MMML: A Complete Machine Learning Pipeline for Molecular Modeling

Command-Line Tools for Production-Ready ML Potentials

MMML Development Team

Molecular ML Framework

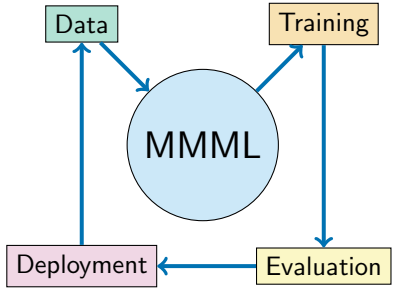
November 5, 2025

Outline

What is MMML?

Modern Molecular ML Framework

- Production-ready CLI tools
- PhysNet + DCMNet architectures
- Efficient data handling
- Comprehensive validation
- Multi-state predictions



Key Features

Data Processing

- Auto-padding removal
- Quality control
- Train/val/test splits
- Memory-mapped support

Model Training

- PhysNet (E/F/D)
- DCMNet (ESP)
- Multi-state (charge/spin)
- Auto-tuned hyperparams

Analysis Tools

- Model evaluation
- MD simulations
- Vibrational analysis
- ASE integration

Data Cleaning

Remove problematic structures from your dataset

```
# Clean dataset with quality control
python -m mmml.cli.clean_data glycol.npz \
    -o glycol_cleaned.npz \
    --max-force 10.0 \
    --min-distance 0.4 \
    --max-energy -1e-3
```

What it does:

- Removes zero/NaN/Inf energies
- Filters high forces (SCF failures)
- Checks for overlapping atoms
- Keeps only essential fields (E, F, R, Z, N, D)

Result: Clean, validated dataset ready for training

Data Exploration

Understand your dataset before training

```
# Explore dataset statistics
python -m mmml.cli.explore_data glycol_cleaned.npz \
    --plot-distributions \
    --analyze-bonds \
    --output-dir analysis/
```

Provides:

- Energy distribution
- Force statistics
- Bond length analysis
- Molecular size distribution
- Element composition
- Distance matrices
- Quality metrics
- Summary statistics

Example output: 5,782 structures, 10 atoms, C/H/O composition

Dataset Splitting

Create train/validation/test splits

```
# Split dataset (80% train, 10% val, 10% test)
python -m mmml.cli.split_dataset glycol_cleaned.npz \
    -o splits/ \
    --train-size 0.8 \
    --valid-size 0.1 \
    --test-size 0.1 \
    --seed 42
```

Features:

- Reproducible splits (seed-based)
- Filters out non-per-structure fields
- Saves split indices for reproducibility
- Stratified splitting available

Output: `data_train.npz`, `data_valid.npz`, `data_test.npz`

Automatic Padding Removal

NEW: Training now auto-detects and removes padding!

```
# Before: Dataset padded to 60 atoms (10 real + 50 padding)
# After: Automatically detected and removed during training

python -m mmml.cli.make_training \
    --data splits/data_train.npz \
    --ckpt_dir checkpoints/glycol
```

Console output:

```
Auto-detecting number of atoms from dataset...
     Actual molecule size: 10 atoms (from max(N))
        Data is PADDED: 60 atoms (padding: 50)
      Auto-removing padding to train efficiently...
     Saved unpadded data to: data_train_unpadded.npz
```

Result: 6x faster training! (10 vs 60 atoms)

Basic Training

Train a PhysNet model for energies, forces, and dipoles

```
# Simple training with auto-detection
python -m mmml.cli.make_training \
    --data splits/data_train.npz \
    --ckpt_dir checkpoints/glycol_run1 \
    --n_train 4000 \
    --n_valid 500 \
    --num_epochs 100 \
    --batch_size 16
```

Auto-detected:

- Number of atoms (from dataset)
- Padding removal (if needed)
- Checkpoint path (made absolute)

Training Configuration

Customize your training with many options

```
python -m mmml.cli.make_training \
    --data splits/data_train.npz \
    --ckpt_dir checkpoints/glycol_production \
    --n_train 4000 --n_valid 500 \
    --num_epochs 100 --batch_size 16 \
    --learning_rate 0.001 --features 128 \
    --num_iterations 3 --cutoff 10.0 \
    --energy_weight 1.0 --forces_weight 1.0 \
    --dipole_weight 1.0
```

Model architecture:

- `--features`: Hidden size
- `--num_iterations`: MP steps
- `--cutoff`: Interaction range
- `--num_basis_functions`: RBF

Loss weights:

- `--energy_weight`
- `--forces_weight`
- `--dipole_weight`
- `--charges_weight`

Joint PhysNet+DCMNet Training

Advanced: Train for ESP prediction

```
# Train joint model for electrostatic potential
python -m mmml.cli.train_joint \
    --train-efd train_efd.npz \
    --train-esp train_esp.npz \
    --valid-efd valid_efd.npz \
    --valid-esp valid_esp.npz \
    --epochs 100 --batch-size 4 \
    --optimizer adamw --use-recommended-hparams
```

Architecture:

Features:

- Multiple optimizers (Adam, AdamW, RMSprop, Muon)
- Auto-tuned hyperparameters
- Comprehensive ESP validation plots

Memory-Mapped Training

Train on large datasets (>100k structures)

```
# Train on memory-mapped data (no RAM limits)
python -m mmml.cli.train_memmap \
    --data_path openqdc_packed_memmap \
    --batch_size 32 \
    --num_epochs 100 \
    --num_atoms 60 \
    --bucket_size 8192
```

Benefits:

- No RAM limits (memory-mapped)
- Bucketed batching (minimizes padding)
- HPC-friendly (efficient I/O)
- Scales to millions of structures

Compatible with: OpenQDC, custom packed formats

Multi-State Training

Train models for different charge and spin states

```
# Train charge/spin conditioned model
python -m mmml.cli.train_charge_spin \
    --data_path openqdc_packed_memmap \
    --batch_size 32 \
    --charge_min -2 --charge_max 2 \
    --spin_min 1 --spin_max 5
```

Applications:

- Ionic species (charge states)
- Excited states (spin states)
- Redox reactions
- Photochemistry

Method: Learned embeddings for charge and spin

Model Inspection

Inspect trained models and checkpoints

```
python -m mmml.cli.inspect_checkpoint \
    checkpoints/glycol_run1/ \
    --show-structure \
    --count-parameters
```

Provides:

- Total parameter count
- Layer-by-layer breakdown
- Model configuration
- Parameter statistics

Parameter Breakdown:

Component	Parameters
Embedding	3,584
Atomic embeddings	2,560
Edge embeddings	1,024
Interaction Blocks	49,152
MessagePass 0	12,288
MessagePass 1	12,288
MessagePass 2	12,288
Residual MLPs	12,288
Output Layers	8,192
Energy head	4,096
Forces head	2,048
Dipole head	2,048
Total	**60,928**

Configuration:

- Features: 128
- Iterations: 3

Model Evaluation

Comprehensive evaluation on test set

```
# Evaluate on test set
python -m mmml.cli.evaluate_model \
    checkpoints/glycol_run1/params*.pkl \
    --test-data splits/data_test.npz \
    --output-dir evaluation/ \
    --plot-results
```

Generates:

- Energy predictions
- Force predictions
- Dipole predictions
- Error distributions
- Scatter plots
- Residual analysis
- Per-property MAE/RMSE
- Statistical summaries

Training History

Visualize training progress

```
# Plot training history
python -m mmml.cli.plot_training \
    checkpoints/glycol_run1/ \
    --output training_curves.png
```

Shows:

- Train/validation loss curves
- Energy MAE progression
- Forces MAE progression
- Learning rate schedule
- Convergence indicators

Tip: Use to diagnose overfitting or underfitting

ML/MM Hybrid Simulations

Combine machine learning with classical force fields

```
# Run hybrid ML/MM simulation
python -m mmml.cli.run_sim \
    --pdbfile init.pdb \
    --checkpoint checkpoints/model/params.pkl \
    --cell 32.0 \
    --n-monomers 100 \
    --n-atoms-monomer 10 \
    --ml-cutoff 1.5 \
    --mm-switch-on 5.0 \
    --mm-cutoff 3.0 \
    --include-mm \
    --temperature 300 \
    --timestep 0.5 \
    --nsteps 50000
```

Use cases:

- Large systems (ML for QM region, MM for bulk)
- Long-range electrostatics
- Explicit solvation
- Periodic boundary conditions

Cutoff Optimization

Optimize ML/MM cutoff parameters for best accuracy

```
# Grid search for optimal cutoffs
python -m mmml.cli.opt_mmml \
    --dataset dimers.npz \
    --checkpoint checkpoints/model/params.pkl \
    --n-monomers 2 \
    --n-atoms-monomer 10 \
    --ml-cutoff-grid 0.5,1.0,1.5,2.0 \
    --mm-switch-on-grid 4.0,5.0,6.0 \
    --mm-cutoff-grid 1.0,2.0,3.0 \
    --include-mm \
    --out cutoff_opt.json
```

Optimizes:

- ML cutoff distance
- MM switch-on distance
- MM cutoff width
- Energy accuracy
- Force accuracy
- Computational cost

Output: Best parameters for energy/force trade-off

Diffusion Monte Carlo (DMC)

Quantum Monte Carlo simulations with ML potentials

```
# Run DMC simulation
python -m mmml.dmc.dmc \
    --natm 20 \
    --nwalker 512 \
    --stepsize 5.0e-4 \
    --nstep 5000 \
    --eqstep 1000 \
    --checkpoint checkpoints/model/params.pkl \
    --max-batch 512 \
    -i molecule.xyz
```

Features:

- Quantum nuclear effects
- Zero-point energy corrections
- Ground state wavefunctions
- High-accuracy energetics

Applications: Hydrogen bonding, tunneling, isotope effects

HPC Deployment

Deploy on computing clusters with SLURM

```
#!/bin/bash
#SBATCH --job-name=mmml_md
#SBATCH --nodes=1
#SBATCH --ntasks=8
#SBATCH --mem-per-cpu=3000
#SBATCH --partition=gpu
#SBATCH --gres=gpu:1

# Load environment
source ~/mmml/.venv/bin/activate
module load gcc cuda

# Run simulation
python -m mmml.cli.run_sim \
    --checkpoint checkpoints/model/params.pkl \
    --pdbfile system.pdb \
    --temperature 300 \
    --nsteps 1000000
```

Supports: SLURM, PBS, multi-GPU, distributed training

Periodic Systems

Simulations with periodic boundary conditions

Setup:

```
# Create periodic box
python make_box.py \
    --n 100 \
    --side_length 32.0

# Run PBC simulation
python -m mmml.cli.run_sim \
    --pdbfile box.pdb \
    --cell 32.0 \
    --ensemble nvt
```

Applications:

- Liquids
- Crystals
- Interfaces
- Solvation
- Bulk properties

Supports: Cubic, orthorhombic, triclinic cells

ASE Calculator Interface

Use trained models as ASE calculators

```python
# Python script using ASE calculator
from mmml.cli.calculator import MMMLCalculator
from ase import Atoms

# Load model
calc = MMMLCalculator(
    checkpoint_path="checkpoints/glycol_run1/params*.pkl"
)

# Create molecule
atoms = Atoms('H2O', positions=[[0,0,0], [1,0,0], [0,1,0]])
atoms.calc = calc

# Get predictions
energy = atoms.get_potential_energy() # eV
forces = atoms.get_forces()         # eV/
```

Compatible with: All ASE tools (optimization, MD, NEB, etc.)

Molecular Dynamics

Run MD simulations with trained models

```
# Run MD simulation
python -m mmml.cli.dynamics \
    --checkpoint checkpoints/glycol_run1/params*.pkl \
    --input molecule.xyz \
    --temp 300 \
    --steps 10000 \
    --timestep 0.5 \
    --output trajectory.npz
```

Simulation types:

- NVE (constant energy)
- NVT (constant temperature, Langevin)
- NPT (constant pressure, Berendsen)
- Energy minimization

Output: NPZ trajectory compatible with ASE visualization

Vibrational Analysis

Compute normal modes and IR spectra

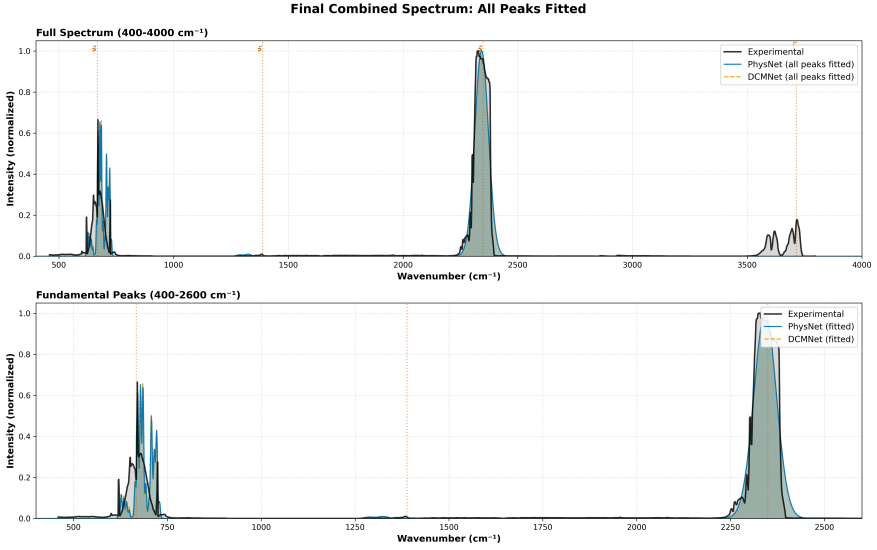
```
# Compute vibrational frequencies
python -m mmml.cli.dynamics \
    --checkpoint checkpoints/glycol_run1/params*.pkl \
    --input optimized.xyz \
    --mode vibrations \
    --output-modes modes.pkl
```

Provides:

- Vibrational frequencies (cm^{-1})
- Normal mode vectors
- IR intensities (if dipoles available)
- Zero-point energy
- Thermodynamic properties

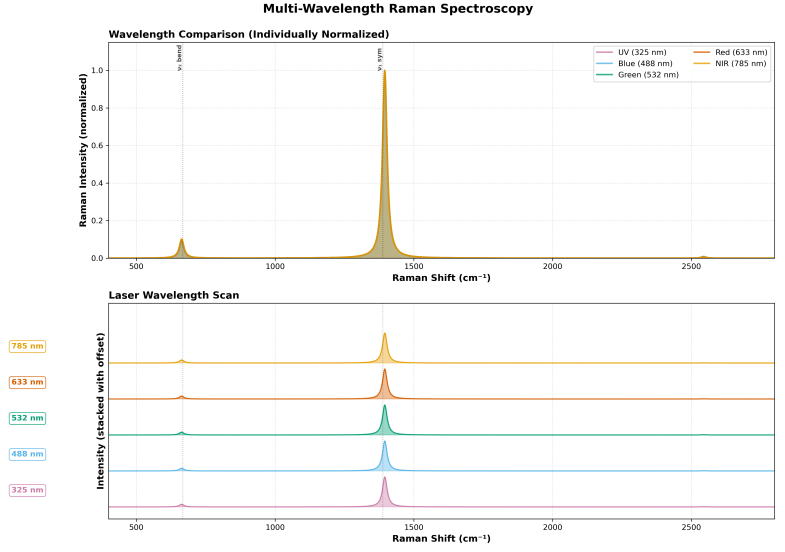
Visualization: Export to ASE trajectory for viewing

CO2: Combined IR & Raman Spectrum



Computed spectra: IR absorption + Raman scattering from ML potential

CO2: Multiwavelength Raman Spectroscopy



Wavelength dependence: 532nm, 633nm, 785nm excitation sources

Glycol Example: Complete Pipeline

End-to-end workflow for ethylene glycol

```
# 1. Clean data (5,904  5,782 structures)
python -m mmml.cli.clean_data glycol.npz -o glycol_cleaned.npz

# 2. Split dataset (80/10/10)
python -m mmml.cli.split_dataset glycol_cleaned.npz -o splits/

# 3. Train model (auto-removes padding: 6010 atoms)
python -m mmml.cli.make_training \
    --data splits/data_train.npz \
    --ckpt_dir checkpoints/glycol --num_epochs 100

# 4. Evaluate on test set
python -m mmml.cli.evaluate_model \
    checkpoints/glycol/params*.pkl \
    --test-data splits/data_test.npz

# 5. Run MD simulation
python -m mmml.cli.dynamics \
    --checkpoint checkpoints/glycol/params*.pkl \
    --input glycol.xyz --temp 300 --steps 10000
```

Result: Production-ready model in ˜30 minutes!

Glycol Workflow: Key Results

Dataset statistics after cleaning:

Before Cleaning

- 5,904 structures
- 113 zero-energy structures
- 9 high-force structures
- Padded to 60 atoms

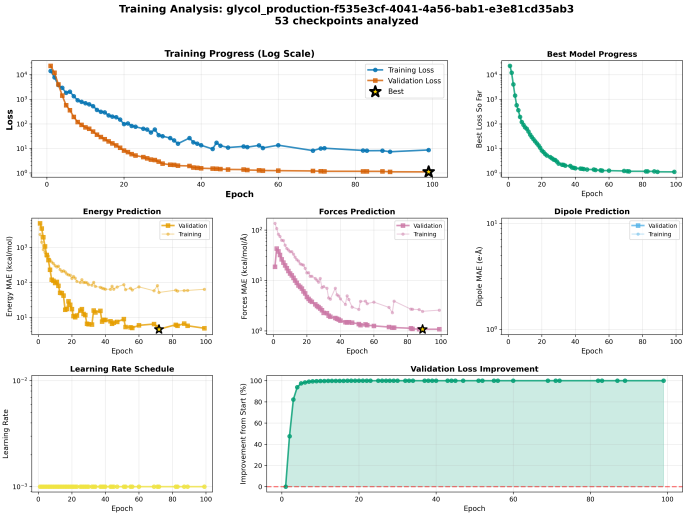
After Cleaning

- 5,782 structures
- Clean energy range
- All forces valid
- Auto-unpads to 10 atoms

Training efficiency:

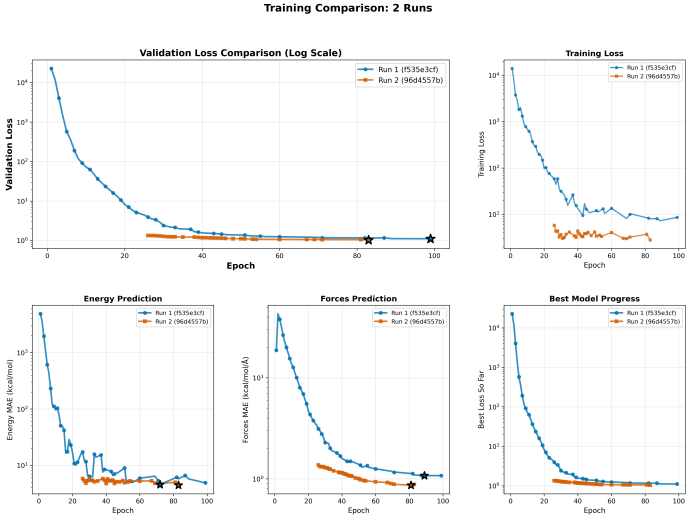
Metric	Before	After
Atoms processed	60	10
Training time	1.0×	**6.0× faster**
Memory usage	1.0×	**6.0× less**

Glycol Training Results



Metric	Value
Epochs	99
Checkpoints	53
Energy MAE (kcal/mol)	4.55
Forces MAE (kcal/mol/Å)	1.08
Convergence	100%
Best epoch	99

Glycol Training Comparison



Metric	Run 1	Run 2
Epochs	99	83
Checkpoints	53	29
Valid Loss	1.110	**1.031**
Energy MAE (kcal/mol)	4.55	**4.43**
Forces MAE (kcal/mol/Å)	1.08	**0.86**
Winner		**Run 2**
Improvement		**20%**

CO2 Example: ESP Prediction

Joint PhysNet+DCMNet for electrostatic potentials

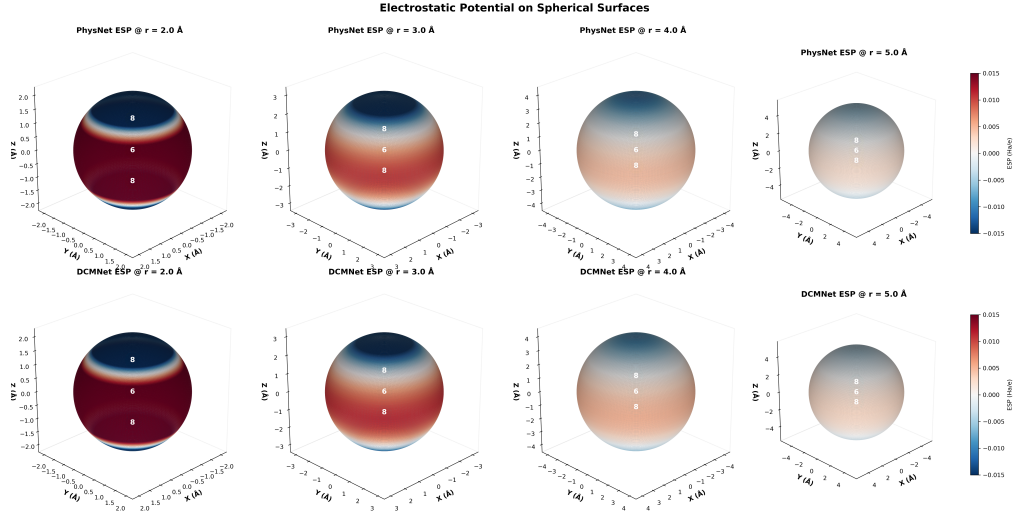
```
# Train joint model
python -m mmml.cli.train_joint \
    --train-efd examples/co2/train_efd.npz \
    --train-esp examples/co2/train_esp.npz \
    --valid-efd examples/co2/valid_efd.npz \
    --valid-esp examples/co2/valid_esp.npz \
    --epochs 100 --plot-freq 10

# Generates comprehensive plots:
# - Energy/forces/dipoles scatter plots
# - ESP prediction accuracy (3D visualization)
# - Distributed charge visualization
# - Radial ESP error analysis
```

Applications:

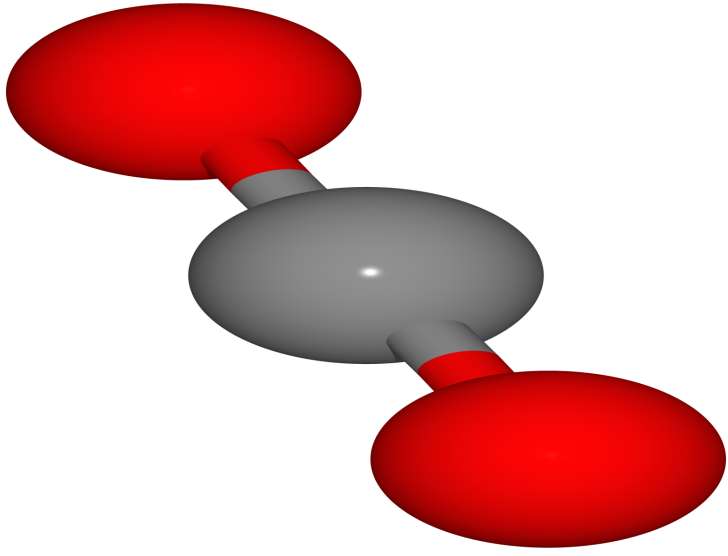
- Solvation free energies
- Protein-ligand docking
- Electrostatic properties
- Charge transfer analysis

CO2: ESP Visualization



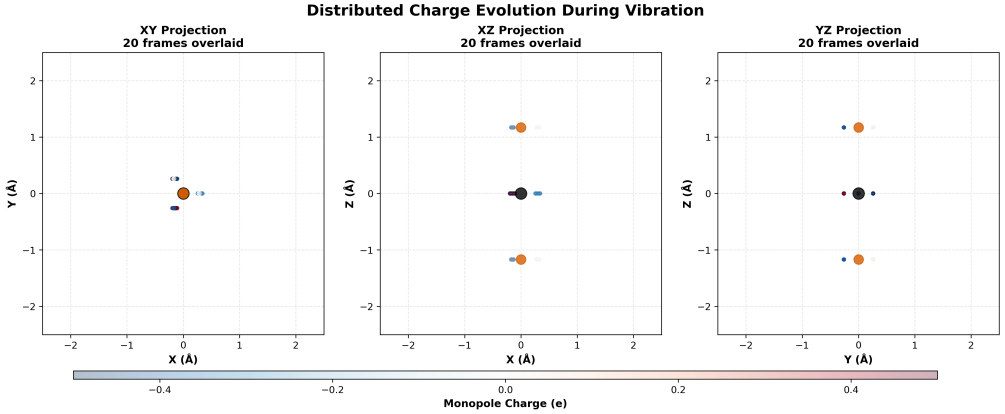
Electrostatic potential on molecular surface - High-resolution 3D rendering

CO2: Molecular Structure & Charges



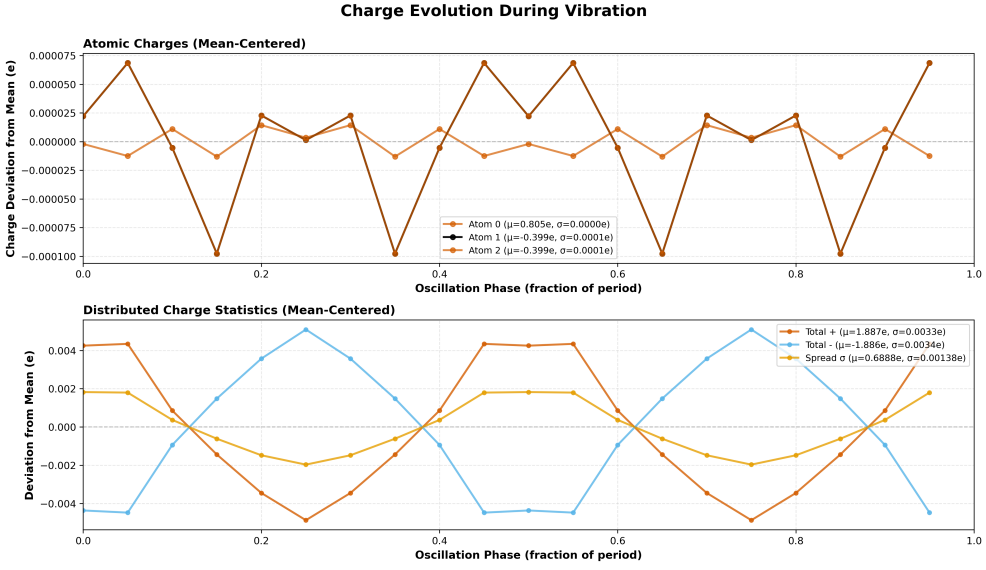
3D visualization: O=C=O linear geometry with distributed multipoles

CO2: Distributed Charges - 2D Projections



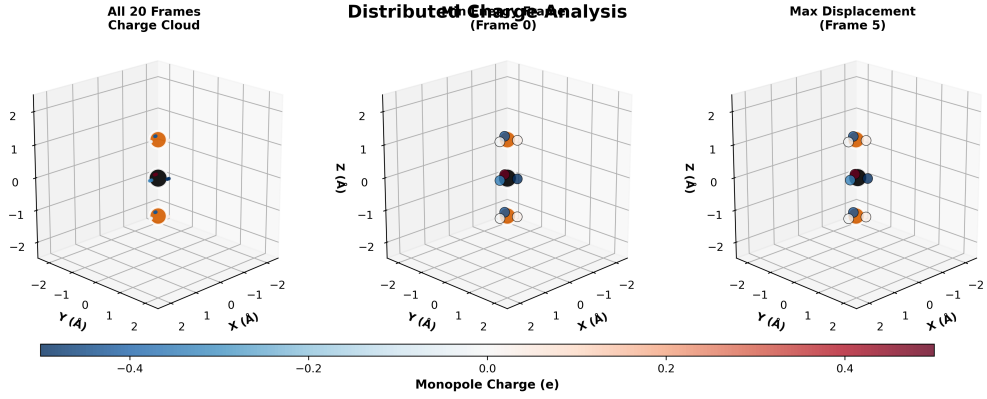
Multi-view analysis: XY, XZ, YZ projections reveal spatial charge arrangement

CO2: Charge Evolution During Vibration



Dynamic charges: Redistribution during asymmetric stretch and bend modes

CO2: 3D Charge Dynamics



Full 3D evolution: Charge positions and magnitudes during vibration

Acetone Example: Periodic Liquid Simulation

ML/MM hybrid simulation of bulk acetone

```
# Train on periodic system data
python -m mmml.cli.make_training \
    --data acetone_bulk.npz \
    --features 24 --num_iterations 2 \
    --cutoff 5.0 --batch_size 64

# Optimize ML/MM cutoffs
python -m mmml.cli.opt_mmml \
    --dataset dimers.npz \
    --ml-cutoff-grid 0.5,1.0,1.5,2.0 \
    --mm-switch-on-grid 4.0,5.0,6.0

# Run bulk simulation (100 molecules, 32 box)
python -m mmml.cli.run_sim \
    --pdbfile acetone_box.pdb --cell 32.0 \
    --n-monomers 100 --include-mm \
    --temperature 300 --nsteps 100000
```

Results: Liquid structure, diffusion, thermodynamics

Best Practices: Data Preparation

1. **Always clean your data first**
   - Remove failed calculations (zero energies)
   - Filter high forces (SCF convergence issues)
   - Check for overlapping atoms
2. **Explore before training**
   - Check energy distributions
   - Verify force statistics
   - Ensure balanced dataset
3. **Use proper splits**
   - 80% train, 10% validation, 10% test
   - Use fixed random seed for reproducibility
   - Keep splits separate!
4. **Let padding removal work automatically**
   - Don't manually specify `--num_atoms` unless needed
   - Training auto-detects and optimizes

Best Practices: Training

1. **Start with defaults**
   - Default hyperparameters work well
   - Auto-detection handles most cases
2. **Monitor training**
   - Check validation loss convergence
   - Use `plot_training` to diagnose issues
   - Early stopping if overfitting
3. **Adjust loss weights if needed**
   - Balance energy and forces
   - Consider property importance
   - Use validation metrics to guide
4. **Save checkpoints frequently**
   - Use `--restart` to continue training
   - Keep best validation checkpoint

Best Practices: Evaluation

1. **Always evaluate on held-out test set**
   - Never use test data during training
   - Report test set metrics
2. **Check multiple metrics**
   - MAE (mean absolute error)
   - RMSE (root mean squared error)
   - Maximum errors
   - Correlation coefficients
3. **Validate predictions**
   - Compare to reference calculations
   - Test on different conformations
   - Check energy conservation in MD
4. **Document everything**
   - Training parameters
   - Data preprocessing steps
   - Model performance metrics

Equivariant vs Non-Equivariant: The Question

Which architecture is better for ESP prediction?

DCMNet (Equivariant)

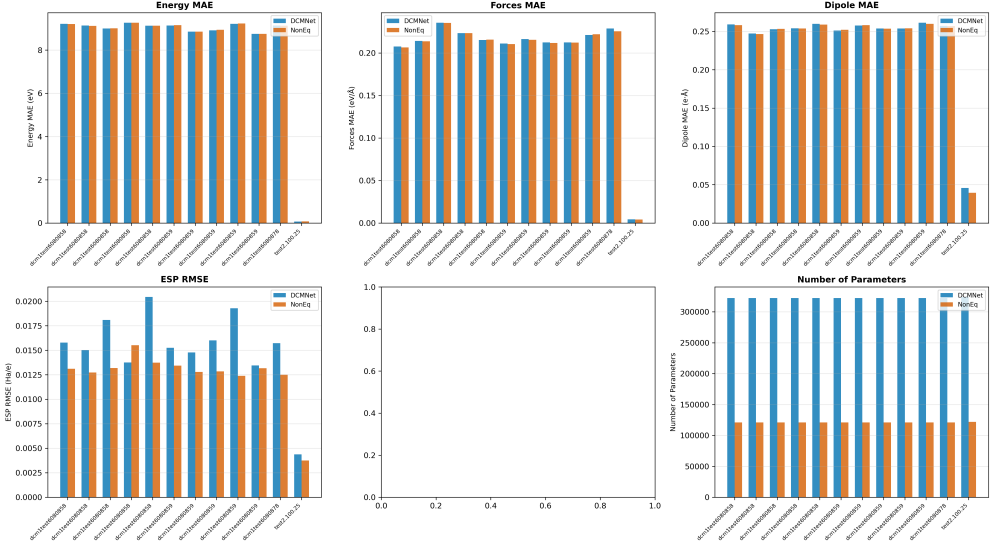
- Spherical harmonics
- Rotation-invariant
- Physically correct
- More parameters

NonEquivariant

- Cartesian displacements
- Direct prediction
- Simpler architecture
- Fewer parameters

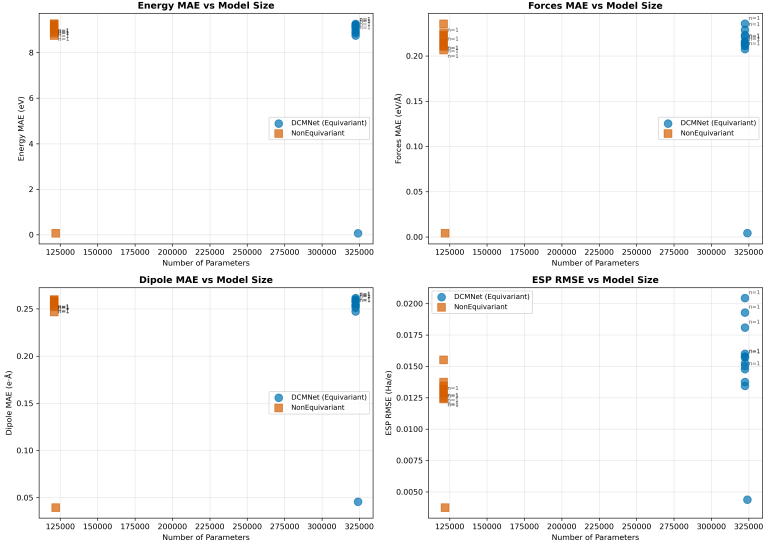
Analysis: 12 paired training runs with multiple n_dcm values
Tool: `compare_equivariant_models.py`

Test Set Performance



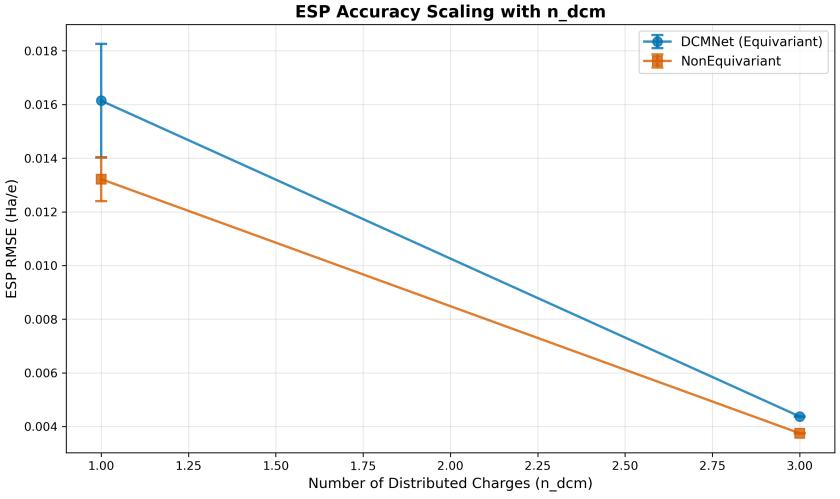
Finding: NonEquivariant has 18% better ESP accuracy on test set

Accuracy vs Model Complexity



Finding: NonEq achieves better ESP with 2.7× fewer parameters

Scaling with Distributed Charges



Optimal configuration: n_dcm = 5-6 charges per atom (diminishing returns after)

The Critical Test: Equivariance

Conclusion: NonEq overfit to training orientations!

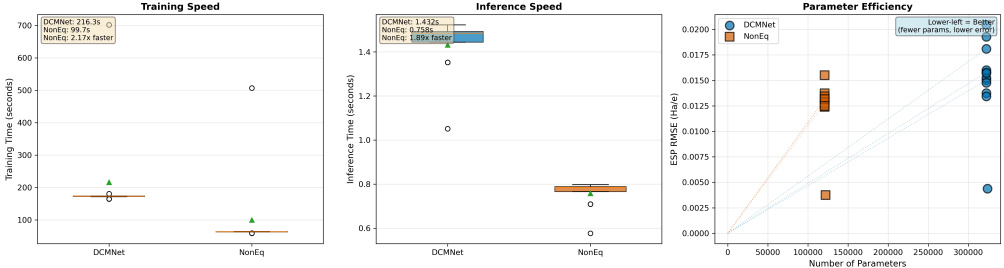
Critical Finding

DCMNet:
$\sim 10^{-6}$ rotation error
(machine precision!)

NonEq:
$\sim 10^{-3}$ rotation error
(**1000× worse!**)

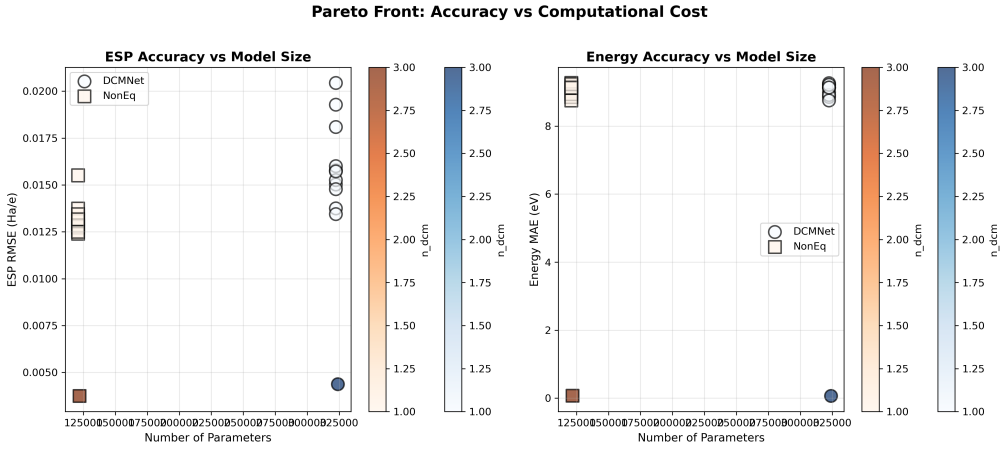
Ratio	**Value**
Dipole error	670×
ESP error	230×
Winner	**DCMNet**

Computational Efficiency



Trade-offs: NonEq 2.2× faster training, 1.9× faster inference; DCMNet 54% more parameter-efficient

Pareto Front: Accuracy vs Cost



ESP: NonEq dominates (lower-left) — **Energy:** Tied — **BUT:** Equivariance test changes everything!

Comparison Summary

NonEquivariant Wins:

- ESP test accuracy (18%)
- Training speed (2.2×)
- Inference speed (1.9×)
- Model size (2.7× smaller)

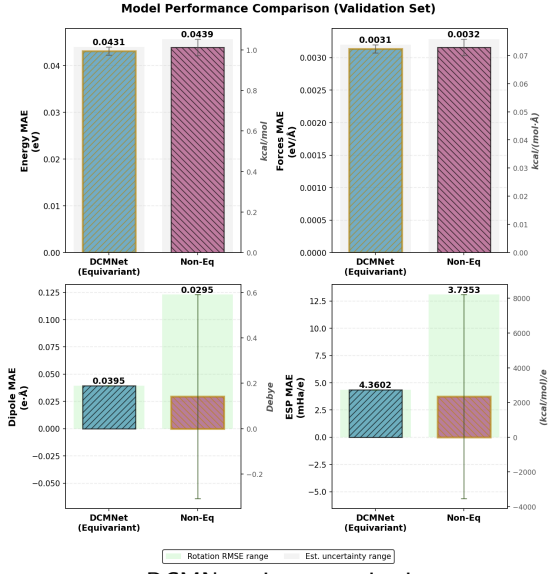
DCMNet Wins:

- **Equivariance (1000×!)**
- Parameter efficiency (54%)
- Physical correctness
- Generalization

Recommendation

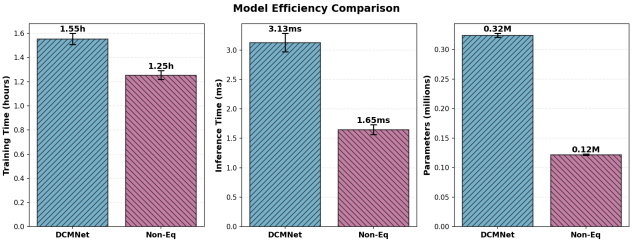
Use DCMNet for production. The equivariance test reveals that NonEq's better test accuracy is misleading - it only works for orientations similar to training data. DCMNet's guaranteed rotational invariance makes it the robust choice.

Performance Comparison: Detailed Metrics



Key finding: NonEq wins on test set, DCMNet wins on equivariance

Efficiency Comparison: Speed vs Accuracy



	DCM	NonEq
Train (s)	216	**100**
Infer (s)	1.43	**0.76**
Params (k)	323	**121**
ESP RMSE	0.0152	**0.0124**
Rotation Error	**Critical**	
	$\mathbf{10^{-6}}$	10^{-3}
Winner	**DCMNet** (Equivariance)	

When to Use Each Architecture

Use DCMNet When:

- Production deployment
- New orientations expected
- Physical correctness needed
- Generalization critical
- **DEFAULT CHOICE**

Use NonEquivariant When:

- Fixed orientations
- Heavy data augmentation
- Inference speed critical
- Model size constrained
- Prototyping only

Key Lesson: Test accuracy $\neq$ Real-world performance!
Always test equivariance for molecular models.

MMML CLI Tools Summary

Data Tools:

- `clean_data` - Quality control
- `explore_data` - Analysis
- `split_dataset` - Train/val/test

Training Tools:

- `make_training` - Basic
- `train_joint` - ESP
- `train_memmap` - Large-scale
- `train_charge_spin` - Multi-state

Evaluation Tools:

- `inspect_checkpoint` - Model info
- `evaluate_model` - Test metrics
- `plot_training` - History

Deployment Tools:

- `calculator` - ASE interface
- `dynamics` - MD simulations
- `run_sim` - ML/MM hybrid
- `opt_mmml` - Cutoff optimization
- `convert_npz_traj` - Visualization

16+ production-ready CLI tools + comprehensive documentation!

Key Innovations

1. **Automatic Padding Removal**
   - 6× training speedup
   - No manual intervention
   - Smart detection from data
2. **Comprehensive Data Cleaning**
   - Energy validation
   - Force filtering
   - Essential field extraction
3. **Joint PhysNet+DCMNet**
   - ESP prediction
   - Multiple architectures
   - Auto-tuned hyperparameters
4. **ML/MM Hybrid Simulations**
   - Combine ML and classical FF
   - Automatic cutoff optimization
   - Periodic boundary conditions
5. **Production-Ready Workflow**
   - End-to-end pipeline
   - HPC deployment support
   - Extensive validation

Getting Started

Installation:

```
pip install -e .
# or with optional features
pip install -e '.[plotting,tensorboard]'
```

Test installation:

```
python -m mmml.cli.test_deps
```

Quick start:

```
# 1. Clean data
python -m mmml.cli.clean_data data.npz -o clean.npz

# 2. Split dataset
python -m mmml.cli.split_dataset clean.npz -o splits/

# 3. Train model
python -m mmml.cli.make_training \
    --data splits/data_train.npz \
    --ckpt_dir checkpoints/
```

Documentation

Available documentation:

- `docs/cli.rst` - Basic CLI tools
- `docs/cli_advanced.rst` - Advanced training
- `examples/glycol/` - Complete workflow
- `examples/co2/` - ESP prediction
- `AI/` - Development notes

Example workflows:

- Glycol training (standard E/F/D)
- CO2 ESP prediction (PhysNet+DCMNet)
- Large-scale training (memory-mapped)
- Multi-state predictions (charge/spin)

All tools fully documented with examples!

Future Directions

Upcoming features:

1. **Enhanced Visualization**
   - Interactive MD trajectory viewer
   - 3D ESP visualization tools
   - Real-time training monitoring
2. **Additional Models**
   - Graph neural networks (GNNs)
   - Transformer architectures
   - Ensemble methods
3. **Advanced Features**
   - Active learning workflows
   - Uncertainty quantification
   - Transfer learning support
4. **Integration**
   - More quantum chemistry packages
   - Cloud training support
   - HPC job submission

Acknowledgments

Thank you for using MMML!

MMML Development Team

Built with:

- JAX & Flax (neural networks)
- e3x (equivariant operations)
- ASE (atomic simulations)
- NumPy & SciPy (numerical computing)

Questions?

GitHub: `mmml`
Documentation: `docs/`