ÉCOLE NATIONALE DES PONTS ET CHAUSSÉES

IMI DEPARTMENT PROJECT 2024/2025

PREDICTING MOLECULAR DYNAMICS WITH AI

Score-matching last-layer finetuning for stable AI-predicted molecular dynamics

Alexandre DUSSOLLE, Kenji CHIKHAOUI, Téodora GOSPODARU

under the direction of
Mihai-Cosmin MARINICA et Clovis LAPOINTE
CEA Paris-Saclay

ABSTRACT

An important challenge of molecular dynamics (MD) is the stability of the simulations induced by the predicted forces and energies over time. Recent studies seem to imply that Graphical Neural Network (GNN) models trained with Mean Square Error (MSE) loss might not completely address this issue with more compute, as reduced MSE error doesn't always lead to better time-stability in MD trajectories. (The energy diverging is an example of such instabilities).

Previous works have shown that regularization of the Loss landscape sharpness improves time-stability [1]. This can be intuited quite well, but we hypothesize that a pure regularization might be too binding compared to approaching the true landscape gradients. This is why we propose to finetune pre-trained model on the Fisher divergence loss to more explicitly penalize irregularities in the derivatives of the predicted energy landscape.

By optimizing only the last-layer parameters with respect to the Fisher divergence, we avoid having to compute second order terms or roundabout way to get the Fisher divergence as done in most of score-matching approaches. We aim to make smoother force fields predictions and improve the long-term stability of induced MD simulations. While promoting a rather low-compute approach, as it reuses and improves previously trained models, without costly complete retraining.

I. INTRODUCTION

A. Context

Imagine a molecule made of multiple atoms, physics equations give how they will behave precisely, but if you want to consider a large number of atoms at once, it might be necessary (compute-wise) to consider an approximation of the energy and force fields. Such approximations are currently best done by Graph Neural Networks. Artificial intelligence is rapidly reshaping molecular science by enabling new ways to model, simulate, and understand complex molecular systems. There have been major breakthroughs recently, such as the AlphaFold2 model (Nobel Prize in Chemistry 2024). One of the next challenges is to simulate molecular dynamics over time, to make large-scale drugs or materials behaviors simulation possible, and reduce trials costs. However, for such models to be truly useful, they must produce energy and force predictions that remain stable over long timescales, a goal that current approaches like the state-of-the-art Graph Neural Network are only beginning to reach, and might require other approaches than simple RMSE minimization [5].

B. Problem

The main question in this project is how to improve the stability of simulations induced by GNN predictions of energy and forces. To do that, many approaches are possible, such as stability-aware training [2]. However, for a low-compute approach, we will focus on fine-tuning pretrained GNN models to minimize the Fisher divergence instead (of the RMSE). This idea is motivated by intuitions the sharpness of the loss landscape and its impact on time-stability. One key difficulty is the wide arrays of ways in which those instabilities appear, examples of that is found in the attempts made by last year's students [4]. The predicted energies would end up slowly diverging or completely explode after a certain time as we can see in the following graphics :

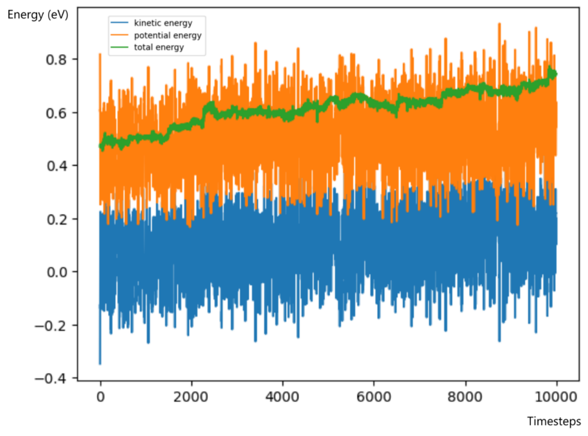

FIGURE 1. *Anomalous divergence of the predicted energy over time*

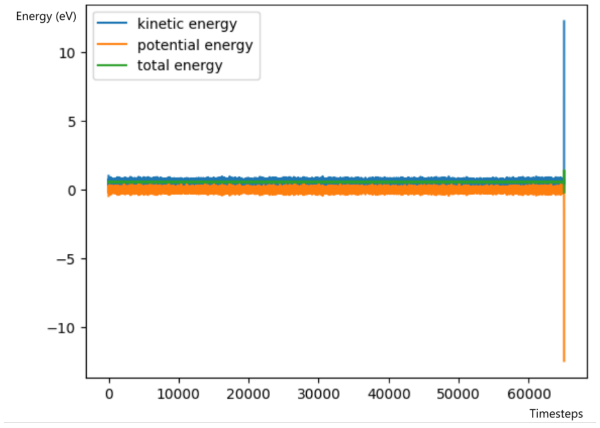

FIGURE 2. *Explosion of the predicted energy values after a certain number of timesteps, from last-year work*

Simulations in this paper will be carried out with a time step of one femtosecond (10^{-15}s). An useful estimate of minimal time, upon which we would need stable energy predictions, for them to be useful. is in the order of 100 nanosecond (10^{-9}s). Last year attempts seem to show instabilities way before this timestep, as they mostly happen around 0.1 ns (or before). In their current state, those are of little practical use, as diverging energies correspond to physical instabilities (e.g. the molecule exploding, even though it is supposed to be stable)

C. Problem framework and modeling

We consider a molecule of N atoms, described by their atomic positions : $\mathbf{x} = \{\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_N\} \subset \mathbb{R}^{3N}$ and corresponding atomic numbers $\mathbf{z} = \{z_1, z_2, \dots, z_N\}$.
We quickly list here the other physical quantities that we will use for studying molecules :

— **potential energy :** $U(\mathbf{x})$
— **forces :** $\nabla_{\mathbf{x}} U(\mathbf{x})$
The quantities the model output should approximate
— **probability density function :**

$$p(\mathbf{x}) = Z^{-1} e^{-\beta U(\mathbf{x})}$$

where Z is the partition function, $\beta^{-1} = k_B T$ and $U(\mathbf{x})$ is the potential energy function.
We will be working under the hypothesis that $\mathbf{x}$ follows a Boltzmann distribution. In reality it might not always the case, but we chose adapted datasets that verify the hypothesis to train and test our neural network. We will discuss this below, see Annexe 2.1.
— **score of the Boltzmann distribution :**

$$s(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x}) = -\beta \nabla_{\mathbf{x}} U(\mathbf{x})$$

Unknown as we can't differentiate $\log p(\mathbf{x})$.

Each quantity computed from $U(x)$ has an empirical counterpart computed from the predicted values $U_{\boldsymbol{\theta}(x)}$, where $\boldsymbol{\theta}$ denotes all the model's parameters.

D. Objectives

Initially the neural network takes the positions, atomic numbers (and other descriptors of the system) as input and predicts the corresponding forces and energies, those give their RMSE loss, and the model updates its weights from the (weighted) sum of those two losses. Some remarkable elements of its architecture are the following :

— Hyperparameters, the most significant ones being the temperature T, There's also the model's hyperparameters described in section II C.

— Layers, some ensure equivariances, some are the core of the GNN, giving the descriptors $\mathbf{D}(\mathbf{x})$ of an input x

— The last linear layer (θ vector) giving the predicted energy $U_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{D}(\mathbf{x})$

Trained and set to minimize the RMSE (energies $\&$ forces)

Our goal is to find another linear layer θ^* that improves the time-stability, from our hypotheses, one that minimizes the Fisher divergence instead of the RMSE is our preferred candidate. We ideally would like to do this without having to retrain the whole model (thus the descriptors), as it would be much easier for other researchers to adapt their own models that way.

To achieve that we need an explicit formula for the linear case. Which will replace the final linear layer of the model. Once we have this implemented, we need to evaluate if this does improves significantly the time-stability of the simulation induced by the energies and forces predictions.

II. LINEAR CASE SOLUTIONS

A. Computing θ^*

We freeze all parameters of the GNN, except the last linear layer, thus, we consider a D-dimensional machine learning model based on some descriptor functions

$$\mathbf{D}(\mathbf{x}) : \mathbb{R}^{3N} \to \mathbb{R}^D$$

for which the energy can be written as : $U_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{D}(\mathbf{x})$. We aim to find the optimal parameter $\boldsymbol{\theta}^* \in \mathbb{R}^D$ that minimizes the Fisher divergence between the true distribution $p(\mathbf{x})$ and the model distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$, where $p_{\boldsymbol{\theta}}(\mathbf{x}) = Z_{\boldsymbol{\theta}} e^{-\beta \boldsymbol{\theta}^\top D(\mathbf{x})}$. The Fisher divergence between $p(\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{x})$ is :

$$F(p, p_{\boldsymbol{\theta}}) = \frac{1}{2} \int p(\mathbf{x}) \left\| \nabla_{\mathbf{x}} \log p(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) \right\|^2 d\mathbf{x}$$

It can be rewritten using the score $s_{\boldsymbol{\theta}}(\mathbf{x})$:

$$F(p, p_{\boldsymbol{\theta}}) = \frac{1}{2} \int p(\mathbf{x}) \left\| s(\mathbf{x}) - s_{\boldsymbol{\theta}}(\mathbf{x}) \right\|^2 \, d\mathbf{x} \tag{1}$$

We can simplify the formula to have an easier to compute objective function $J(\boldsymbol{\theta})$:

$$\begin{aligned} F(p, p_{\boldsymbol{\theta}}) &= \frac{1}{2} \int p(\mathbf{x}) \left(\|s(\mathbf{x})\|^2 - 2 s(\mathbf{x})^\top s_{\boldsymbol{\theta}}(\mathbf{x}) + \|s_{\boldsymbol{\theta}}(\mathbf{x})\|^2 \right) d\mathbf{x} \\ &= \frac{1}{2} \mathbb{E}_p \left[\|s(\mathbf{x})\|^2 \right] - \mathbb{E}_p \left[s(\mathbf{x})^\top s_{\boldsymbol{\theta}}(\mathbf{x}) \right] + \frac{1}{2} \mathbb{E}_p \left[\|s_{\boldsymbol{\theta}}(\mathbf{x})\|^2 \right] \end{aligned}$$

The first term $\mathbb{E}_p \left[\|s(\mathbf{x})\|^2 \right]$ is independent of $\boldsymbol{\theta}$, so it can be ignored when minimizing with respect to $\boldsymbol{\theta}$. This gives us the following score function :

$$J(\boldsymbol{\theta}) = \mathbb{E}_{p(\mathbf{x})} \left[\frac{1}{2} \|s_{\boldsymbol{\theta}}(\mathbf{x})\|^2 - s_{\boldsymbol{\theta}}(\mathbf{x})^\top \nabla_{\mathbf{x}} \log p(\mathbf{x}) \right] \tag{2}$$

We develop the different terms :

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = -\beta \boldsymbol{\theta}^\top \mathbf{D}(\mathbf{x}) - \ln Z_{\boldsymbol{\theta}}$$

$$s_{\boldsymbol{\theta}}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) = -\beta \sum_{i=1}^{D} \theta_i \nabla_{\mathbf{x}} D_i(\mathbf{x}) = -\beta \mathbf{G}(\mathbf{x}) \boldsymbol{\theta}$$

is $3N$-dimensional function where $\mathbf{G}(\mathbf{x}) \in \mathbb{R}^{3N \times D}$ is the Jacobian matrix of $\mathbf{D}(\mathbf{x})$. And so :

$$\|s_{\boldsymbol{\theta}}(\mathbf{x})\|^2 = \beta^2 \boldsymbol{\theta}^\top \mathbf{G}(\mathbf{x})^\top \mathbf{G}(\mathbf{x}) \boldsymbol{\theta}$$

We can then write :

$$J(\boldsymbol{\theta}) = \frac{1}{2} \boldsymbol{\theta}^\top \mathbf{T} \boldsymbol{\theta} + \boldsymbol{\theta}^\top \mathbf{c} \tag{3}$$

where : $\mathbf{T} = \mathbb{E}_{p(\mathbf{x})} \left[\beta^2 \mathbf{G}(\mathbf{x})^\top \mathbf{G}(\mathbf{x}) \right]$ represents the covariance (Gram) matrix of the gradients of $\mathbf{D}(\mathbf{x})$ and

$$\mathbf{c} = \mathbb{E}_{p(\mathbf{x})} \left[\beta^2 \mathbf{G}(\mathbf{x})^\top \nabla_{\mathbf{x}} U(\mathbf{x}) \right]$$

represents the expected inner product between the gradients of $\mathbf{D}(\mathbf{x})$ and the force field $\nabla_{\mathbf{x}} U(\mathbf{x})$. The gradient of $J(\boldsymbol{\theta})$ is :

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbf{T} \boldsymbol{\theta} + \mathbf{c}$$

making the optimization very easy with a gradient descent or even analytical formal optimal solution :

$$\boldsymbol{\theta}^\star = -\mathbf{T}^{-1} \mathbf{c}$$

and explicitly, which can be computed :

$$\boldsymbol{\theta}^\star = -\left(\mathbb{E}_{p(\mathbf{x})} \left[\mathbf{G}(\mathbf{x})^\top \mathbf{G}(\mathbf{x}) \right] \right)^{-1} \left(\mathbb{E}_{p(\mathbf{x})} \left[\mathbf{G}(\mathbf{x})^\top \nabla_{\mathbf{x}} U(\mathbf{x}) \right] \right) \tag{4}$$

B. Difficulties

One of the first difficulty was how to approach a problem as broad as improving the time-stability (of the simulations), while not impeding too much on the accuracy. As mentioned in the introduction, there is a lot of different way to tackle this issue (or at least try to tackle it). Some are more explicitly focused on the stability, such as the work of Raja, Amin et al. [2], others like our approach, attack it from a derived way, this is the case in Ibayashi et al. [1], where the focus is on the loss sharpness regularization, which helps stability as an afterthought.

Deciding which way to go wasn't an easy choice, and we tried different ones before the Fisher divergence, you can see in annex 2.2 why we chose the Fisher divergence over the more usual Kullback-Leibler divergence. It's mostly due to the unpractical computations needed to get its gradient. kl, stability-aware training, etc

Other than the classic coding difficulty in the implementation, an important issue was that studying instabilities over long time-steps is costly, and to tinker and debug more easily we wanted to produce situations where it becomes unstable quickly enough to limit computational cost, but not unstable too quickly where both the default (RMSE minimization) and the Fisher model would diverge almost immediately, making the statistical study invalid. This is an important bottleneck for large study of how much it improves time-stability, as it's hard to predict beforehand which situations is inherently unstable and how the model will react to the situations. Here an example of a worst case scenario, where both are unstable too quickly.

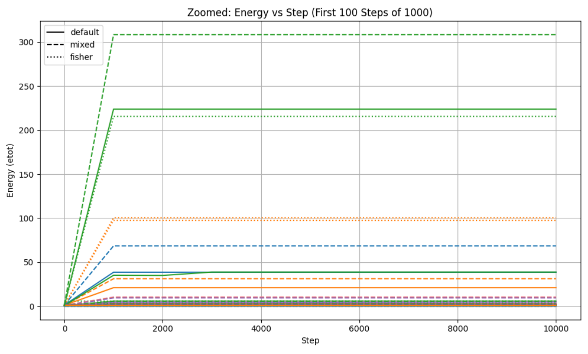

FIGURE 3. *Immediate explosion of the total energies (situation way too unstable, for both default and fisher predictions)*

Another difficulty, is on the Boltzmannian aspect of the dataset used to fine-tune on the Fisher divergence, to grasp why it is needed, see annex 2.1. However, it might be needed to study if (and if so how much) the results degrade when the Boltzmann hypothesis is breached.

C. Neural Network implementation

1. *Architecture outline*

The neural network we use is quite standard, mostly similar to that of last year [4]. In our case, the network is implemented using the E3x library, which provides a convenient and modular framework for constructing equivariant and invariant graph neural networks operating over 3D molecular data (for translation, rotation and reflexion). This is primordial to reduce the problem complexity (one of the main bug we had was a misplaced layer that killed those invariance leading to poor results). The molecular system is represented as a graph, where each atom is a node, and edges correspond to interatomic distances within a fixed cutoff (one of the hyperparameters). The neural network uses a message-passing mechanism that updates atomic features over several iterations, combining information from neighboring atoms to learn local chemical environments.

2. *Model's Hyperparameters*

The model has the following hyperparameters :

— **features** = 32 : Each atom is embedded into a 32-dimensional feature vector, which encodes its local chemical and geometric environment.
— **max_degree** = 2 : This controls how far the message-passing goes, while it is an hyperparameters, most of our tests show that on limited data, a max degree above 3 is way too unstable and hard to train for this project.
— **num_basis_functions** = K = 32 : it sets the number of radial basis functions used to represent interatomic distances in the interaction blocks
— **cutoff** = 3.0 (Ångströms) : The spatial range within which atom-atom interactions are considered. Atoms beyond this distance do not contribute to the message-passing or energy/force prediction.
— **batch_size** = 30 : The number of molecules processed simultaneously during training. A moderate batch size balances memory usage and learning stability.
— **learning_rate** = 0.01 : Controls the speed at which the model updates its weights during training, as usual.
— **forces_weight** = 1.0 : Weighting factor in the loss function to balance the relative importance of force and energy prediction. (setting it to 1.0 ensures that both targets are treated equally)

3. *Outline of a Graph Neural Network*

We recall that a molecule is represented by a set of atomic positions $\mathbf{x} = \{\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N\} \subset \mathbb{R}^{3N}$ and the corresponding atomic numbers $\mathbf{z} = \{z_1, z_2, \ldots, z_N\}$, which encode the identity of each atom. These atomic numbers are first mapped to an initial feature space through an embedding :

$$h_i^{(0)} = \text{Embed}(z_i) \in \mathbb{R}^D,$$

resulting in an initial node feature matrix.

$$\mathbf{H}^{(0)} = \text{Embed}(\mathbf{z}) \in \mathbb{R}^{N \times D}.$$

Since predictions will not be based on positions but rather on relative positions between the atoms (for equivariance), the neural network also computes, for each atomic pair (i, j), the relative displacement vector

$$\Delta \mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i.$$

This vector is expanded using a set of basis functions (e.g., Bessel functions, spherical harmonics, Bernstein polynomials) with a spatial cutoff to encode local geometric structure :

$$\mathbf{b}_{ij} = \text{Basis}(\Delta \mathbf{r}_{ij}) \in \mathbb{R}^K,$$

where K is the number of basis functions.

While using the relative displacement is quite intuitive (translation invariance, etc), the projection unto basis functions might be harder to grasp, it's mostly a way to inscribe human expert knowledge. It reduce the problem complexity by leveraging the symmetries and equivariance using the E3x architectures [7].

The GNN then uses L iterations of message passing to update atomic features and obtain the final atomic representation. At each iteration ℓ, the feature of atom i is updated using messages from its neighbors :

$$h_i^{(\ell+1)} = h_i^{(\ell)} + \phi\left(h_i^{(\ell)}, \left\{\psi(h_j^{(\ell)}, \mathbf{b}_{ij})\right\}_{j \in \mathcal{N}(i)}\right),$$

where :

- — ψ is the message function that combines the neighbor's features and geometric information.
- — ϕ is the update function that aggregates messages, typically including non-linear layers, normalization, and residual connections.
- — $\mathcal{N}(i)$ denotes the neighbors of atom i within the cutoff.

After L iterations, we obtain the final atomic representations :

$$h_i^{(L)} \in \mathbb{R}^D.$$

Each of which contributes to the total molecular energy via a linear readout :

$$\epsilon_i = \boldsymbol{\theta}_r^\top h_i^{(L)} + b_{z_i},$$

where $\boldsymbol{\theta}_r$ are the weights of the last layer of the GNN, so the weights used to compute energy and forces. b_{z_i} is an atomic bias specific to the element. The total predicted energy is then :

$$U_{\boldsymbol{\theta}_{GNN}}(\mathbf{x}) = \sum_{i=1}^{N} \epsilon_i.$$

where θ_{GNN} are all the weights, including those that will stay fixed as we compute $\boldsymbol{\theta}^*$ which will replace only $\boldsymbol{\theta}_r$.

Forces on atoms are obtained by automatic differentiation of the energy with respect to atomic positions :

$$-\nabla_{\mathbf{x}} U_{\boldsymbol{\theta}_{GNN}}(\mathbf{x}).$$

4. *Adding $\boldsymbol{\theta}^*$*

To calibrate the model or to perform further tasks such as score matching, we define a global descriptor :

$$\mathbf{D}(\mathbf{x}) = \left(\sum_{i=1}^{N} h_i^{(L)}\right) \oplus \left(\sum_{i=1}^{N} b_{z_i}\right) \in \mathbb{R}^{D+1},$$

where $\oplus$ denotes vector concatenation.

This descriptor is used to linearly recalibrate the energy :

$$U_{\text{cal}}(\mathbf{x}) = \boldsymbol{\theta}_c^\top \mathbf{D}(\mathbf{x}),$$

Where $\boldsymbol{\theta}_c$ is the new value with which we will replace $\boldsymbol{\theta}_r$

The Jacobian of the descriptor with respect to the positions, denoted $\mathbf{G}(\mathbf{x}) = \nabla_{\mathbf{x}} \mathbf{D}(\mathbf{x}) \in \mathbb{R}^{3N \times (D+1)}$, is used in score matching to estimate $\boldsymbol{\theta}^*$, the optimal valeue of $\boldsymbol{\theta}_c$ for the Fisher Divergence (see section II A).

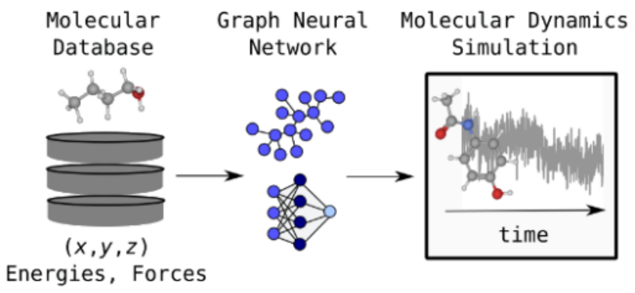

FIGURE 4. process overview from *Stocker, Gasteiger et al.* [6])

III. RESULTS

We first implemented a working pipeline of GNN training and Fisher fine-tuning, the goal here was not to have the perfect neural network for the task, but rather have a wide population of study of reasonable predictors. Here's below the training and validation losses of the differents runs we did (varying hyperparameters, and for each set of hyperparameters, multiple instance trained).

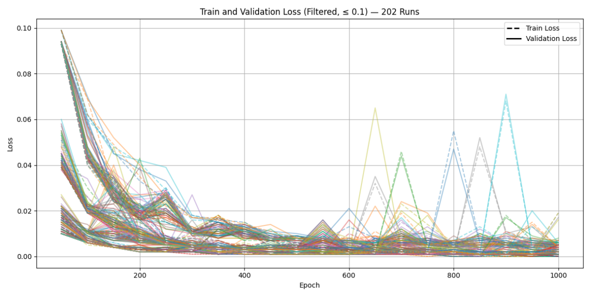

FIGURE 5. *GNN Training and Validation RMSE Losses*

The sort of 3 groups we see in the curves depend on how much training examples they had access to. While we mostly see converging losses, we do have some sets of hyperparameters where the GNN default loss don't converge, but it's a mostly negligible portion of them (they're quite visible as the hundreds of others align).

There's something to be said on the fact that those loss diverging variations increase as the number of training example increases, for a low number of training examples (here 200), we have a relatively smooth descent.

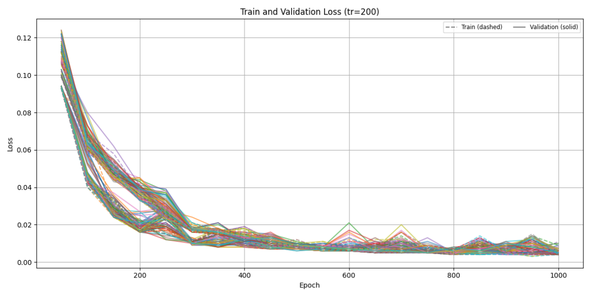

FIGURE 6. *GNN Training and Validation RMSE Losses, for 200 training examples, relatively converging.*

While for bigger training examples, this phenomenon seems more frequent, as shown below. While a bit worrying, it also means that the improvement of the time-stability when fine-tuned for the Fisher divergence is not just due to being presented with more data examples, as this wouldn't necessarily make the GNN more accurate. This highlights the importance of our method over compute-only approaches.

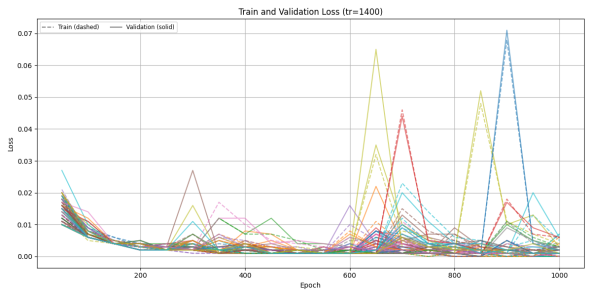

FIGURE 7. *GNN Training and Validation RMSE Losses for 1400 different training examples, more frequent non-converging losses*

Afterward, we used those models to conduct one of our main experiment. Where we predict the molecular dynamics of a system of 600 molecules of ethanol. The quantity of interest is their average energy drift over time, to see if our method of stabilization by fine-tuning on the Fisher divergence has had a significant impact. As we can see in the two graphics below, the Fisher model does greatly improves the coherence of the simulations.

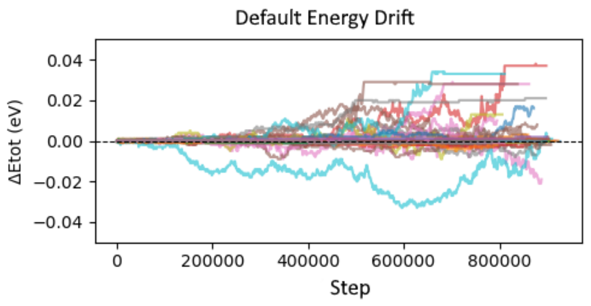

FIGURE 8. *Drift of the total energy over time in default setting, for different situations*

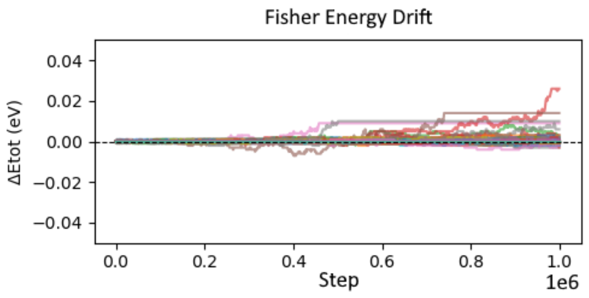

FIGURE 9. *Same situations but with Fisher-minimizing last layer*

Those runs are quite costly to compute, as they imply the training of new GNN from scratch when changing the hyperparameters, but necessary for statistically significant test of our results.

IV. CONCLUSION

In this paper, we fine-tuned the last layer of molecular dynamics models using the Fisher divergence of the energy distribution as the loss function. This approach is motivated by the fact that the Fisher divergence compares the force fields (i.e., the gradients of the energy), with the aim of improving the stability of the resulting molecular simulations.

Furthermore, in the case of an ideal Boltzmann distribution of the molecular configurations, the optimization problem for the Fisher divergence becomes linear, which allows for a closed-form solution for the optimal last-layer parameters.

Extensive experiments across differents hyperparameters settings over a GNN, trained with a dataset of 600 ethanol molecules in different arrangements, showed that our fine-tuned models were more stable than the default model by a large margin. The observed energy drift from simulations using the fine-tuned models is much lower than that of the default models. This implies that our fine-tuning greatly improves the stability of the molecular dynamic simulations.

The method introduced in this paper is particularly promising because it is general, lightweight, and highly compatible with existing models. It offers a simple yet effective way to enhance the physical reliability of advanced molecular dynamics prediction models, without the need for retraining or high computational cost. As such, it opens up a practical path toward more stable and physically meaningful AI-driven molecular simulations.

Nevertheless, this method has yet to be tested on advanced MD models to test its effectivity in real use-cases. Furthermore, since our approach requires our structures to have a certain energy distribution, the effectiveness on more complex distributions is not obvious. Even if our intuition suggests that the reproducibility of our results on the other simulation and models is possible, it is still to be proven by experiments.

V. ACKNOWLEDGMENTS

Many thanks to our project leader, Mihai-Cosmin Marinica, researcher at the CEA, for his valuable help and support throughout this project, his ideas, and his commitment. We would also like to express our sincere gratitude to Clovis Lapointe of the CEA for his help. Their contributions were essential to the successful completion of this project, and we are truly grateful for their guidance and involvement.

ANNEX 1

References

[1] H. Ibayashi, T. M. Razakh, L. Yang, T. Linker, M. Olguin, S. Hattori, Y. Luo, R. K. Kalia, A. Nakano, et al., *Allegro-legato : Scalable, fast, and robust neural-network quantum molecular dynamics via sharpness-aware minimization*, International Conference on High Performance Computing, Springer (2023).

[2] S. Raja, I. Amin, F. Pedregosa, and A. Krishnapriyan, *Stability-Aware Training of Machine Learning Force Fields with Differentiable Boltzmann Estimators*, Department of Computer Science, UC Berkeley ; Google DeepMind ; LBNL (2023).

[3] A. Hyvärinen, *Estimation of Non-Normalized Statistical Models by Score Matching*, Journal of Machine Learning Research, vol. 6, no. 24, pp. 695–709, 2005. Available at : `http://jmlr.org/papers/v6/hyvarinen05a.html`

[4] M. El Yaagoubi and F. Fourreau, *PROJET IMI : Instabilité de la dynamique hamiltonienne avec des potentiels appris par machine*, 15 mai 2024.

[5] X. Fu, Z. Wu, W. Wang, T. Xie, S. Keten, R. Gomez-Bombarelli, and T. Jaakkola, *Forces are not Enough : Benchmark and Critical Evaluation for Machine Learning Force Fields with Molecular Simulations*, Massachusetts Institute of Technology ; Northwestern University ; Microsoft Research (2023). Available at : `https://openreview.net/forum?id=A8pqQipwkt`

[6] S. Stocker, J. Gasteiger, F. Becker, S. Günnemann, and J. T. Margraf, *How Robust are Modern Graph Neural Network Potentials in Long and Hot Molecular Dynamics Simulations ?*, Fritz-Haber-Institute of the Max-Planck-Society ; Technical University of Munich (2022). DOI : 10.1063/5.0083880

[7] O. T. Unke and H. Maennel, *E3x :* $\mathrm{E}(3)$*-Equivariant Deep Learning Made Easy*, arXiv preprint arXiv :2401.07595, 2024. Available at : `https://arxiv.org/abs/2401.07595`

ANNEX 2

1. Why the Boltzmann distribution hypothesis ?

In reality, many molecules do not have a Boltzmann probability distribution function. In that case, we can't get the true score $s(\mathbf{x})$. Therefore, we have to define an objective function $J(\boldsymbol{\theta})$ that does only depend on the predicted score. It is not the case in (2), where the cross term involves $s(\mathbf{x})$. We can handle it using integration by parts :

$$\begin{aligned}\mathbb{E}_p\left[s_{\boldsymbol{\theta}}(\mathbf{x})^\top s(\mathbf{x})\right] &= \int p(\mathbf{x}) s_{\boldsymbol{\theta}}(\mathbf{x})^\top \nabla_{\mathbf{x}} \log p(\mathbf{x})\, d\mathbf{x} \\ &= \int s_{\boldsymbol{\theta}}(\mathbf{x})^\top \nabla_{\mathbf{x}} p(\mathbf{x})\, d\mathbf{x}\end{aligned}$$

Applying integration by parts (assuming boundary terms vanish), we get :

$$\begin{aligned}\int s_{\boldsymbol{\theta}}(\mathbf{x})^\top \nabla_{\mathbf{x}} p(\mathbf{x})\, d\mathbf{x} &= -\int \left(\nabla_{\mathbf{x}} \cdot s_{\boldsymbol{\theta}}(\mathbf{x})\right) p(\mathbf{x})\, d\mathbf{x} \\ &= -\mathbb{E}_p\left[\nabla_{\mathbf{x}} \cdot s_{\boldsymbol{\theta}}(\mathbf{x})\right]\end{aligned}$$

Putting everything together, we obtain the score matching objective :

$$J(\boldsymbol{\theta}) = \frac{1}{2}\mathbb{E}_p\left[\|s_{\boldsymbol{\theta}}(\mathbf{x})\|^2\right] + \mathbb{E}_p\left[\nabla_{\mathbf{x}} \cdot s_{\boldsymbol{\theta}}(\mathbf{x})\right] \tag{5}$$

All the developed quantities remain the same and we now also have :

$$\nabla_{\mathbf{x}} \cdot s_{\boldsymbol{\theta}}(\mathbf{x}) = -\beta \boldsymbol{\theta}^\top \mathbf{v}(\mathbf{x})$$

where $\mathbf{v}(\mathbf{x}) \in \mathbb{R}^D$ is defined as :

$$\mathbf{v}(\mathbf{x}) = \begin{bmatrix} \Delta_{\mathbf{x}} D_1(\mathbf{x}) \\ \Delta_{\mathbf{x}} D_2(\mathbf{x}) \\ \vdots \\ \Delta_{\mathbf{x}} D_D(\mathbf{x}) \end{bmatrix}$$

with $\Delta_{\mathbf{x}} D_i(\mathbf{x}) = \operatorname{tr}\left(\nabla^2_{\mathbf{x}} D_i(\mathbf{x})\right)$ being the Laplacian (i.e., the trace of the Hessian) of the scalar descriptor $D_i(\mathbf{x})$. The objective $J(\boldsymbol{\theta})$ can then be rewritten :

$$J(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^\top \mathbf{T}\boldsymbol{\theta} + \boldsymbol{\theta}^\top \mathbf{b} \tag{6}$$

where : $\mathbf{T}$ is the same as in (3) and $b = \mathbb{E}_{p(\mathbf{x})}\left[-\beta \mathbf{v}(\mathbf{x})\right]$. The gradient is now :

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbf{T}\boldsymbol{\theta} + \mathbf{b}$$

making the optimization very easy with a gradient descent or even analytical formal optimal solution, which becomes

$$\boldsymbol{\theta}^\star = -\mathbf{T}^{-1}\mathbf{b}$$

For the case where we have a large collection of points $(\mathbf{x}_m)_{m=1,\ldots,M}$ sampled with the measure $p(\mathbf{x})$ the T and b quantities are simple to estimate as :

$$T = \frac{\beta^2}{M}\sum_m \mathbf{G}(\mathbf{x}_m)^\top \mathbf{G}(\mathbf{x}_m)$$
$$b = -\frac{\beta}{M}\sum_m \mathbf{v}(\mathbf{x}_m) = -\frac{\beta}{M}\sum_m \Delta\mathbf{D}(\mathbf{x}_m)$$

where $\Delta\mathbf{D}(\mathbf{x})$ is a vectors of Laplacians $(\Delta D_1(\mathbf{x}), \Delta D_2(\mathbf{x}), \ldots, \Delta D_D(\mathbf{x}))^\top$. Since $\mathbf{D}(\mathbf{x})$ are the descriptors of our neural network, the computational cost to compute their laplacians is very high and we cannot get them in practice. This is why we needed the hypothesis that $p(\mathbf{x})$ was a Boltzmann distribution, even though in reality it is not always the case and we had to adapt the training datasets to satisfy the hypothesis.

2. *Physical accuracy*

Another approach when fitting $p_{\boldsymbol{\theta}}(\mathbf{x})$ could be to minimize the Kullback–Leibler divergence (D_{KL}) :

$$D_{\mathrm{KL}}(p, p_{\boldsymbol{\theta}}) = \int p(\mathbf{x}) \ln \frac{p(\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x})}\, d\mathbf{x} \tag{7}$$

That is because D_{KL} is linked to free energy in the following way :

$$\Delta F = F_{\boldsymbol{\theta}} - F = -\beta^{-1}(\log Z_{\boldsymbol{\theta}} - \log Z) \tag{8}$$

So :

$$\beta^{-1} D_{\mathrm{KL}}(p, p_{\boldsymbol{\theta}}) = -\Delta F - \beta \mathbb{E}_{p(\mathbf{x})}\left[U(\mathbf{x}) - U_{\boldsymbol{\theta}}(\mathbf{x})\right] \tag{9}$$

From a physical point of view, the KL divergence has a clear interpretation : it corresponds to the excess free energy introduced when approximating the true distribution $p(\mathbf{x})$ with $p_{\boldsymbol{\theta}}(\mathbf{x})$. However, minimizing this objective directly is not practical in our case because the partition function $Z_{\boldsymbol{\theta}}$ is generally intractable to compute or differentiate in high dimensions, making gradient-based optimization of the KL divergence infeasible. The Fisher Divergence is a good alternative because it still respects the physical structure of the problem even though it is not clearly linked to the free energy. That is due to the fact that it optimizes score functions which contain information about the local geometry of the distribution (like forces for instance).