We now assess the performance of our regularized and transformer models described in Sections 2.4 and Section 2.5, comparing them to the results presented by Schütt *et al.* [1]. We evaluate them in terms of runtime and mean absolute errors of the combined energy and force task. We provide an in-depth exploration of the benefits of regularization, followed by an experiment to examine the confidence measure. Lastly, we will delve into the hyperparameters tuning and the explainability of the model.

3.1 Comparison of Final SchNet models

First, we compare the runtime of the training. On the specified hardware, the entire training procedure took approximately 5.5 hours for the standard SchNet model, 7 hours for the RMSNorm and 8 hours for the transformer version. However, this does not describe how *efficiently* the model was trained. We analyze the training behavior of the regularized and the default model in Section 3.2.

We compare the average mean absolute error (MAE) of the combined energy and force predictions in the validation set for the model architectures. We present the results for the final models in Table 2.

We note that the models we trained performed slightly better than the metrics presented in the original paper. This is expected for the regularized model. However, higher performance compared to the original paper raises uncertainty, and we hypothesize that differences in training time and other hyperparameters might contribute to this variance. The original authors do not specify their exact procedure and have not uploaded pre-trained models at this time, making direct comparisons and reproduction difficult.

The transformer architecture seems to perform significantly worse. Similarly to our analysis at the second milestone [2], we fit a linear regression to the train and test loss, revealing a downward trend even in the later stages of training. Exact results are provided in Appendix A. This suggests potential undertraining due to the heavy layer regularization by this architecture [13], making model convergence slower compared to the other methods. We suggest future work to model the interactions between atom types and positions in a more sophisticated manner, which could boost model performance.

Table 2: Validation Loss for the examined SchNet architectures, in mean absolute error of the combined energy force tasks, rounded to two decimal places. Better performance (lower loss) is highlighted in bold.

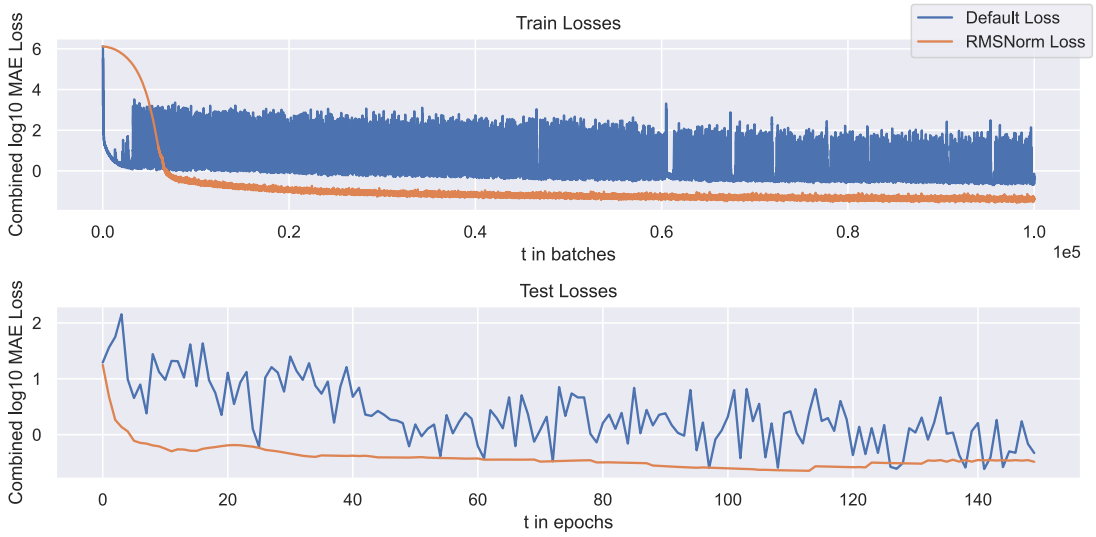
Model	**Val. Loss**
Schütt *et al.* [1]	0.36
Default SchNet	0.25
Ours – RMSNorm	**0.10**
Ours – Transformer	1.16

3.2 Analysis of Regularized Models

In Section 3.1 we compare the models and found, that the RMSNorm regularized model performs significantly better than the non-regularized counterpart. However, due to additional computation, model training took approx. 27% longer for the regularized model.

We justify this trade-off by presenting the training behavior of the models. Figure 3 shows the train losses and validation losses, with the losses on a $\log_{10}$ scale to see the behavior more clearly of the overall training.

Figure 3: $\log_{10}$ of train and test losses for the default and regularized SchNet model.



Despite the initial slower decrease in loss, the training for the regularized model is notably more stable, achieving an overall lower loss. Additionally, the variance of the losses is significantly lower, indicated by the compactness of the plot.

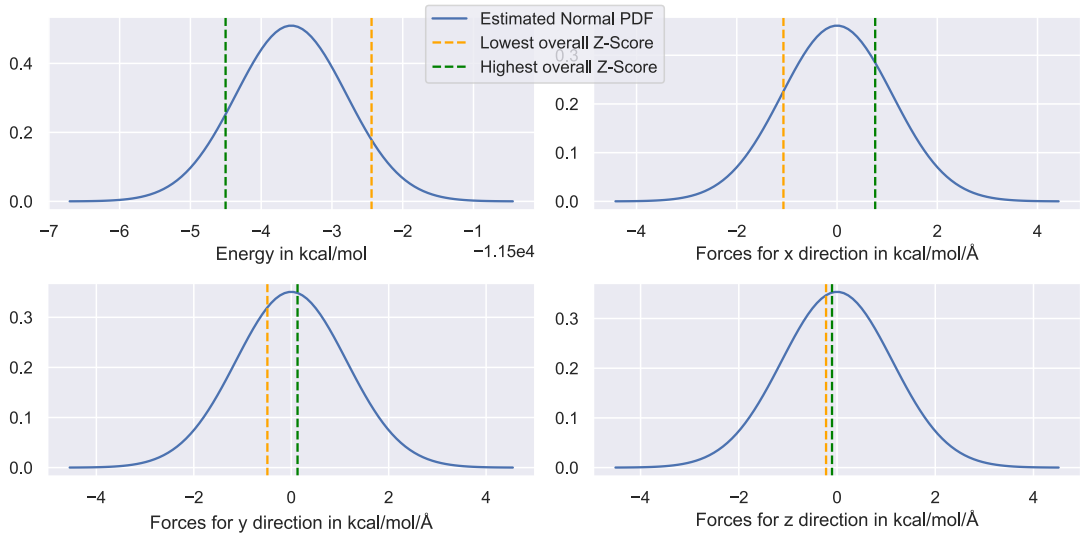
Furthermore, we achieve significantly more stable and lower test losses. In contrast, the train loss for the default model falls sharply early on. However, this does not translate to the test loss. The default model encounters challenges in generalization, leading to oscillations in test loss due to overfitting.

The regularized model does not suffer from this problem. It demonstrates robust generalization, even with variations in the input space. This shows the adaptability to new data, allowing for further training and fine-tuning. However, there might a slight hint at potential overfit for this model, as the validation loss starts to rise at the end of training. Exploring other regularization techniques such as weight decay and dropout could further enhance generalization capabilities.

3.3 Analysis of Confidence Measure

To analyze the confidence measure, we conduct the following experiment. Using the validation set for ISO17, we compute the total confidence of a prediction by summing the confidence of each feature. We collect the lowest and highest overall confidence and compare the two data points. We present the results in Figure 4.

Figure 4: The estimated normal PDF in blue, with the most confident (lowest z-score) prediction in orange and least confident (highest z-score) prediction in green.



Rounding to two decimal places, the highest recorded confidence (lowest z-score) is 6.88, while the lowest recorded confidence (highest z-score) is 80.42. Despite this considerable gap, both predictions show low losses of 0.19 and 0.15. In particular, the validation loss for both predictions is lower than the mean (0.25). Moreover, the higher confident prediction has a worse loss than the less confident one. This suggests that the confidence measure might not be suitable to measure the quality of the prediction. The measure provides more of a probabilistic view. Predictions that are closer to the mean will have a higher score, as these values are more likely.

However, we observe a similarity to the analysis from the ensemble neural networks of the previous milestone [2]. Even in the worst case, the model seems to be much more confident in the force predictions than in the energy predictions.

3.4 Hyperparameter Tuning Results

Our experiments validate the Milestone 2 hypothesis, confirming that as model complexity increases, the loss diminishes. The best model achieved a validation loss of 0.25, surpassing the benchmark set by [1]. This underscores the efficacy of our hyperparameter tuning strategy. Figure 7 displays parameter combinations with a loss below 1 within 100 epochs. Empirical analysis recommends adjustments for atom embeddings dimension, number of interaction blocks, and the number of filters for the RBF.

Regarding the activation functions, LeakyReLU appears unfavorable, as observed in the worst models. GELU slightly underperforms SSP, the latter being the optimal choice with lower loss, consistent with [1].

Increasing the embedding dimension suggests improved performance, capturing more nuances in the data. Larger values for number of interaction layers yield better results, with 8 identified as the optimal value. Surprisingly, the smallest value is chosen for the number of RBF filters, indicating superior performance with a smaller resolution size. The optimal RBF Max value is 30, consistent with [1].

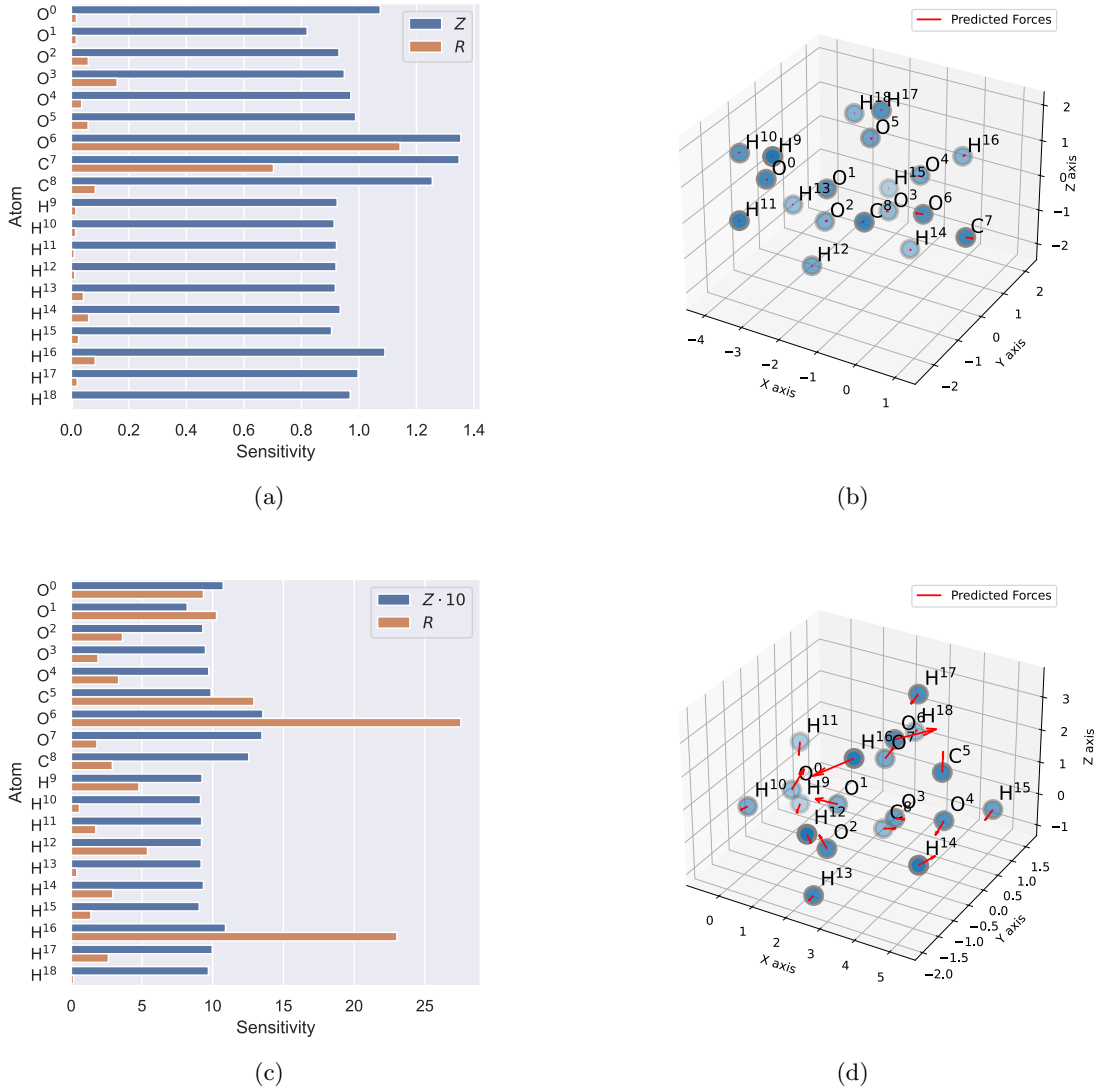
In exploration, only N Interactions and RBF Max use their highest values, creating the most com-

plex models. Conversely, Atom Embedding employs the second-highest value, implying potential longer convergence due to increased complexity. Overall, three of the five hyperparameters were updated, suggesting that the original SchNet parameters were suboptimal. This coincides with our preliminary experiment [2], indicating an avenue for future work with more complex models. Further exploration may reveal insights for better models, as proposed by Schütt *et al.* [1].

3.5 Explainability

To observe examples for explainability, we set up the following experiment. We feed the model the validation set and extract the lowest and highest energy predictions for sensitivity analysis, as detailed in Section 2.3. For the atomic numbers Z, we have the sensitivity for each atom. However, the atom positions R are three-dimensional. To enable a comparison, we take the norm of it, providing the magnitude for each atom.

We show the resulting sensitivities for each atom of the respective molecules in Figure 5 for the lowest energy and highest energy state molecule. We denote the index of the individual atoms by a superscript.



(a) (b)

(c) (d)

Figure 5: Sensitivities of Z and R for each atom and plot of lowest energy state molecule (a, b) and highest energy state molecule (c, d) in the validation set. The predicted forces are scaled to 20%. We scale the sensitivity to Z in (c) by a factor of 10, to make it more visible.

First, we note that the true forces in the lowest energy state molecule are zero, indicating that

it is in equilibrium. The magnitudes of the derived forces of the prediction, and in extension the sensitivity, are almost zero. Sensitivity for the atom types is almost equal.

However, there is a deviation for the atoms O^6 and C^7. The sensitivity to Z and R are relatively high. Examining these atoms in three-dimensional space, we see that they are next to each other, with forces pointing in opposite directions. This indicates, that these two atoms may be the most influential for future changes in the molecules' energy level. However, we note that the prediction quality of the molecule in equilibrium is less accurate (loss 0.48 compared to the mean 0.25).

For the high energy state molecule, the magnitudes of the sensitivity to R are significantly higher across all atoms. This is expected, as a higher energy state implies more forces acting on the atoms. We scale the sensitivity to Z by a factor of 10, for comparison with R. The magnitudes for Z remain similar to the lower energy case, but are now low in comparison to R.

We can again see, that two of the molecules. namely O^6 and H^{16} are the highest contributors to the sensitivity of this energy state, repelling each other based on the force directions. This observation suggests that the most prominent interactions involve oxygen coupled with other atoms.

4 Discussion

The work carried out in these three reports has demonstrated SchNet's ability to model complex molecular properties. In this section, we will delve into the implications of SchNet for quantum chemistry, particularly in relation to existing MD simulation solutions. Comparing their efficiency with that of SchNet can be challenging because they traditionally operate in a different realm.

Classical molecular dynamics (MD) software uses force fields to simulate the motion of atoms and molecules. These are empirical models that are calibrated against experimental or quantum mechanical data. Parameters are adjusted to reproduce the forces and energies and other properties. Other simulation methods also use quantum mechanical calculations, such as Density Functional Theory (DFT), sometimes combined with classical molecular dynamics.

During a simulation, initial atomic positions, velocities, and a pre-optimized force field are known. At each step, forces and energies are calculated based on the current configuration and force field parameters. Numerical integration of equations of motion yields the next atom's states. Positions and velocities are updated iteratively, allowing the simulation to progress over time.

Instead of attempting a direct comparison, we can shift our attention to the potential contributions SchNet could make to the field of MD simulations. Indeed, it is possible to incorporate ML solutions such as SchNet into MD simulations. Traditional force fields can be replaced by machine learning potential energy surfaces, generated using techniques such as neural networks, from which we can derive a force field.

Schütt *et al.* [16] use SchNet in this manner, using the model beforehand to learn an energy-conserving force field that will be used in a PIMD (Path-Integral Molecular Dynamics) simulation. The use of SchNet accelerates the simulation by three orders of magnitude, reducing runtime from several years to a few hours. It thus allows molecular dynamics studies that won't be computationally feasible with classical MD simulations based on common DFT approaches.

This demonstrates the need for fast and accurate machine learning models, such as SchNet, to explore chemical interactions and quantum behavior in greater depth, and ultimately to better understand the characteristics of molecules and materials.

5 Conclusion

In conclusion, we explored methods to interpret model predictions and suggestions for improvement of the original architecture. We introduced RMSNorm as a regularization mechanism, the transformer architecture and explored the hyperparameter space using an efficient scheduler.

Our reproduction of the model surpasses the results of the original paper, although pinpointing the cause for this improvement remains challenging due to the lack of reproducibility details. The

inclusion of RMSNorm in the architecture yielded significantly better results, and our hyperparameter tuning experiments suggest better performance with a more complex model. While our custom SchNet Transformer architecture shows promising results, it performs worse than the original SchNet.

Our measure for confidence and explainability are efficient to compute and aid in the understanding of the model predictions. However, this is less the case for the Z-score, as it demonstrates mixed quality results.

For future work, we suggest exploring additional options for regularization, explainability, confidence, parameter-tuning to further enhance model performance. Additionally, combining SchNet with more contemporary methods such as Cartesian Tensor Representations of TensorNet [17] and application to non-molecular graph datasets could be a promising research direction.

References

[1] K. T. Schütt, P.-J. Kindermans, H. E. Sauceda, *et al.*, "SchNet: A continuous-filter convolutional neural network for modeling quantum interactions," no. arXiv:1706.08566, Dec. 2017. arXiv: 1706.08566 [physics, stat].

[2] C. K. Loan, T. D. Ha, and A. Oesch, *Project Machine Learning Milestone 2: Model Selection and Evaluation*, Jan. 2024.

[3] C. K. Loan, T. D. Ha, and A. Oesch, *Project Machine Learning Milestone 1: Data Sets and Prototype*, Nov. 2023.

[4] B. P. Welford, "Note on a Method for Calculating Corrected Sums of Squares and Products," *Technometrics*, vol. 4, no. 3, pp. 419–420, Aug. 1962, ISSN: 0040-1706. DOI: 10.1080/00401706.1962.10490022.

[5] N. Pochinkov, *Nickypro/welford-torch*, https://github.com/nickypro/welford-torch, Dec. 2022.

[6] P. Moritz, R. Nishihara, S. Wang, *et al.*, "Ray: A distributed framework for emerging AI applications," *CoRR*, vol. abs/1712.05889, 2017. arXiv: 1712.05889.

[7] L. Li, K. Jamieson, A. Rostamizadeh, *et al.*, *A system for massively parallel hyperparameter tuning*, 2020. arXiv: 1810.05934 [cs.LG].

[8] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, vol. 503, pp. 92–108, Sep. 2022, ISSN: 09252312. DOI: 10.1016/j.neucom.2022.06.111.

[9] G. Montavon, W. Samek, and K.-R. Müller, "Methods for interpreting and understanding deep neural networks," *Digital Signal Processing*, vol. 73, pp. 1–15, Feb. 2018, ISSN: 10512004. DOI: 10.1016/j.dsp.2017.10.011.

[10] B. Zhang and R. Sennrich, "Root Mean Square Layer Normalization," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[11] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer Normalization*, Jul. 2016. DOI: 10.48550/arXiv.1607.06450. arXiv: 1607.06450 [cs, stat].

[12] H. Touvron, L. Martin, K. Stone, *et al.*, *Llama 2: Open Foundation and Fine-Tuned Chat Models*, Jul. 2023. arXiv: 2307.09288 [cs].

[13] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[14] T. Lin, Y. Wang, X. Liu, *et al.*, "A survey of transformers," *AI Open*, vol. 3, pp. 111–132, Jan. 2022, ISSN: 2666-6510. DOI: 10.1016/j.aiopen.2022.10.001.

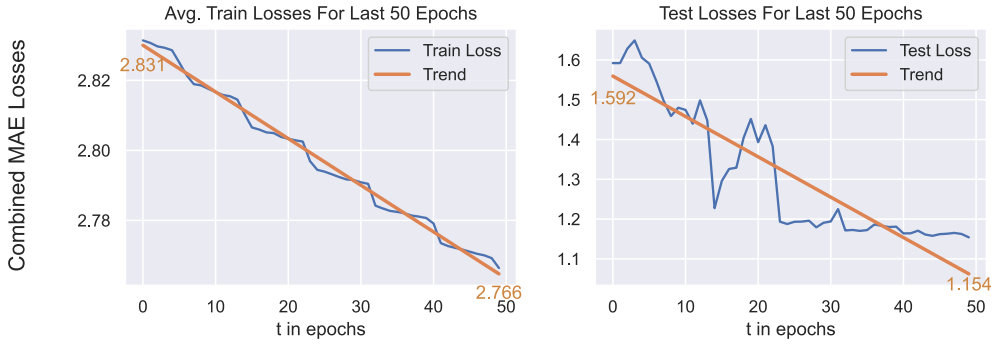
[15] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, Jun. 2021. DOI: 10.48550/arXiv.2010.11929. arXiv: 2010.11929 [cs].

[16] K. T. Schütt, H. E. Sauceda, P.-J. Kindermans, *et al.*, "Schnet – a deep learning architecture for molecules and materials," *The Journal of Chemical Physics*, vol. 148, p. 241 722, 2018. DOI: 10.1063/1.5019779.

[17] G. Simeon and G. de Fabritiis, *TensorNet: Cartesian Tensor Representations for Efficient Learning of Molecular Potentials*, Oct. 2023. arXiv: 2306.06482 [physics].

A Appendix: SchNet Transformer Training

Figure 6: Train losses and test losses for the SchNet Transformer model. The train losses have been averaged over the epochs. We show the losses for the last 50 epochs of the training. We fit a linear regression on this data and see a clear downward trend.



B Appendix: Hyperparametergrid

Table 3: Explored hyperparameter grid for SchNet with 324 different combinations. Each model was trained with a learning rate of 0.001, batch size of 32, for 100 epochs, with 1000 test and 10000 train samples using the Ray library.

Parameter	**Values Explored**	**Default**	**Best Value**
Atom Embedding Dimension	32, 64, 128, 256	64	128
N Interactions	2, 4, 8	3	8
RBF Max	20, 30, 40	30	30
N RBF	100, 200, 400	300	100
Activation Function	SSP, LeakyRELU, GELU	SSP	SSP

Figure 7: Distribution of parameters which yielded a SchNet model with loss within the range of 0 and 3 within 100 epochs.