Group Meeting - February 5, 2021

Paper review & Research progress

Truong Son Hy *

*Department of Computer Science
The University of Chicago

Ryerson Physical Lab

Content

- Research update
- Literature review:
  1. **Molecule Optimization by Explainable Evolution** (ICLR 2021), https://openreview.net/pdf?id=jHefDGsorp5
  2. **Boltzmann Generators – Sampling Equilibrium States of Many-Body Systems with Deep Learning**, https://arxiv.org/pdf/1812.01729v2.pdf
  3. **Conformation-Guided Molecular Representation with Hamiltonian Neural Network** (ICLR 2021), https://openreview.net/pdf?id=q-cnWaaoUTH

Research update

1. TensorFlow API for the old C++/CPU CG (modified from the old FastCG) starts working and tested. I am going to implement the graph-based Cormorant in TF for 3D conformation generation.
2. E3NN (PyTorch): `https://github.com/e3nn/e3nn`
3. I still have a technical problem with IPython/Jupyter notebook, but after it resolved, we can tackle **PSI4** for DFT computation (as an alternative to evaluate 3D conformations), and **Boltzmann Generators**.

Paper 1

Molecule Optimization by Explainable Evolution (ICLR 2021)
Binghong Chen, Tianzhe Wang, Chengtao Li, Hanjun Dai, Le Song
`https://openreview.net/pdf?id=jHefDGsorp5`

Note: I think the idea of subgraph recognition can be applied here.

Proposal

Proposal

A novel algorithm for optimizing molecule properties via an **Expectation Maximization (EM)-like explainable evolutionary process**. Alternate between two stages:

- Explainable local search which identifies rationales, i.e. critical subgraph patterns accounting for desired molecular properties.
- Molecule completion which explores the larger space of molecules containing good rationales.

Note

In simple words: mimics the process of molecule optimization by human experts.

Overall

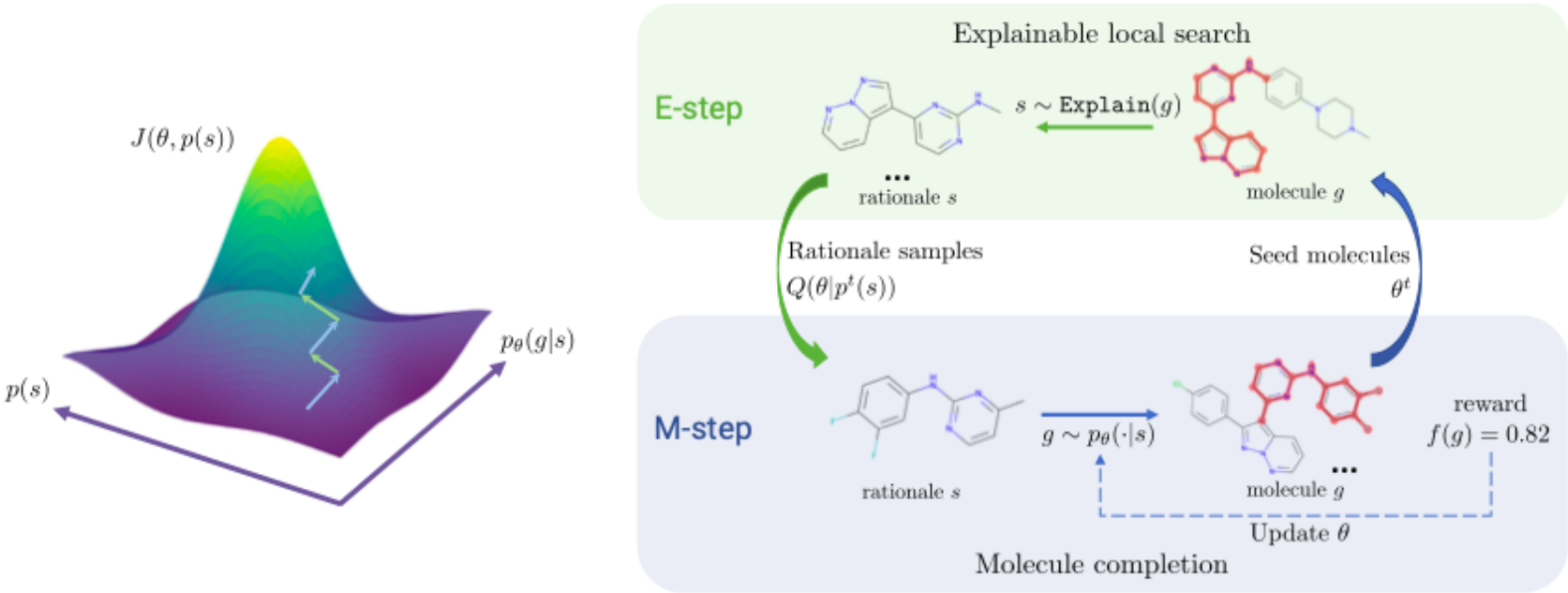


Figure 1: Overview of our EM-like evolution-by-explanation algorithm. Left: climbing up the energy landscape $J(\theta, p(s))$ by alternatively taking an E-step and M-step. Right: illustrations for the E-step and M-step. In the E-step of time t, we draw samples from $Q(p(s)|\theta^{t-1})$ to approximate $Q(\theta|p^t(s))$ using rationales extracted from the seed molecules via an explainable model. Then in the M-step, we optimize $Q(\theta|p^t(s))$ w.r.t. θ, *i.e.* pushing the graph completion model $p_\theta(\cdot|s)$ towards generating higher scoring molecules conditioned on the rationale samples.

Molecule optimization

Molecule optimization

Given a scoring function $f : \mathcal{G} \to [0, 1]$, and a set of seed molecules $\mathcal{G}_0 \subset \mathcal{G}$, the goal is to learn a molecule generative model $p(g)$ such that the expected score of the generated molecules is maximized:

$$\max_{p(.)} \mathbb{E}_{g \sim p(.)}[f(g)] = \int_{g \in \mathcal{G}} p(g) f(g) dg$$

The set of rationales

When experts are optimizing a molecular property, they will first look for **substructures** that result in the formation of that property, and use them as the **foundation** for building novel molecules. These **subgraphs** are called **rationales**:

$$\mathcal{S} = \{s | \exists g \in \mathcal{G}, \text{s.t.} \quad s \quad \text{is a subgraph of} \quad g\}$$

Rationale-based hierarchical generative model (1)

Generative model

1. Sample rationales s from a rationale distribution $p(s)$.
2. Molecules g will be generated according to conditional distribution $p_\theta(g|s)$:

$$p_\theta(g) = \int_{s \in \mathcal{S}} p(s) p_\theta(g|s) ds$$

where θ is the learnable parameters.

Graph completion model

$p_\theta(g|s)$ is a graph completion model from rationale s. We use a latent variable model with a Gaussian prior $p(z)$:

$$p_\theta(g|s) = \int_z p(z) p_\theta(g|s,z) dz$$

where $p_\theta(g|s,z)$ is a variant of the GraphRNN - **conditional graph generation on subgraphs**.

Rationale-based hierarchical generative model (2)

To improve the **diversity** of the generated molecules, we will also regularize the entropy of the rationale distribution $p(s)$, leading to the following diversity promoting objective function:

$$J(\theta, p(s)) = \mathbb{E}_{g \sim p_\theta(.)}[f(g)] + \lambda \cdot \mathbb{H}[p(s)]$$

where $\lambda > 0$ is a hyper-parameter.

Alternating optimization algorithm

Generic overview

We seek to optimize $p_\theta(g|s)$ and $p(s)$ in an alternating fashion, akin to the EM algorithm:

1. **Expectation step** (E-step): updating the rationale distribution $p(s)$.
2. **Maximization step** (M-step): improving the molecule completion model $p_\theta(g|s)$.

E-step

We want to maximize the objective J with respect to the rationale distribution $p(s)$ given $p_{\theta^{t-1}}(g|s)$:

$$p^t(s) = \arg\max_{p(s)} Q(p(s)|\theta^{t-1}) = \frac{1}{Z_\theta} \exp\left(\frac{1}{\lambda}\mathbb{E}_{g\sim p_{\theta^{t-1}}(\cdot|s)}[f(g)]\right)$$

We will maintain a finite support set $\mathcal{S}^t$, which is obtained from an **explanable graph model** that takes a graph input g and outputs the corresponding rationale s which explains why the graph g can obtain a high property score $f(g)$:

$$\mathcal{S}^t = \bigcup_{i=1}^{t} \left\{ \text{Explain}(g) : g \in \mathcal{G}^i \right\}$$

that will be treated as particle locations for representing $p^t(s)$.
This is the place I think a better subgraph recognition model can contribute.

M-step

With $\{s_i\}_{i=1}^m$ from $p^t(s)$, the MC estimate of the objective:

$$Q(\theta|p^t(s)) \approx \sum_{i=1}^{m} \int p_\theta(g|s_i) f(g) dg + \text{constant}$$

We maximize it with respect to the parameters θ using REINFORCE:

$$\theta^t \leftarrow \theta^{t-1} + \alpha \frac{1}{m} \sum_{i=1}^{m} f(g_i) \nabla \log p_{\theta^{t-1}}(g_i|s_i)$$

$$\alpha > 0, \qquad g_i \sim p_{\theta^{t-1}}(\cdot|s_i)$$

After the parameter is updated to θ^t, we will sample a seed set of molecules $\mathcal{G}^t = \{g_i\}_{i=1}^{n_s}$ from $p_{\theta^t}(g|s)$ by completing the rationale samples $\{s_i\}_{i=1}^m$:

$$g_i \sim g_{\theta^t}(\cdot|s), \qquad s \sim \text{Uniform}(\{s_1, .., s_m\})$$

Algorithm 1

Algorithm 1: Molecule Optimization by Explainable Evolution (`MolEvol`)

Input: Seed molecules $\mathcal{G}^0$, pretrained graph completion model $p_\theta(g|s)$ on ChEMBL.

Initialize $S^0 = \{\}$.
for $t \leftarrow 1$ **to** N_{rounds} **do**
  $\mathcal{S}^t = \mathcal{S}^{t-1} \cup \{\texttt{Explain}(g) : g \in \mathcal{G}^{t-1}\}$.
  Sample $s_1, s_2, \cdots, s_m$ from $\mathcal{S}^t$ using Eq (7) with self-normalization.
  for $j \leftarrow 1$ **to** N_{epochs} **do**
    Sample $g_1, \cdots, g_m$ from $p_\theta(g|s_1), \cdots, p_\theta(g|s_m)$ respectively.
    Update θ with REINFORCE (Eq (10)).
  Sample seed molecules $\mathcal{G}_t$ with Eq (11).
return $p_\theta(g)$

Explainer (subgraph recognition model)

Variational Objective. We want to learn an explainer for the conditional distribution $\mathbb{P}(Y = 1|g) \triangleq f(g)$ (treating $f(g)$ as a probability), with random variables $Y \in \{0, 1\}$ where $Y = 1$ indicates that the molecule has the property, and 0 otherwise. We will learn a graph vertex sampler $h_\phi(g)$ jointly with a variational approximation $\mathbb{Q}(Y|g)$ of $\mathbb{P}(Y|g)$, such that the mutual information between Y and s is maximized

$$\max_{h_\phi(\cdot),\mathbb{Q}} \mathbb{E}_{Y\sim\mathbb{P}(\cdot|g)}\Big[\log \mathbb{Q}(Y \mid s)\Big], \quad \text{such that } s = (\mathcal{U}, \mathcal{E}_g^{\mathcal{U}}) \text{ and } \mathcal{U} \sim h_\phi(g). \tag{12}$$

Note

- In my understanding, this is an unsupervised way to learn the set of rationales (subgraphs).
- The graph vertex sampler $h_\phi(g)$ actually can be understood as a probability distribution over the set of vertices: $h_\phi(g)_v$ is the probability that vertex v contributes to our desired property.
- If we sample the first k vertices from $h_\phi(g)$, we get a (maybe **disconnected**) rationale.

Algorithm 2

Algorithm 2: $\texttt{Explain}_\phi(g)$

Input: Molecule g, vertex sampling policy ϕ.
$h_\phi(g) = \text{Softmax}(\text{FC}_\phi(\text{MPN}_\phi(g)))$.
Sample $\mathcal{U} \sim h_\phi(g)$ with Gumbel-softmax trick.
$s' = \texttt{Expand}((\mathcal{U}, \mathcal{E}_g^{\mathcal{U}}))$ as defined in Eq (13).
return s'

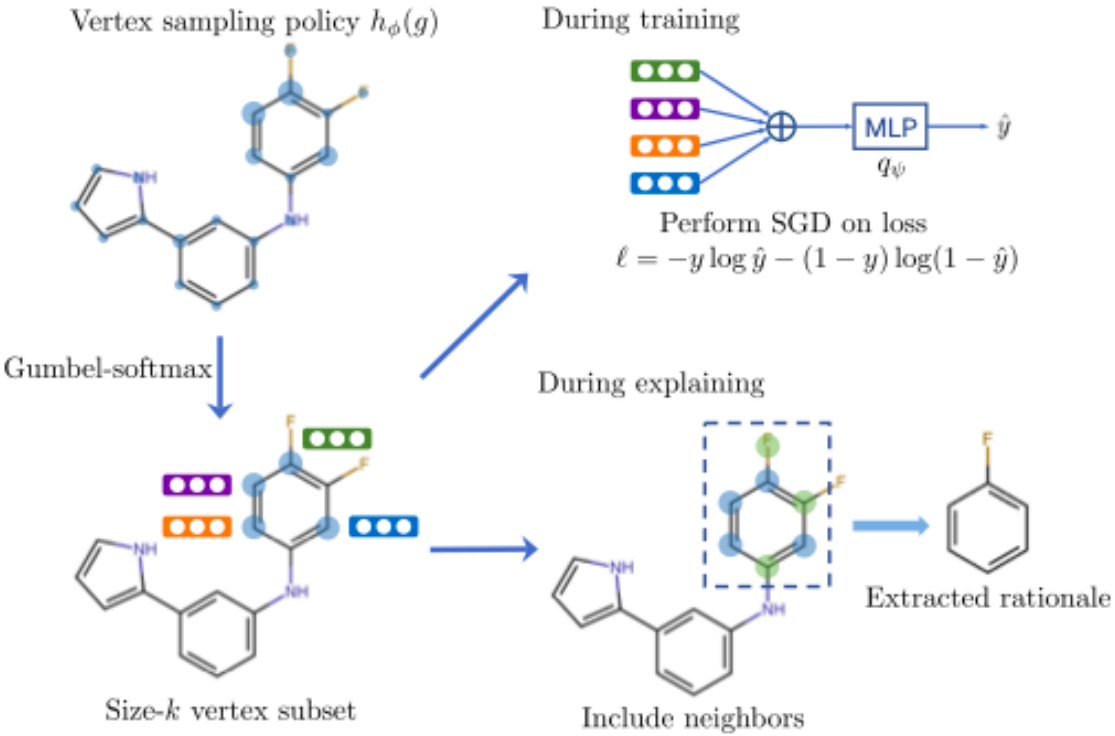


Figure 2: Steps of the explaining process (Alg. 2). The explainer is a subgraph selector containing two steps. First, a vertex sampling policy $h_\phi(g)$ is computed (top-left). Then k vertices are selected using the Gumbel-softmax trick (bottom-left). During training, the embeddings on the selected vertices are pooled together and fed into a MLP q_ψ which predicts the property score (top-right). During explaining, The induced subgraph of the selected vertices and their neighbors is extracted as the predicted rationale.

Experiments (1)

Table 1: Results on multi-property molecule optimization. `MolEvol` is compared with three variants and four baselines in terms of success rate, novelty, diversity and an overall metric (QNU). The diversity of MSO and GA-D(t) is not reported here due to their extremely low novelty scores.

Algorithm	MolEvol	[MCTS]	[FixM]	[FixR]	RationaleRL	REINVENT	MSO	GA-D(t)
Success rate	**93.0%**	77.7%	67.3%	66.3%	61.1%	46.6%	57.7%	62.0%
Novelty	**75.7%**	72.5%	67.4%	54.6%	57.4%	66.4%	28.6%	19.4%
Diversity	0.681	0.707	0.723	0.727	**0.749**	0.666	-	-
QNU	**52.7%**	47.4%	39.3%	28.3%	29.5%	7.4%	16.4%	12.0%

Experiments (2)

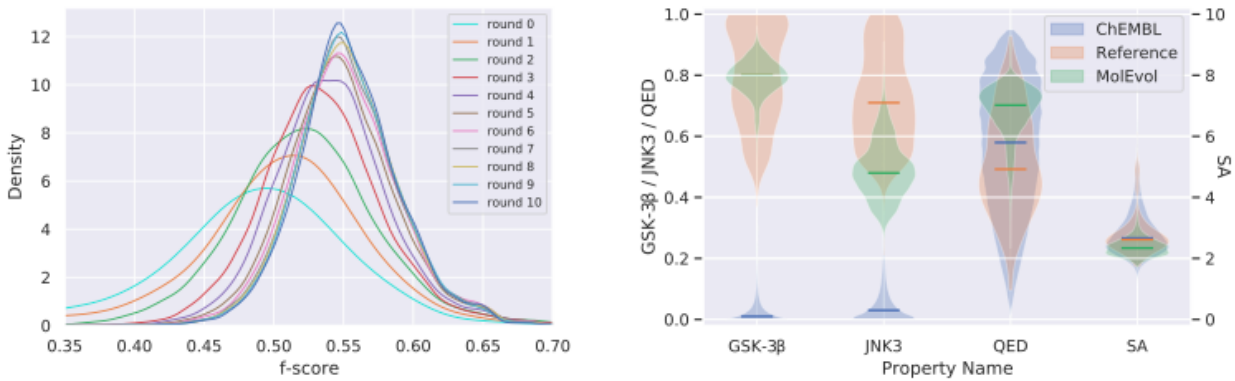


Figure 3: Property score distribution of the generated molecules. Left: the evolution of the f-score distribution of `MolEvol` over the number of iterations. Right: the distribution of four property scores of our generated molecules, the ground truth molecules in the reference set, and the molecules in ChEMBL. The higher the better for JNK3/GSK-3β/QED, the lower the better for SA.

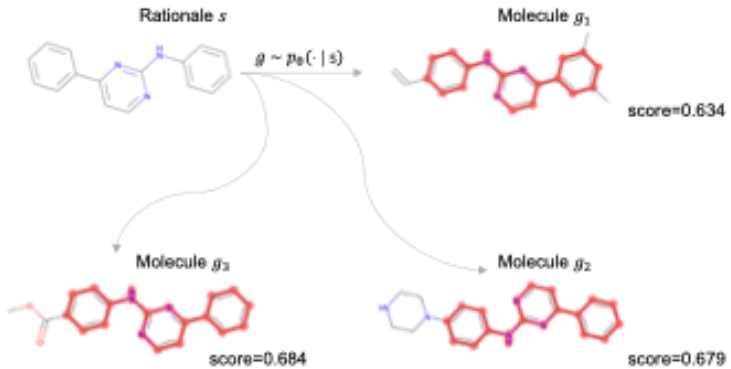


Figure 4: An example of rationale and corresponding generated molecules with f-scores.

Paper 2

Boltzmann Generators – Sampling Equilibrium States of Many-Body Systems with Deep Learning

Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu

`https://arxiv.org/pdf/1812.01729v2.pdf`

Note: As last time we discussed about learning the Boltzmann distribution for the task of 3D conformation generation, this paper is prerequisite for **Equivariant Flows**.

Proposal

Problem

Computing equilibrium states in condensed-matter many-body systems. Molecular dynamics is computationally expensive.

Proposal

Boltzmann Generators:

- To generate unbiased one-shot equilibrium samples of representative condensed matter systems and proteins.
- Uses neural networks to learn a coordinate transformation of the complex configurational equilibrium distribution to a distribution that can be easily sampled. (The authors meant normalizing flows.)

Metric = Free energy difference.

Background

Observation

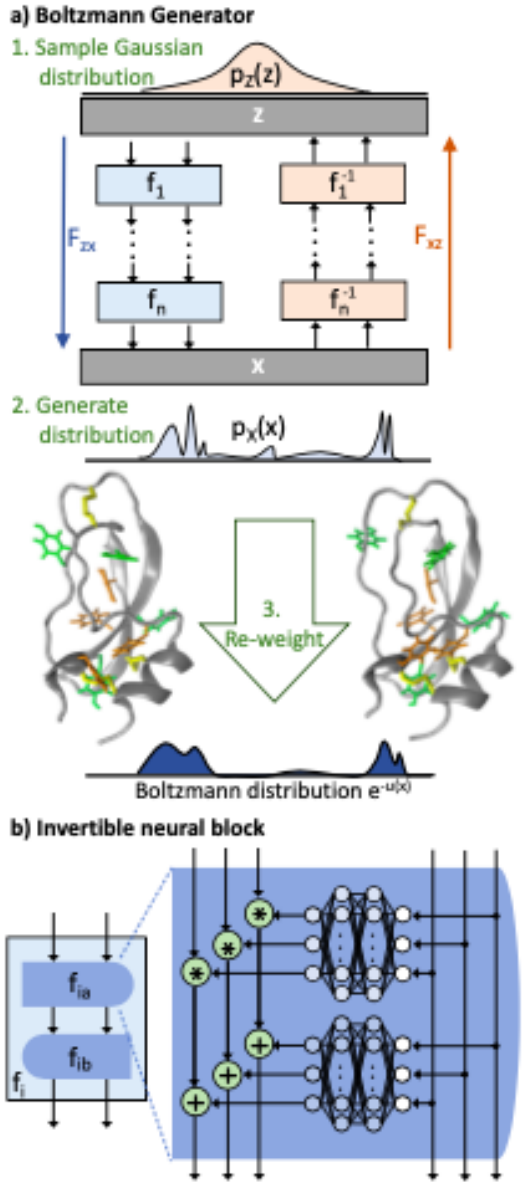
- Under a wide range of conditions, the equilibrium probability of a microscopic configuration $\boldsymbol{x}$ (setting of all spins, positions of all protein atoms, etc.) is proportional to $e^{-u(\boldsymbol{x})}$, the well-known Boltzmann distribution.
- The dimensionless energy $u(\boldsymbol{x})$ contains the potential energy of the system, the temperature and optionally other thermodynamic quantities.

Trajectory methods

Markov Chain Monte Carlo (MCMC) or **Molecular Dynamics** (MD):

- Conventional methods for sampling from the Boltzmann distribution.
- Make tiny changes to $\boldsymbol{x}$ in each step. Require many simulation steps.
- Biased user-defined order parameters called **reaction coordinates** to enhance sampling of the rare events.

Overview


a) Boltzmann Generator
1. Sample Gaussian distribution
$p_Z(z)$
z
f_1
f_1^{-1}
F_{zx}
F_{xz}
f_n
f_n^{-1}
x
2. Generate distribution
$p_X(x)$
3. Re-weight
Boltzmann distribution $e^{-u(x)}$
b) Invertible neural block
f_{ia}
f_{ib}
f_i

Boltzmann Generators

1. Given a sample $\boldsymbol{z}$ from a simple prior (Gaussian), an **invertible** neural network transformation $F_{zx}(\boldsymbol{z})$ provides a configuration $\boldsymbol{x}$ which has a high Boltzmann weight, from a distribution $p_X(\boldsymbol{x})$ that is similar to the target one.
2. Reweight the generated distribution $p_X(\boldsymbol{x})$ to the Boltzmann distribution $e^{-u(\boldsymbol{x})}$:

$$w(\boldsymbol{x}) = e^{-u(\boldsymbol{x})}/p_X(\boldsymbol{x})$$

for every sample $\boldsymbol{x}$.

Boltzmann Generators – Training by Energy

KL divergence between the proposal distribution $p_X(\boldsymbol{x})$ and Boltzmann distribution $e^{-u(\boldsymbol{x})}$ can be computed as the expectation over samples $\boldsymbol{z}$:

$$J_{KL} = \mathbb{E}_{\boldsymbol{z}}[u(F_{zx}(\boldsymbol{z})) - \log R_{zx}(\boldsymbol{z})]$$

where:

- $u(F_{zx}(\boldsymbol{z}))$ is the energy of the generated configuration.
- R_{zx} is the determinant of the Boltzmann Generator's Jacobian matrix.

Observation

- $\mathbb{E}_{\boldsymbol{z}}[u(F_{zx}(\boldsymbol{z}))]$ is the mean potential energy, that tries to **minimize the energy** and trains BG to sample **low-energy structures** (high-probability).
- $\mathbb{E}_{\boldsymbol{z}}[\log R_{zx}(\boldsymbol{z})]$ is equal to the entropic contribution to the free energy, that tries to **maximize the entropy** of the generated distrubtuion and prevents **mode-collapse**.

Boltzmann Generators – Training by Examples

Training by Energy

- Approximate J_{KL} using a batch of around 1,000 samples.
- Gradient descent to decrease J_{KL}.

Training by Examples

- Training by Energy alone is **not** sufficient as it tends to focus on sampling of the most stable states.
- First, initialize BG with some **valid** configurations $\boldsymbol{x}$. Transform (invertly) them to the latent space $\boldsymbol{z} = F_{xz}(\boldsymbol{x})$.
- Maximize the likelihood by minimizing:

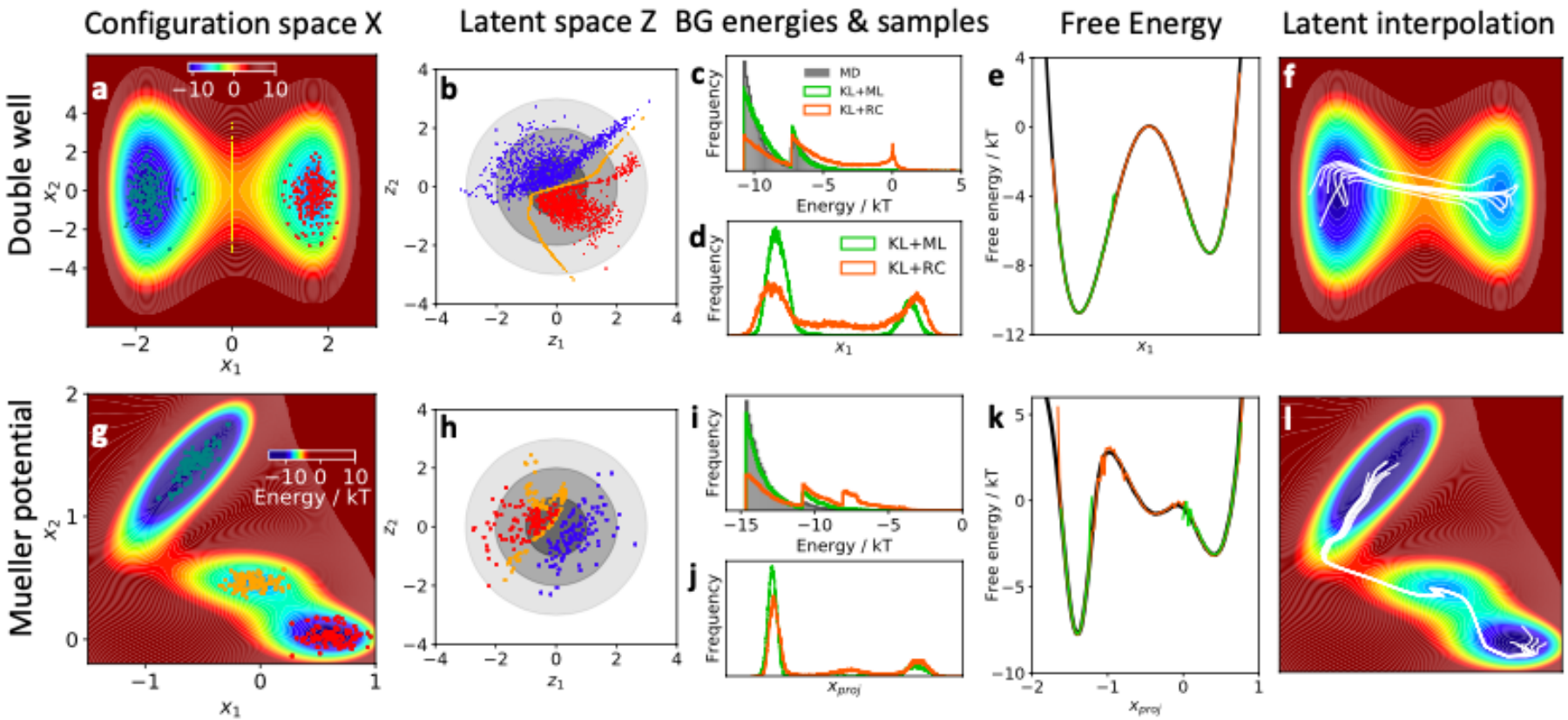
$$J_{ML} = \mathbb{E}_{\boldsymbol{x}}\left[\frac{1}{2}||F_{xz}(\boldsymbol{x})||^2 - \log R_{xz}(\boldsymbol{x})\right]$$

to help F_{zx} to focus on relevant parts of state space.

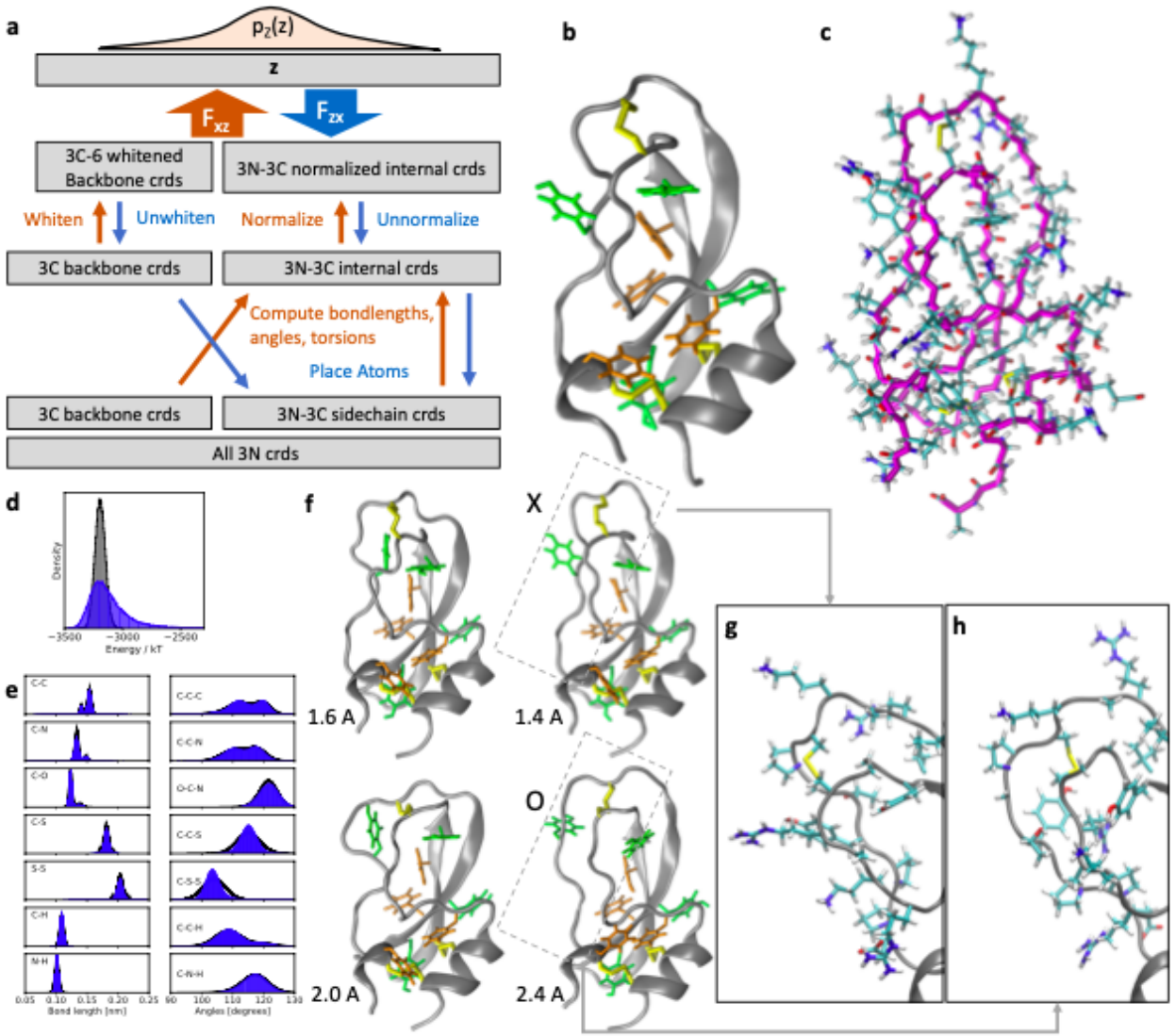
Experiment – 2D model potentials (toy model)

Experiments:

1. 2D toy model
2. Condensed matter systems
3. Complex molecules (proteins)



Experiment – Complex molecules (proteins)



Paper 3

Conformation-Guided Molecular Representation with Hamiltonian Neural Network (ICLR 2021)

Ziyao Li, Shuwen Yang, Guojie Song, Lingsheng Cai

`https://openreview.net/pdf?id=q-cnWaaoUTH`

Note: It is kind of weird when people apply the notion of Hamiltonian in Newtonian physics to a quantum physics problem. However, it is just my opinion.

Proposal

Proposal

A novel molecular representation algorithm which **preserves 3D conformations** of molecules with a **Molecular Hamiltonian Network** (HamNet).

HamNet

Implicit positions and momentums of atoms in a molecule interact in the **Hamiltonian Engine** following the **discretized** Hamiltonian equations.

- translation rotation-invariant losses.
- inputs to **Fingerprint Generator**, a MPNN for other downstream molecular tasks.

Hamiltonian equations

The Hamiltonian equations depict Newton's laws of motion in the form of first-order PDEs. Considering a system of n particles with positions $(\boldsymbol{q}_1, .., \boldsymbol{q}_n)$ and momentums $(\boldsymbol{p}_1, .., \boldsymbol{p}_n)$, the dynamics of the system follow:

$$\dot{\boldsymbol{q}}_i = \frac{d\boldsymbol{q}_i}{dt} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{p}_i}, \qquad \dot{\boldsymbol{p}}_i = \frac{d\boldsymbol{p}_i}{dt} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{q}_i}$$

where $\mathcal{H}$ is the Hamiltonian of the system, equals to the total system energy:

$$\mathcal{H} = \sum_{i=1}^{n} \mathcal{T}_i + \mathcal{U}$$

If dissipation exists in the system, $\boldsymbol{\Phi}$ denotes the dissipation function which describes how the system enery is dissipated by the outer environment:

$$\dot{\boldsymbol{q}}_i = \frac{\partial \mathcal{H}}{\partial \boldsymbol{p}_i}, \qquad \dot{\boldsymbol{p}}_i = -\left(\frac{\partial \mathcal{H}}{\partial \boldsymbol{q}_i} + \frac{\partial \boldsymbol{\Phi}}{\partial \dot{\boldsymbol{q}}_i}\right) = -\left(\frac{\partial \mathcal{H}}{\partial \boldsymbol{q}_i} + m_i \frac{\partial \boldsymbol{\Phi}}{\partial \boldsymbol{p}_i}\right)$$

Discretized Hamiltonian equations (1)

The **Hamiltonian engine** is designed to simulate the physical interactions between atoms in a molecule. To correctly incorporate the laws of motion, we **discretize** the Hamiltonian equations, and model the energy and dissipation with **learnable** functions. At the t-th step in the engine, for atom i,

$$\boldsymbol{q}_i^{(t+1)} = \boldsymbol{q}_i^{(t)} + \eta \frac{\partial \mathcal{H}^{(t)}}{\partial \boldsymbol{p}_i^{(t)}}, \qquad \boldsymbol{p}_i^{(t+1)} = \boldsymbol{p}_i^{(t)} - \eta \left(\frac{\partial \mathcal{H}^{(t)}}{\partial \boldsymbol{q}_i^{(t)}} + m_i \frac{\partial \boldsymbol{\Phi}^{(t)}}{\partial \boldsymbol{p}_i^{(t)}} \right)$$

where:

- η is a hyper-parameter of step size which controls the granularity of the discretization.
- m_i is the (normalized) mass of atom i.
- $\mathcal{H}^{(t)}$ and $\boldsymbol{\Phi}^{(t)}$ are the learnable Hamiltonian and dissipation functions of $\boldsymbol{q}^{(t)}$ and $\boldsymbol{p}^{(t)}$.

Discretized Hamiltonian equations (2)

$$\mathcal{H} = \sum_{i=1}^{n} T_i(\boldsymbol{p}_i) + U(\boldsymbol{q}_i, .., \boldsymbol{q}_n), \qquad \Phi = \sum_{i=1}^{n} \phi(\boldsymbol{p}_i)$$

- $\boldsymbol{q}, \boldsymbol{p} \in \mathbb{R}^{d_f}$, $d_f > 3$ are **implicit** positions and momentums in a **generalized** d_f-dimensional space. $\{W_T, W_\phi, W_U\}$ are network parameters.
- Kinetic energy $T = p^2/2m$ as the quadratic forms:

$$T_i(\boldsymbol{p}_i) = \frac{\boldsymbol{p}_i^T W_T^T W_T \boldsymbol{p}_i}{2m_i}$$

- Rayleigh's dissipation function $\Phi = 1/2 \sum_{i,j=1}^{n} c_{ij} \dot{\boldsymbol{q}}_i \dot{\boldsymbol{q}}_j$:

$$\phi_i(\boldsymbol{p}_i) = \frac{\boldsymbol{p}_i^T W_\phi^T W_\phi \boldsymbol{p}_i}{2m_i^2}$$

- Lennard-Jones potential $U(r) = \epsilon(r^{-12} - r^{-6})$:

$$U = \sum_{i \neq j} u_{ij}, \qquad u_{ij} = r_{ij}^{-4} - r_{ij}^{-2}, \qquad r_{ij}^2 = (\boldsymbol{q}_i - \boldsymbol{q}_j)^T W_U^T W_U (\boldsymbol{q}_i - \boldsymbol{q}_j)$$

Overall

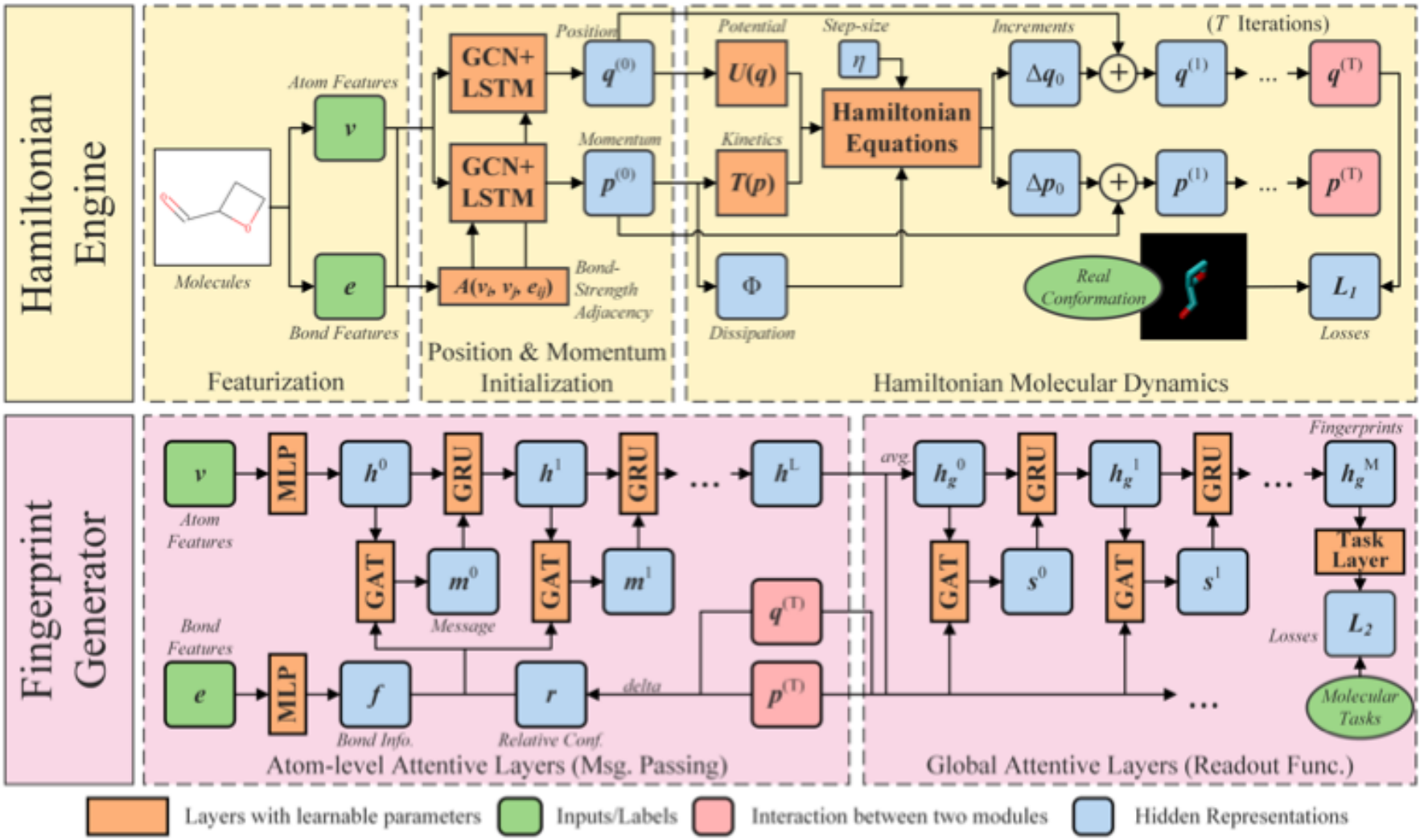


Figure 1: The overall structure of HamNet.

Initial positions and momentums

Graph-based neural networks are used to initialize these spatial quantities, given the **bond-strength-adjacency**:

$$\boldsymbol{A}_{ij} = \text{sigmoid}(\text{MLP}(\boldsymbol{v}_i \oplus \boldsymbol{e}_{ij} \oplus \boldsymbol{v}_j))$$

An LSTM over the GCN outputs to generate unique positions and momentums:

$$\tilde{\boldsymbol{q}}_i = \bigoplus_{\ell=0}^{L} \boldsymbol{f}_i^{(\ell)}, \qquad \boldsymbol{q}_i^{(0)} = \text{LSTM}_{s_i}(\tilde{\boldsymbol{q}}_{s_1}, .., \tilde{\boldsymbol{q}}_{s_n})$$

$$\tilde{\boldsymbol{p}}_i = \bigoplus_{\ell=0}^{L} \boldsymbol{g}_i^{(\ell)}, \qquad \boldsymbol{p}_i^{(0)} = \text{LSTM}_{s_i}(\tilde{\boldsymbol{p}}_{s_1}, .., \tilde{\boldsymbol{p}}_{s_n})$$

where $\boldsymbol{f}_i^{(\ell)}$, $\boldsymbol{g}_i^{(\ell)}$ are hidden representations of atom i in the ℓ-th GCN layer.

Conformation preserving

After the dynamical process in the Hamiltonian engine, positions in the generalized $\mathbb{R}^{d_f}$ space are transformed into real 3D space **linearly**:

$$\hat{\boldsymbol{Q}} = \boldsymbol{Q}\boldsymbol{W}_{\text{trans}}$$

where $\hat{\boldsymbol{Q}} \in \mathbb{R}^{n\times 3}$ and $\boldsymbol{Q} = (\boldsymbol{q}_1, .., \boldsymbol{q}_n) \in \mathbb{R}^{n\times d_f}$.

Problem

It is not rotational-equivariant.

Translational- and rotational-invariant metrics

1. Kabsch-RMSD (K-RMSD):

$$\hat{\boldsymbol{Q}}^K = \text{Kabsch}(\hat{\boldsymbol{Q}}, \boldsymbol{Q}^R), \qquad L_{\text{k-rmsd}}(\hat{\boldsymbol{Q}}, \boldsymbol{Q}^R) = \sqrt{\frac{\sum_{i=1}^n m_i ||\hat{\boldsymbol{q}}_i^K - \boldsymbol{q}_i^R||_2^2}{\sum_{i=1}^n m_i}}$$

2. Distance loss:

$$L_{\text{dist}}^2(\hat{\boldsymbol{Q}}, \boldsymbol{Q}^R) = \frac{1}{n^2} \sum_{i,j=1}^n \left(||\boldsymbol{q}_i - \boldsymbol{q}_j||_2^2 - ||\boldsymbol{q}_i^R - \boldsymbol{q}_j^R||_2^2 \right)^2$$

3. ADJ-k loss considering the distances between k-hop atoms:

$$L_{\text{adj-k}}^2(\hat{\boldsymbol{Q}}, \boldsymbol{Q}^R) = \frac{1}{n^2} \sum_{i,j=1}^n \tilde{\boldsymbol{A}}_{ij}^k \left(||\boldsymbol{q}_i - \boldsymbol{q}_j||_2^2 - ||\boldsymbol{q}_i^R - \boldsymbol{q}_j^R||_2^2 \right)^2$$

$$\mathcal{L} = L_{\text{k-rmsd}} + \lambda L_{\text{adj-3}}$$

Experiments (1)

Table 1: Quantitative results of conformation prediction on QM9.

METRIC	Kabsch-RMSD (Å)	Distance Loss (10^{-2} Å)
MPNN	1.708	8.620
RDKit	1.649	7.519
Ham. Eng. (*w/o LSTM*)	2.039	10.871
Ham. Eng. (*w/o dyn.*)	1.442	5.519
Ham. Eng. (*w/o* Φ)	1.389	5.227
Ham. Eng. (*w/o ADJ-3*)	**1.084**	7.746
Ham. Eng. (*as proposed*)	1.384	**5.186**

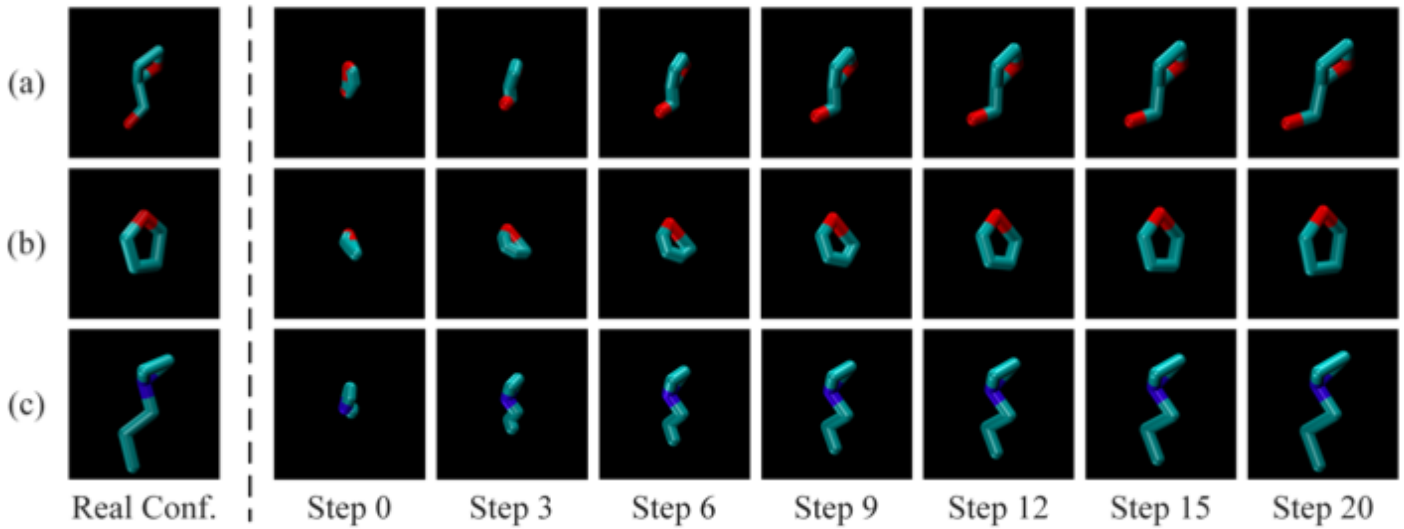
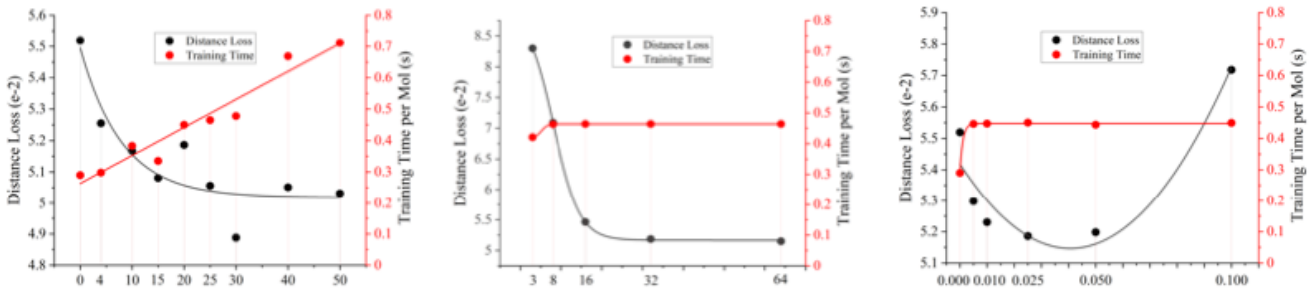


Figure 2: Visualized conformations at different steps of the Hamiltonian Engine.

Experiments (2)

Table 2: Quantitative results on various datasets of baselines, HamNet, and its variants. Baselines using 3D conformations as inputs are marked bold. "↑" indicates that higher is better, "↓" contrarily. We directly take reported performances from references, and leave unreported entries blank ("–").

DATASET METRIC	QM9 Multi-MAE↓	Tox21 Multi-ROC↑	Lipop RMSE↓	FreeSolv RMSE↓	ESOL RMSE↓
MoleculeNet (2017)	2.350	0.829	0.655	1.150	0.580
3DGCN (2019)	–	–	–	0.824±0.014	0.558±0.069
DimeNet (2020)	1.920	–	–	–	–
Attentive FP (2020)	1.292	0.857	0.578	0.736	0.505
CMPNN (2020)	–	0.856± 0.006	–	0.808±0.129	0.547±0.011
HamNet (*w/o conf.*)	1.237±0.030	0.868±0.012	0.572±0.011	0.737±0.025	0.458±0.010
HamNet (real conf.)	1.199±0.017	0.864±0.006	0.566±0.015	0.710±0.036	0.463±0.011
HamNet (*ours*)	**1.194±0.038**	**0.875± 0.006**	**0.557±0.014**	**0.694±0.053**	**0.444±0.011**



(a) Engine depth (T). (b) Dimensionality of $\boldsymbol{q}, \boldsymbol{p}$ (d_f). (c) Step size (η).

Figure 3: Effects on conformation prediction of hyperparameters in the Hamiltonian Engine. Distance losses and running time versus **(a)** the engine depth T; **(b)** the dimensionality of the generalized space d_f; and **(c)** the step size η of discretization are plotted.