

A Limitations

Coupled Atomic Motions. Atomic motions are coupled. For example, the stretch-bend coupling is a well-known effect where the stretching of a bond can influence the angle between adjacent atoms. While AniDS models anisotropic noise at the per-atom level with pairwise directional corrections, it does not explicitly capture higher-order couplings among three or more atoms. This may limit its ability to fully represent complex correlated motions, such as torsional barriers or cooperative vibrations in larger systems. Future work could explore incorporating multi-body structural priors or higher-order interaction terms into the noise generator to better reflect these coupling effects.

Approximated Force Field Modeling. Similar to prior works [17, 27], the proposed AniDS framework approximates force field learning by modeling the noise distribution. To achieve high-accuracy predictions, it should be paired with supervised force learning—either during fine-tuning or joint training. The role of AniDS is to enhance the effectiveness of supervised force field learning by providing a more informative and physically grounded initialization, or act as a regularization.

B More Details on Methodology

B.1 The Backbones of Denoising Auto-encoder and Force Encoding

EquiformerV2 [24] is an improved SE(3)/E(3)-equivariant Transformer architecture designed to scale to higher-degree representations in 3D atomistic systems. Building upon the original Equiformer, EquiformerV2 replaces computationally expensive SO(3) convolutions with efficient eSCN convolutions based on SO(2) operations, reducing the computational complexity from $\mathcal{O}(L_{\max}^6)$ to $\mathcal{O}(L_{\max}^3)$. To further leverage higher-order equivariant features, it introduces three architectural innovations: *attention re-normalization*, *separable S^2 activation*, and *separable layer normalization*. These enhancements improve model expressivity, data efficiency, and numerical stability. EquiformerV2 achieves leading performances on large-scale datasets such as OC20 and OC22 for energy and force prediction, demonstrating strong accuracy over previous equivariant GNNs.

GeoMFormer [23] is a general Transformer-based architecture for geometric molecular representation learning that simultaneously models both invariant and equivariant properties of molecular systems. It employs a dual-stream design: one stream learns invariant features (*e.g.*, energy-related properties), while the other learns equivariant features (*e.g.*, forces or atomic positions). These two streams are connected via carefully designed cross-attention modules, allowing mutual information exchange and enhancing representation quality. This modular design enables GeoMFormer to flexibly incorporate geometric constraints (*e.g.*, SE(3) invariance/equivariance) and supports strong performance across diverse tasks, including energy prediction and molecular dynamics.

Force Encoding. Following [27], in implementation, for the equivariant network of EquiformerV2 [24], we include the additional node-level features of:

$$EM_{\mathbf{f}_i} = \text{SO3}_{\text{linear}}(\|\mathbf{f}_i\| \cdot Y(\frac{\mathbf{f}_i}{\|\mathbf{f}_i\|})) \quad (22)$$

For the invariant network of GeoMFormer [23], we have additional edge-level features of:

$$EM_{\mathbf{f}_{ij}} = \text{linear}(\mathbf{f}_i \cdot \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|}) \cdot \|\mathbf{r}_{ij}\|. \quad (23)$$

B.2 Proof for the Covariance Matrix’s Properties

Proof that the covariance matrix is positive definite.

Proof. For any non-zero vector $\mathbf{v} \in \mathbb{R}^3$, we compute:

$$\mathbf{v}^\top \Sigma_i \mathbf{v} = a_i \mathbf{v}^\top \mathbf{I} \mathbf{v} - a_i \sum_j \gamma_{ij} \mathbf{v}^\top \left(\frac{\mathbf{r}_{ij} \mathbf{r}_{ij}^\top}{|\mathbf{r}_{ij}|^2} \right) \mathbf{v}.$$

Step 1: Simplify Each Term.

1. Isotropic Term:

$$a_i \mathbf{v}^\top \mathbf{I} \mathbf{v} = a_i \|\mathbf{v}\|^2.$$

2. Anisotropic Term:

$$\mathbf{v}^\top \left(\frac{\mathbf{r}_{ij} \mathbf{r}_{ij}^\top}{|\mathbf{r}_{ij}|^2} \right) \mathbf{v} = \left(\frac{\mathbf{v}^\top \mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} \right)^2 = \left(\mathbf{v}^\top \frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} \right)^2 = (\mathbf{v} \cdot \mathbf{u}_{ij})^2,$$

where $\mathbf{u}_{ij} = \mathbf{r}_{ij}/|\mathbf{r}_{ij}|$ is a unit vector.

Step 2: Combine Terms. Substitute back into $\mathbf{v}^\top \Sigma_i \mathbf{v}$:

$$\mathbf{v}^\top \Sigma_i \mathbf{v} = a_i \|\mathbf{v}\|^2 - a_i \sum_j \gamma_{ij} (\mathbf{v} \cdot \mathbf{u}_{ij})^2.$$

Factor out $a_i > 0$:

$$= a_i \left(\|\mathbf{v}\|^2 - \sum_j \gamma_{ij} (\mathbf{v} \cdot \mathbf{u}_{ij})^2 \right).$$

Step 3: Bounding the Anisotropic Correction. The term $\sum_j \gamma_{ij} (\mathbf{v} \cdot \mathbf{u}_{ij})^2$ represents the weighted sum of squared projections of \mathbf{v} onto the directions \mathbf{u}_{ij} . By the Cauchy-Schwarz inequality:

$$(\mathbf{v} \cdot \mathbf{u}_{ij})^2 \leq \|\mathbf{v}\|^2 \|\mathbf{u}_{ij}\|^2 = \|\mathbf{v}\|^2.$$

Thus:

$$\sum_j \gamma_{ij} (\mathbf{v} \cdot \mathbf{u}_{ij})^2 \leq \sum_j \gamma_{ij} \|\mathbf{v}\|^2 = \|\mathbf{v}\|^2 \sum_j \gamma_{ij}.$$

Step 4: Ensuring Positivity. From Equation (9), the normalization ensures:

$$\sum_j \gamma_{ij} = \frac{\sum_j \exp(b_{ij})}{\sum_j \exp(b_{ij}) + c_i} < 1,$$

since $c_i = \exp(\omega_2(\mathbf{h}_i)) > 0$. Therefore:

$$\|\mathbf{v}\|^2 - \sum_j \gamma_{ij} (\mathbf{v} \cdot \mathbf{u}_{ij})^2 \geq \|\mathbf{v}\|^2 - \|\mathbf{v}\|^2 \sum_j \gamma_{ij} = \|\mathbf{v}\|^2 (1 - \sum_j \gamma_{ij}) > 0.$$

Step 5: Final Result. Since $a_i > 0$ and the bracketed term is positive:

$$\mathbf{v}^\top \Sigma_i \mathbf{v} = a_i \cdot (\text{positive term}) > 0.$$

Thus, Σ_i is positive definite. □

B.3 Detailed Method for Applying AniDS with Partially Corrupted Material Structures

Specifically, for a material \mathbf{M} , we first stochastically decide whether to apply noise perturbation with probability p_{DeNS} . If applied, we perturb only a subset of atoms with the fraction r_{DeNS} . This separates the material into two subsets: the *perturbed atoms* \mathbf{M}_{pert} and the *unperturbed atoms* $\mathbf{M}_{\text{unpert}}$. For the perturbed atoms, the atomic-level forces $\mathbf{F}_{\text{pert}} \in \mathbb{R}^{N_{\text{pert}} \times 3}$ are explicitly encoded as model inputs, and they are optimized using the AniDS loss. For the unperturbed atoms, we do not use atomic-level forces $\mathbf{F}_{\text{unpert}} \in \mathbb{R}^{(N-N_{\text{pert}}) \times 3}$ as input, but use it to train the model. This design ensures the model learns to infer forces on unperturbed regions from contextual information, while leveraging explicit force signals from perturbed regions to guide denoising.

The denoising module ψ consumes the partially perturbed structure $\tilde{\mathbf{M}}$ and produces several outputs: (1) atom-wise noise predictions $\omega_{\varepsilon} \circ \psi(\tilde{\mathbf{M}}, \mathbf{F}(\mathbf{M}))$, (2) a global energy prediction $\omega_{\hat{E}} \circ \psi(\tilde{\mathbf{M}}, \mathbf{F}(\mathbf{M}))$, (3) atom-wise force predictions $\omega_{\hat{f}} \circ \psi(\tilde{\mathbf{M}}, \mathbf{F}(\mathbf{M})) = \{\hat{f}_i \in \mathbb{R}^3 \mid i \in \mathbf{M}\}$, and (4) a stress tensor

prediction $\omega_{\hat{\rho}} \circ \psi(\tilde{\mathbf{M}}, \mathbf{F}(\mathbf{M}))$. $\omega_{\hat{\epsilon}}$, $\omega_{\hat{f}}$, and $\omega_{\hat{E}}$ are prediction heads, depending on the used backbone molecular denoising autoencoder. The loss functions for these properties are defined as follows:

$$\mathcal{L}_E = \mathbb{E}_{q_{\Sigma}(\tilde{\mathbf{X}}, \mathbf{X})} \left\| E(\mathbf{M}) - \hat{E}(\tilde{\mathbf{M}}, \mathbf{F}(\mathbf{M})) \right\|, \quad (24)$$

$$\mathcal{L}_F = \frac{1}{|\mathbf{M}_{\text{unpert}}|} \mathbb{E}_{q_{\Sigma}(\tilde{\mathbf{X}}, \mathbf{X})} \sum_{i \in \mathbf{M}_{\text{unpert}}} \left\| f_i(\mathbf{M}) - \hat{f}_i(\tilde{\mathbf{M}}, \mathbf{F}(\mathbf{M})) \right\|^2, \quad (25)$$

$$\mathcal{L}_{\rho} = \mathbb{E}_{q_{\Sigma}(\tilde{\mathbf{X}}, \mathbf{X})} \left\| \rho(\mathbf{M}) - \hat{\rho}(\tilde{\mathbf{M}}, \mathbf{F}(\mathbf{M})) \right\|, \quad (26)$$

The overall loss of this formulation is:

$$\mathcal{L} = \lambda_E \mathcal{L}_E + \lambda_F \mathcal{L}_F + \lambda_{\rho} \mathcal{L}_{\rho} + \lambda_{\text{AniDS}} \mathcal{L}_{\text{AniDS}} + \lambda_{\text{KL}} \mathcal{L}_{\text{KL}} + \lambda_{\gamma} \mathcal{L}_{\gamma}, \quad (27)$$

where the AniDS and KL loss are only computed for the perturbed atoms.

B.4 Proof of that DenoiseVAE and DeNS are Special Cases of AniDS

Proof. DenoiseVAE as a Special Case of AniDS.

To demonstrate that DenoiseVAE is a special case of the AniDS framework, we show that AniDS reduces to DenoiseVAE under the following constraints:

Step 1. Restriction to Diagonal Covariance Matrices. In AniDS, the noise generator produces a full covariance matrix $\Sigma_i \in \mathbb{R}^{3 \times 3}$ for each atom i . DenoiseVAE, however, assumes isotropic noise with diagonal covariance:

$$\Sigma_i = \sigma_i^2 \mathbf{I},$$

where $\sigma_i \in \mathbb{R}^+$ is a scalar variance. This reduces the anisotropic noise in AniDS to isotropic noise, ignoring directional correlations.

Step 2. Alignment of Denoising Objectives. The denoising loss in AniDS (Equation (9)) becomes equivalent to DenoiseVAE’s loss under diagonal covariance:

$$\mathcal{L}_{\text{AniDS}} = \mathbb{E} \sum_{i=1}^N \left\| \phi(\tilde{\mathbf{M}})_i - \Sigma_i^{-1} (\tilde{\mathbf{X}}_i - \mathbf{X}_i) \right\|^2.$$

For $\Sigma_i = \sigma_i^2 \mathbf{I}$, the inverse $\Sigma_i^{-1} = \frac{1}{\sigma_i^2} \mathbf{I}$, leading to:

$$\mathcal{L}_{\text{AniDS}} = \mathbb{E} \sum_{i=1}^N \sigma_i^2 \left\| \phi(\tilde{\mathbf{M}})_i - \frac{\tilde{\mathbf{X}}_i - \mathbf{X}_i}{\sigma_i^2} \right\|^2,$$

which matches DenoiseVAE’s loss $\mathcal{L}_{\text{Denoise}}$ (Equation (3) in the paper).

Step 3. Noise Generation and Training Paradigm. AniDS: Jointly trains a noise generator (for Σ_i) and a denoising autoencoder ϕ . DenoiseVAE: Jointly trains a noise generator (for σ_i^2) and a denoising module \mathcal{D}_{θ} , following a VAE framework. When Σ_i is diagonal, AniDS’s noise generator simplifies to DenoiseVAE’s variance predictor, and both use the same variational training strategy.

Conclusion. By constraining AniDS’s covariance matrices Σ_i to be isotropic ($\Sigma_i = \sigma_i^2 \mathbf{I}$), the framework exactly recovers DenoiseVAE. This makes DenoiseVAE a diagonal-covariance special case of the more general AniDS framework, which supports full anisotropic noise distributions. \square

DeNS [27] can also be viewed as a special case of AniDS. Specifically, it corresponds to the setting where the noise covariance Σ_i is fixed and shared across all atoms, *i.e.*, $\Sigma_i = \sigma^2 \mathbf{I}$ for a constant scalar σ . Unlike DenoiseVAE, which learns σ_i per atom, DeNS applies a uniform and non-trainable isotropic noise. The derivation follows the same structure as above and can be obtained by applying this additional constraint to the AniDS framework.

Table 4: MAE comparison between models trained on MPTrj and tested on the ALEXANDRIA set.

Model	Params	Energy (eV)	Force (eV/Å)	Stress (eV/Å ³)
DPA3-v2-MPTrj [64]	4.92M	0.2738	0.0207	0.0035
MACE-MP-0 [65]	4.69M	0.4939	0.0256	0.0046
EquiformerV2 + DeNS [66]	31M	0.2729	0.0134	0.0018
GeoMFormer + AniDS	3M	0.3030	0.0211	—
EquiformerV2 + AniDS (ours)	34M	0.2679 ^{+1.83%}	0.0124 ^{+7.5%}	0.0016 ^{+11.1%}

C More Experimental Results

C.1 Results on MPTrj

Following a similar experimental setup as in Section 4.2, we perform supervised learning with partial corruption and auxiliary denoising on the MPTrj dataset. For evaluation, we use a part of the ALEXANDRIA dataset [63] (random 15k structures) as the test set instead of MPTrj’s own split, since prior works adopt different data partitions for MPTrj. This approach helps avoid potential data leakage across models and ensures a fair comparison.

As shown in Table 4, our proposed AniDS framework outperforms baselines on all of three tasks. These results highlight the effectiveness of AniDS in improving force field modeling across diverse evaluation settings.

C.2 More Experimental Settings

C.2.1 Pre-training and Fine-tuning

Pre-training setup. We trained the Equiformer-V2 model on PCQM4Mv2. Our code implementation is based on the repository provided by DeNS [27]. We provide comprehensive training parameters in Table 5. The model was trained on 4 NVIDIA A100 GPUs(40G) and required 32 GPU-hours. The pseudocode for pre-training can be found in Algorithm 1.

Table 5: Hyper-parameters for PCQM4Mv2 pretrain.

Hyper-parameters	Value or description
Optimizer	AdamW
Learning rate scheduling	Cosine
Warmup steps	4000
Maximum learning rate	4×10^{-4}
Batch size (per GPU)	40
Number of steps	40000
Weight decay	0.0
Noise Generator layer	2
Prior distribution σ_p	0.1
KL loss coefficient λ_{kl}	1.0
Regularizer weight λ_γ	1.0

Finetune setup For fine-tuning on MD17, we utilized the code and hyperparameters provided by DeNS [27], specifically as detailed in Table 8. Each task was trained on a single NVIDIA V100 GPU (32GB), with the average running speed per task being similar to that of DeNS. The fine-tuning training procedure is similar to "Supervised Learning with Partial Corruption and Auxiliary Denoising," with the key difference being that during fine-tuning, we no longer train the noise generator (i.e., its associated parameters are frozen). The pseudocode for this training process is similar to that presented in Algorithm 2.

To further assess the robustness of AniDS under more challenging scenarios, we adopted the evaluation protocol established in prior works such as NequIP [10], MACE [65], and ICTP [67]. We conducted experiments on four subsets of the revised MD17 (rMD17) dataset [68] and one subset of the MD22 dataset [69]. The results are presented in Tables 6 and 7.

Table 6: Force MAE (meV/Å) on rMD17 datasets. Lower is better.

Model	Aspirin	Benzene	Toluene	Uracil
NequIP [10]	52.0	2.9	15.1	40.1
MACE [65]	43.9	2.7	12.1	25.9
ICTP [67]	40.19	2.45	11.24	25.97
EquiformerV2	71.40	3.92	19.28	50.45
EquiformerV2 + DeNS	45.52	2.28	11.37	34.46
EquiformerV2 + AniDS (Ours)	40.21	2.31	10.7	28.8

Table 7: Performance comparison on the AT-AT-CG-CG dataset.

Dataset	ICTP [67]	ViSNet-LSRM [70]	MACE [65]	Allegro [70]	TorchMD-Net [70]	sGDML [69]	Equiformer [70]	Equiformer + AniDS (Ours)
AT-AT-CG-CG	3.37	4.61	5.00	5.55	14.13	30.36	5.43	3.22

For each molecule in rMD17, we used 50 training samples, following the ICTP configuration. Both pretraining and fine-tuning employed the $l_{\max} = 3$ variant of EquiformerV2. The hyperparameters used for fine-tuning are summarized in Table 9.

Table 8: Hyper-parameters for fine-tuning on MD17 dataset.

Hyper-parameter	Aspirin	Benzene	Ethanol	Malonaldehyde	Naphthalene	Salicylic acid	Toluene	Uracil
Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
Learning rate scheduling	Cosine	Cosine	Cosine	Cosine	Cosine	Cosine	Cosine	Cosine
Warm epochs	10	10	10	10	10	10	10	10
Weight decay	1e-6	1e-6	1e-6	1e-6	1e-6	1e-6	1e-6	1e-6
Epochs	1500	1500	1500	1500	1500	1500	1500	1500
Learning rate	5e-4	1e-4	5e-4	5e-4	5e-4	5e-4	5e-4	5e-4
Batch size	8	8	8	8	8	8	8	8
Ema decay	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999
Energy coefficient λ_E	1	1	1	1	2	1	1	1
Force coefficient λ_F	80	80	80	100	20	80	80	20
Probability of optimizing AniDS p_{AniDS}	0.25	0.25	0.25	0.25	0.25	0.25	0.125	0.25
Denoise coefficient $\lambda_{Denoise}$	5	5	5	5	5	5	5	5
Corruption ratio r_{AniDS}	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25

Table 9: Hyper-parameters for fine-tuning on rMD17 dataset.

Hyper-parameter	Aspirin	Benzene	Toluene	Uracil
Optimizer	AdamW	AdamW	AdamW	AdamW
Learning rate scheduling	Cosine	Cosine	Cosine	Cosine
Warm epochs	10	10	10	10
Weight decay	1e-6	1e-6	1e-6	1e-6
Epochs	2000	2000	2000	2000
Learning rate	2e-4	1e-4	2e-4	2e-4
Batch size	2	2	2	2
Energy coefficient λ_E	1	1	1	1
Force coefficient λ_F	80	80	80	20
Probability of optimizing AniDS p_{AniDS}	0.25	0.25	0.125	0.25
Denoise coefficient $\lambda_{Denoise}$	5	5	5	5
Corruption ratio r_{AniDS}	0.25	0.25	0.25	0.25

C.2.2 Supervised Learning with Partial Corruption and Auxiliary Denoising

OC22 For OC22, we employed the code and hyperparameters provided by DeNS [27], with a comprehensive list of training parameters presented in Table 10 and Table 11. It is important to note that, as we utilized a pre-trained model, we initially trained only the noise generator using KL loss to ensure training stability. This separate training phase continued until the generated KL loss

consistently fell below a threshold of 2.0, after which the entire model was trained. We performed this training on 4 NVIDIA A100 GPUs (80G). We also found that incorporating AniDS as an *auxiliary denoising objective* helps the VAE learn more informative noise generation from the provided force and energy supervision. In this configuration, we set $\lambda_\gamma = 0$ to simplify optimization, but a larger value may further enhance the anisotropic regularization effect and improve noise modeling performance. The pseudocode for this training process is similar to that provided in [2]

Table 10: Hyper-parameters for OC22 dataset.

Hyper-parameters	Value or description
Optimizer	AdamW
Learning rate scheduling	Cosine learning rate with linear warmup
Warmup epochs	0.002
Maximum learning rate	4×10^{-5}
Batch size (per gpu)	2
Number of epochs	1
Weight decay	1×10^{-3}
Dropout rate	0.1
Stochastic depth	0.1
Energy coefficient λ_E	4
Force coefficient λ_F	100
Gradient clipping norm threshold	50
Model EMA decay	0.999
Probability of optimizing AniDS p_{AniDS}	0.5
Denoise coefficient $\lambda_{Denoise}$	25
KL loss coefficient λ_{KL}	5
Prior distribution σ_p	0.15
Corruption ratio r_{AniDS}	0.5
Regularizer weight λ_γ	0.0

Table 11: Model hyper-parameters for OC22 dataset.

Model Hyper-parameters	Noise Generator	Denoising Autoencoder
Cutoff radius (\AA)	12	12
Maximum number of neighbors	20	20
Number of radial bases	32	600
Dimension of hidden scalar features in radial functions d_{edge}	(0, 64)	(0, 128)
Maximum degree L_{max}	2	6
Maximum order M_{max}	2	2
Number of Transformer blocks	4	18
Embedding dimension d_{embed}	(2, 64)	(6, 128)
$f_{ij}^{(L)}$ dimension d_{attn_hidden}	(2, 32)	(6, 64)
Number of attention heads h	4	8
$f_{ij}^{(0)}$ dimension d_{attn_alpha}	(0, 32)	(0, 64)
Value dimension d_{attn_value}	(2, 8)	(6, 16)
Hidden dimension in feed forward networks d_{ffn}	(2, 64)	(6, 128)
Resolution of point samples R	14	18

MPtrj On MPtrj, we employed two distinct models to validate the effectiveness of AniDS. First, GeoMFormer was trained from scratch on MPtrj, utilizing AniDS for auxiliary training; its parameters are detailed in Table [14] and Table [15]. Second, similar to our approach for OC22, we performed continued training on the eqV2-S-DeNS model provided in Omat24 (which was exclusively trained on MPtrj). For this model, we adopted the same KL loss regularization strategy as for OC22 to ensure training stability, with its parameters specified in Table [12] and Table [13]. Both models were trained on 8 NVIDIA A100 GPUs (40GB). The pseudocode for this training process is similar to that provided in [2]

C.3 Dynamic Simulation Tasks

We conducted a geometry optimization task using our model on the Aspirin subset of MD17. Specifically, we randomly selected 15 equilibrium structures and employed our force predictor

Table 12: Hyper-parameters for MPtrj dataset (EqV2).

Hyper-parameters	Value or description
Optimizer	AdamW
Learning rate scheduling	Cosine learning rate with linear warmup
Warmup epochs	0.1
Maximum learning rate	5×10^{-5}
Batch size (per gpu)	3
Number of epochs	7
Weight decay	1×10^{-3}
Energy coefficient λ_E	5
Force coefficient λ_F	20
stress isotropic coefficient $\lambda_{S_{iso}}$	5
stress anisotropic coefficient $\lambda_{S_{ani}}$	5
Probability of optimizing AniDS p_{AniDS}	0.5
Denoise coefficient $\lambda_{Denoise}$	10
KL loss coefficient λ_{KL}	10
Prior distribution σ_p	0.1
Corruption ratio r_{AniDS}	1.0
Regularizer weight λ_γ	0.0

Table 13: Model hyper-parameters for MPtrj dataset (EqV2).

Model Hyper-parameters	Noise Generator	Denoising Autoencoder
Cutoff radius (\AA)	12	12
Maximum number of neighbors	20	20
Dimension of hidden scalar features in radial functions d_{edge}	(0, 64)	(0, 128)
Maximum degree L_{max}	4	4
Maximum order M_{max}	2	2
Number of Transformer blocks	2	8
Embedding dimension d_{embed}	(4, 32)	(4, 128)
$f_{ij}^{(L)}$ dimension d_{attn_hidden}	(4, 32)	(4, 64)
Number of attention heads h	4	8
$f_{ij}^{(0)}$ dimension d_{attn_alpha}	(0, 32)	(0, 64)
Value dimension d_{attn_value}	(4, 8)	(4, 16)
Hidden dimension in feed forward networks d_{ffn}	(4, 64)	(4, 128)
Resolution of point samples R	18	18

Table 14: Hyper-parameters for MPtrj dataset (GeoMFormer).

Hyper-parameters	Value or description
Optimizer	Adam
Learning rate scheduling	linear
Warmup epochs	1
Maximum learning rate	2×10^{-4}
Batch size (per gpu)	40
Number of epochs	50
Energy coefficient λ_E	1
Force coefficient λ_F	1
Probability of optimizing AniDS p_{AniDS}	0.25
Denoise coefficient $\lambda_{Denoise}$	0.8
KL loss coefficient λ_{KL}	1e-3
Prior distribution σ_p	0.05
Corruption ratio r_{AniDS}	0.25
Regularizer weight λ_γ	0.0

(EquiformerV2 + AniDS) to optimize their geometries. The optimized configurations were then

Table 15: Model hyper-parameters for MPtrj dataset (GeoMFormer).

Model Hyper-parameters	Noise Generator	Denoising Autoencoder
Cutoff radius (\AA)	5	5
Maximum number of expanded atoms	256	256
Number of radial bases	10	10
Number of Transformer layers	2	6
Embedding dimension d_{embed}	128	128
Number of attention heads h	8	8
Hidden dimension in feed forward networks d_{ffn}	128	128

compared against reference structures obtained via DFT geometry optimization, and we evaluated the RMSD between corresponding atomic positions.

Because the original MD17 dataset used the FHI-aims package with the PBE-vdW-TS functional under tight settings—which is subject to licensing restrictions—we performed our DFT-based geometry optimizations using PySCF at the PBE-D3BJ/def2-SVP level. This setup provides comparable accuracy in molecular geometries while remaining fully open-source.

All models were fine-tuned on the same training samples and evaluated on the same 15 randomly selected equilibrium structures from the Aspirin subset. Table 16 summarizes the RMSD improvements (i.e., reductions in atomic positional deviations before vs. after optimization) under three configurations.

As shown in the table, all three models achieve comparable RMSD improvements, with AniDS exhibiting the largest average reduction. While the absolute differences are modest, partly due to the limited optimization steps used in this evaluation.

Table 16: Comparison of RMSD improvement across different denoising strategies.

Structure Index	AniDS	DeNS	No Denoising
1	0.0829	0.0788	0.0653
2	0.0650	0.0655	0.0650
3	0.2911	0.2895	0.2869
4	0.1883	0.1879	0.1808
5	0.1830	0.1827	0.1827
6	0.0668	0.0664	0.0663
7	0.1083	0.1075	0.1083
8	0.0925	0.0926	0.0922
9	0.2478	0.2445	0.2589
10	0.0741	0.0821	0.0819
11	0.0887	0.0886	0.0893
12	0.0637	0.0637	0.0673
13	0.1197	0.1167	0.1185
14	0.1441	0.1438	0.1438
15	0.0876	0.0868	0.0862
Average	0.1269	0.1265	0.1262

C.4 Dataset License

The datasets used in our experiments are released under the following licenses:

- **PCQM4Mv2**: Creative Commons Attribution 4.0 International License (CC BY 4.0)
- **MD17**: MIT License
- **MPtrj**: MIT License
- **OC22**: Creative Commons Attribution 4.0 International License (CC BY 4.0)
- **ALEXANDRIA**: Creative Commons Attribution 4.0 International License (CC BY 4.0)

D Pseudo Code

Algorithm 1 Pretrain with AniDS

Require: $\lambda_{\text{AniDS}}, \lambda_{\text{KL}}, \lambda_{\gamma}, \kappa, \sigma_p$
Require: Noise Decoder GNN, Noise Generator GNG

```

1: while training do
2:    $L_{\text{total}} \leftarrow 0$ 
3:   Sample a batch of  $B$  structures  $\{S^{(j)}\}_{j=1}^B$ 
4:   for  $j = 1$  to  $B$  do ▷ This for loop can be parallelized
5:      $S^{(j)} = \{(z_i, p_i)\}_{i=1}^{N_j}, L_{\text{KL}} \leftarrow 0, L_{\gamma} \leftarrow 0$ 
6:     for  $i = 1$  to  $N_j$  do ▷ This for loop can be parallelized
7:        $(\Sigma_i, \Gamma_i) \leftarrow \text{GNG}(S^{(j)})$  ▷  $\Gamma_i = \sum_k \gamma_{ik}$  (anisotropic mass)
8:        $L_i \leftarrow \text{Cholesky}(\Sigma_i)$ 
9:       Sample  $\epsilon_i \sim \mathcal{N}(0, \mathbf{I}_3), \tilde{p}_i \leftarrow p_i + L_i \epsilon_i$ 
10:       $L_{\text{KL}} \leftarrow L_{\text{KL}} + \text{KL}(\sigma_p \mathbf{I}, \Sigma_i)$ 
11:       $L_{\gamma} \leftarrow L_{\gamma} + [\max(0, \kappa - \Gamma_i)]^2$ 
12:    end for
13:     $\tilde{S}^{(j)} \leftarrow \{(z_i, \tilde{p}_i)\}_{i=1}^{N_j}$ 
14:     $\hat{\epsilon} \leftarrow \text{GNN}(\tilde{S}^{(j)})$ 
15:     $L_{\text{AniDS}} \leftarrow \frac{1}{N_j} \sum_{i=1}^{N_j} \|L_i^{-T} \epsilon_i - \hat{\epsilon}_i\|_2^2$  ▷ Equivalent to Eq. 9
16:     $L_{\text{KL}} \leftarrow \frac{1}{N_j} L_{\text{KL}}, L_{\gamma} \leftarrow \frac{1}{N_j} L_{\gamma}$ 
17:     $L_{\text{total}} \leftarrow L_{\text{total}} + \lambda_{\text{AniDS}} L_{\text{AniDS}} + \lambda_{\text{KL}} L_{\text{KL}} + \lambda_{\gamma} L_{\gamma}$ 
18:  end for
19:   $L_{\text{total}} \leftarrow \frac{L_{\text{total}}}{B}$ 
20:  Update GNN using  $L_{\text{total}}$ 
21: end while

```

E Computational Cost and Training Efficiency

To quantify the additional computational overhead introduced by AniDS, we compare training efficiency and model complexity across different noise-modeling choices on rMD17 (Aspirin, 50 training structures). Results are summarized in Table 17.

Table 17: Model size and per-epoch training time on rMD17 (Aspirin, 50 train) on single NVIDIA H100 GPU.

Method	#Parameters	Training Time per Epoch (s)
EqV2 (No Noise)	6.35M	6.2
EqV2 + DeNS	6.35M	6.9
EqV2 + DenoiseVAE	7.20M	7.3
EqV2 + AniDS (Ours)	7.36M	7.3

As shown, AniDS incurs a moderate increase in model size and training time, primarily due to its more expressive, structure-aware noise generator that outputs full covariance matrices. This overhead is justified by the significant performance gains reported in our main results (lowest MAE among all baselines).

Algorithm 2 Training with AniDS

Require: $p_{\text{AniDS}}, \lambda_{\text{AniDS}}, r_{\text{AniDS}}, \lambda_E, \lambda_F, \lambda_{\text{KL}}, \lambda_\gamma, \kappa, \sigma_p$

Require: Noise Decoder GNN, Noise Generator GNG

```

1: while training do
2:    $L_{\text{total}} \leftarrow 0$ 
3:   Sample a batch of  $B$  structures  $\{S_{\text{non-eq}}^{(j)}\}_{j=1}^B$ 
4:   for  $j = 1$  to  $B$  do ▷ This for loop can be parallelized
5:      $S_{\text{non-eq}}^{(j)} = \{(z_i, p_i, f_i)\}_{i=1}^{N_j}$ 
6:      $L_{\text{KL}} \leftarrow 0, \quad L_\gamma \leftarrow 0$ 
7:     Sample  $u \sim \mathcal{U}(0, 1)$ 
8:     if  $u < p_{\text{AniDS}}$  then ▷ Decide whether to add noise to the structure
9:       for  $i = 1$  to  $N_j$  do ▷ Atom-wise partial corruption
10:         $(\Sigma_i, \Gamma_i) \leftarrow \text{GNG}(S_{\text{non-eq}}^{(j)})$ 
11:        Sample  $q_i \sim \mathcal{U}(0, 1); \quad m_i \leftarrow \mathbb{I}[q_i < r_{\text{AniDS}}]$ 
12:        if  $m_i = 1$  then
13:           $L_i \leftarrow \text{Cholesky}(\Sigma_i); \quad \text{Sample } \epsilon_i \sim \mathcal{N}(0, \mathbf{I}_3)$ 
14:           $\tilde{p}_i \leftarrow p_i + L_i \epsilon_i$ 
15:        else
16:           $\tilde{p}_i \leftarrow p_i$ 
17:        end if
18:           $\tilde{f}_i \leftarrow f_i \cdot m_i$  ▷ Force encoding only for corrupted atoms
19:           $L_{\text{KL}} \leftarrow L_{\text{KL}} + \text{KL}(\sigma_p \mathbf{I}, \Sigma_i) \cdot m_i$ 
20:           $L_\gamma \leftarrow L_\gamma + [\max(0, \kappa - \Gamma_i)]^2 \cdot m_i$ 
21:        end for
22:         $M_j \leftarrow \max(1, \sum_{i=1}^{N_j} m_i)$ 
23:         $\tilde{S}_{\text{non-eq}}^{(j)} \leftarrow \{(z_i, \tilde{p}_i)\}_{i=1}^{N_j}, \quad \tilde{F}^{(j)} \leftarrow \{\tilde{f}_i\}_{i=1}^{N_j}$ 
24:         $(\hat{E}, \hat{F}, \hat{\epsilon}) \leftarrow \text{GNN}(\tilde{S}_{\text{non-eq}}^{(j)}, \tilde{F}^{(j)})$ 
25:         $L_E \leftarrow |E(S_{\text{non-eq}}^{(j)}) - \hat{E}|$ 
26:         $L_{\text{AniDS}} \leftarrow \frac{1}{M_j} \sum_{i=1}^{N_j} m_i \|L_i^{-T} \epsilon_i - \hat{\epsilon}_i\|_2^2$  ▷ Eq. 9 form
27:         $L_F \leftarrow \frac{1}{N_j} \sum_{i=1}^{N_j} (1 - m_i) \|f_i - \hat{f}_i\|_2^2$ 
28:         $L_{\text{KL}} \leftarrow \frac{1}{M_j} L_{\text{KL}}, \quad L_\gamma \leftarrow \frac{1}{M_j} L_\gamma$ 
29:         $L_{\text{total}} \leftarrow L_{\text{total}} + \lambda_E L_E + \lambda_{\text{AniDS}} L_{\text{AniDS}} + \lambda_F L_F + \lambda_{\text{KL}} L_{\text{KL}} + \lambda_\gamma L_\gamma$ 
30:      else
31:         $(\hat{E}, \hat{F}) \leftarrow \text{GNN}(S_{\text{non-eq}}^{(j)})$ 
32:         $L_E \leftarrow |E(S_{\text{non-eq}}^{(j)}) - \hat{E}|$ 
33:         $L_F \leftarrow \frac{1}{N_j} \sum_{i=1}^{N_j} \|f_i - \hat{f}_i\|_2^2$ 
34:         $L_{\text{total}} \leftarrow L_{\text{total}} + \lambda_E L_E + \lambda_F L_F$ 
35:      end if
36:    end for
37:     $L_{\text{total}} \leftarrow \frac{L_{\text{total}}}{B}$ 
38:    Update GNN using  $L_{\text{total}}$ 
39:  end while

```
