Supplementary Information

Integrating machine learning interatomic potentials with batched optimization for crystal structure prediction

Chengxi Zhao[a,†*], Zhaojia Ma[b,c,†], Dingrui Fan[a,†], Siyu Hu[b,c], Leping Wang[b,c], Feng Hua[b,c], Weile Jia[b,c], En Shao[b,c,*], Guangming Tan[b,c,*], Jun Jiang[a,d,*], & Linjiang Chen[a,e,*]

[a] State Key Laboratory of Precision and Intelligent Chemistry, Hefei National Research Center for Physical Sciences at the Microscale, School of Chemistry and Materials Science, University of Science and Technology of China, Hefei, China

[b] State Key Laboratory of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

[c] University of Chinese Academy of Sciences, Beijing, China

[d] Hefei National Laboratory, University of Science and Technology of China, Hefei, China

[e] School of Chemistry, School of Computer Science, University of Birmingham, Birmingham, UK

† These authors contributed equally: C.Z., Z.M., & D.F.

* Corresponding authors: Chengxi_zhao@ustc.edu.cn (C.Z.), shaoen@ict.ac.cn (E.S.), tgm@ict.ac.cn (G.T.); jiangj1@ustc.edu.cn (J.J.); linjiangchen@ustc.edu.cn (L.C.)

Table of Contents

1 Supplementary Methods

Batched quasi-Newton optimization on GPU

To enhance GPU throughput, this study rewrites the quasi-Newton algorithm in ASE as a batched version in Algorithm 5. First, the matrix and vector computations in lines 3 and 13, as well as the model inference in line 7, are accelerated using the GPU. Second, the algorithm addresses the uncertainty in iteration counts introduced by the double-loop scenario of quasi-Newton and line search. Due to this uncertainty, synchronization before inference of machine learning potentials becomes challenging, making it impossible to predict energies and forces for multiple crystals simultaneously via batching.

We observe that in the quasi-Newton method, each crystal's optimization step follows a repeating pattern of "machine learning potential → BFGS update/LineSearch update → machine learning potential → BFGS update/LineSearch update." Specifically:

1. Each model inference is invariably followed by an optimizer update.
2. The optimizer update could either be a LineSearch step length update or a BFGS update approximating the inverse Hessian matrix.
3. LineSearch updates and BFGS updates are mutually exclusive—only one occurs at a time.

Thus, in lines 3, 5, and 13 of the algorithm, a mask is used to determine which crystals require updates to the Hessian approximation in BFGS or the step size in LineSearch. This mask is updated in line 9 based on LineSearch's convergence criteria. Subsequently, in line 7, forces and energies are predicted simultaneously for all crystals in a batch. The batched inference of *calc* improves computational efficiency compared to parallel process-based computation of multiple *calc* instances.

Algorithm 1 BFGSFusedLS supported by multiple models

1: Initialize x^{batch}, H^{batch}, *mask*, *calc*
2: **while** noCrystalConvergence **do**
3: p^{batch} ← direction (H^{batch}, F^{batch}, *mask*)
4: **while** noLineSearchStop **do**
5: α^{batch} ← update(*mask*)
6: $x$batch trial ← nextPos(x^{batch}, p^{batch}, α^{batch}, *mask*)
7: Fbatch trial ← calc.predict($x$batch trial)
8 **if** Wolfe($x$batch trial, p^{batch}, α^{batch}, F^{batch}, *mask*)
9 *mask* ← updateMask(*mask*)
10 **break**
11 **end if**
12 **end while**
13 H^{batch}, x^{batch} ← update(H^{batch}, α^{batch}, Fbatch trial, *mask*)
14 **end while**

Efficient neighbor-list updates with periodic boundary conditions

After the batch optimizer can invoke machine learning potential functions in batch inference mode, neighbor list updates become the next bottleneck determining overall throughput. Existing methods lack a way to simultaneously update neighbor lists for atoms in multiple crystals, resulting in poor parallelism at this stage, especially for larger crystals, where computation is slow and GPU memory usage is high. To address this issue, this paper implements the neighbor update algorithm described in Algorithm 6.

This algorithm is based on a naive approach that accounts for spatial periodicity through triple loops, with the following optimizations:

(1) On line 11 of the algorithm, CUDA is used to accelerate the neighbor calculation process, effectively leveraging GPU acceleration.

(2) Within the loop, the bottleneck in distance calculation lies in reading *pos1* and *pos2* (from global memory to registers) and writing *mask* (from registers to global memory), which are operations on arrays of size *batch* * n^2, making it a typical memory-bound operation. We fuse the PBC Offset addition with atomic positions in the GPU kernel on line 12, thereby reducing one read and write operation for *pos1* at the cost of an additional read (from global memory to registers) for PBC Offset of size *batch*3*3*. This reduces the total memory access volume to 3/5 of the original, improving computation speed. Additionally, we vectorize memory access in the kernel to further enhance read/write speeds.

(3) The algorithm allows concatenating position information from multiple crystals at the input and appending a *batch id*, enabling simultaneous computation of edges from multiple crystals.

(4) Depending on the crystal size, multiple CPP implementations are compiled, with different implementations dynamically unrolling the loop in lines 6-8 of the algorithm, trading space for time as much as possible under limited memory constraints.

Algorithm 2 CUDA-Accelerated PBC neighbor list

Require: Data *D*, radius *r*, PBC flags **pbc**

Ensure: Edge indices **E**, unit cell offsets **U**

Generate all atom pairs: $\mathbf{index_1}$, $\mathbf{index_2}$

Compute unit cell repetitions: $rep_d = CEIL(r \cdot \| b_d \|)$

GPU Acceleration: Load CUDA extension: pbc_graph_cuda

Pre-allocate GPU tensors: **d**2 gpu, $\mathbf{mask}_{gpu}$, $\mathbf{offset}_{gpu}$

Compute batch offsets: **batch_offsets** = cumsum($\mathbf{natoms}^2$)

for i = -2rep_0 to 2rep_0 **do**

  for j = -2rep_1 to 2rep_1 **do**

    for k = -2rep_2 to 2rep_2 **do**

      Create unit cell: **u** = [i, j, k]

      Compute PBC offsets: **o** = $\mathbf{cell}^T \cdot \mathbf{u}$

      CUDA Kernel Launch:

      pbc_distance_kernel($\mathbf{pos_1}$, $\mathbf{pos_2}$, **o**, **d**2 gpu, $\mathbf{mask}_{gpu}$)

      valid_indices = nonzero($\mathbf{mask}_{gpu}$)

      if |**valid_indices**| > 0 **then**

        Collect GPU results to CPU containers

      end if

    end for

  end for

end for

cudaDeviceSynchronize()

Concatenate GPU results: **E**, **U** = cat_gpu_results()

return E, **U**

Continuous scheduling

When the machine learning potential function and neighbor list updates achieve high parallelism by batching, in order to keep the GPU fully loaded during the optimization of a large number of candidate crystal structures, we specifically designed the scheduling strategy in Algorithm 7. In the batch optimization process of multiple crystal structures, the number of iterations for each crystal varies, so Algorithm 7 implements a continuous batch strategy. Line 3 of Algorithm 7 will select an appropriate batch size based on the number of atoms of the structures currently being processed, convert them into batch data, and generate unit Batch to pass to the optimizer for optimization. Line 6 of Algorithm 7 will return the optimized structures, line 7 performs asynchronous write-back of the structures, line 8 calculates the space released by the converged structures, and line 3 will add new crystal structures to the batch based on the available space in the current batch. In the prepare step at line 3 of the algorithm, when prepare is called for the first time, it triggers a sorting of the crystal structures by size. The prepare method will add structures to the batch in descending order of size. According to formula (1), where N_i is the number of atoms in the i-th structure in the batch, the prepare method checks whether the total number of atoms of all current structures exceeds an empirical value N_{max} each time a crystal structure is added to the batch.

$$\Sigma_i^n N_i \leq \frac{N_{max}}{P} \#(1)$$

Algorithm 3 continuous batching

Initialize *inputPath*, *mask*, *unitBatch*, *optimizer*, *batchSize*

while *convergedCount* < *totalCount* **do**

  batch ← prepare(*inputPath*, *batch*, *mask*, *room*)

  unitBatch ← updateUnitCell(*unitBatch*, *batch*, *mask*)

  optimizer ← optimizer.restore(*unitBatch*, *mask*)

6: *results*, *mask* ← optimizer.run(*batch*)
7: writeResult (*result*, *mask*)
8: *convergedCount*, *room* ← update(*mask*)
9: **end while**

Acceleration efficient test

1. Technical Environment

Hardware: All benchmarks were performed on a server with dual Intel Xeon Platinum 8558 CPUs (2.0 TiB RAM) and 4× NVIDIA H100 80 GB GPUs.

Software: The environment includes Python 3.10, PyTorch 2.4.1+cu121, ASE 3.23.0, e3nn 0.4.4, torch-geometric 2.6.1, and cuEquivariance 0.4.0. The scripts for reproduction are available in our public repository.

2. Baseline vs. Proposed Batching Strategy

Baseline Configuration: The baseline utilizes ASE's standard BFGSLineSearch optimizer. To maximize CPU-level parallelism while avoiding resource contention, we used 64 processes for systems with <= 368 atoms and 60 processes for 736 atoms, with MKL/OMP/OpenBLAS threads fixed to 1. NVIDIA MPS (Multi-Process Service) was enabled to allow multiple CPU processes to share GPU resources efficiently.

Proposed Method: Our method employs GPU-accelerated quasi-Newton BFGS with a continuous batching strategy (Algorithms 1 & 3). The batch size is dynamically managed by the scheduler with an atom-count cap of Nmax = 22,000 to optimize GPU saturation and memory pressure.

3. Wall-clock Comparison & Results

Detailed wall-clock performance data is now provided in Table S1 of SI. Our method demonstrates significant and consistent runtime reductions across all tested crystal sizes:

For small-scale systems (46 atoms, 400 crystals), the runtime was reduced from 171.01s to 95.90s (1.78× speedup).

For large-scale systems (736 atoms, 400 crystals), the runtime decreased from 6293.10s to 2448.02s, representing a 2.57× speedup.

Across all benchmarks, we achieved an average speedup of 2.31× on the mixed-size test set.

We also tested the optimization speed by running 1,000 identical structures for each of the five MLIPs under the same conditions. The results are as follows: Mace ~ 4 min 31 s, Sevennet ~ 13 min 57 s, eqnorm ~ 16 min 59 s, orb ~ 65 min 32 s, and matris ~ 53 min 28 s. Note that all of these runs were performed before enabling the batched-optimization CSP acceleration and, more importantly, they were executed with the same multiprocessing setting. This allows us to compare the runtimes of different models under strictly controlled conditions. However, this setup does not account for GPU memory consumption. In practical CSP workflows, larger models typically occupy substantially more memory, which limits the achievable degree of parallelism; as a result, the runtime gap between smaller and larger models can become even more pronounced than what is shown in this normalized comparison. Here we provide these numbers only as an intuitive side-by-side comparison to give a sense of the relative computational cost of different models.

SOAP and UMAP parameter settings

SOAP descriptors were generated using DScribe with: r_cut = 6.0 Å, n_max = 6, l_max = 4, and periodic = True. Structural similarities were then evaluated using DScribe's AverageKernel with metric = "linear". No additional averaging was applied at the SOAP-generation stage (average = "off"), and all other SOAP parameters were kept at their DScribe default values.

For UMAP, we used n_neighbors = 20, min_dist = 0.5, n_components = 2, and metric = "precomputed". The input to UMAP was a precomputed distance matrix derived from the SOAP kernel matrix, calculated as D = 1.00001 - S, where S is the SOAP similarity matrix. No random seed was specified for the UMAP analysis.

Optimization hyperparameters and convergence

Our BFGSFusedLS optimizer inherits the default hyperparameters of ASE's BFGSLineSearch: maximum atomic displacement "maxstep"=0.2Å, Wolfe coefficients c_1=0.23and c_2=0.46, initial step scale α=10.0, step bounds "stpmin"=10^(-8)and "stpmax"=50.0, extrapolation bounds x_"trapl" =1.1and x_"trapu" =4.0, termination tolerance x_"tol" =10^(-14), and initial trial step α_1=1.0.

Convergence is defined purely by the force criterion f_max=0.01eV/Å (set at the task level). Cell degrees of freedom are optimized simultaneously through ASE's ExpCellFilter under the same force threshold; therefore, separate stress or cell-change tolerances are not used.

Robustness and exception handling.

Several engineering safeguards are implemented to ensure stability during large-scale batch optimization:

1. Masked updates and restarts. An early-stop mechanism with restart_indices and old_batch_indices tracks structure states. BFGS and line-search updates are applied only to unconverged structures via masking.
2. Line-search safeguards. The batch line search uses a maximum of 15 iterations. If a WARN or ERROR state occurs, the algorithm falls back to a conservative step size.
3. Abnormal state detection. An upper force limit f_"upper" =10^25eV/Å is imposed to detect numerical instabilities. Structures reaching the maximum step limit or encountering unrecoverable errors are removed from the batch and replaced by new candidates through the continuous scheduler.
4. Worker recovery. If a worker subprocess exits abnormally (e.g., exit codes −11 or 1), it is automatically restarted to maintain uninterrupted execution.

2 Supplementary Figures and Tables

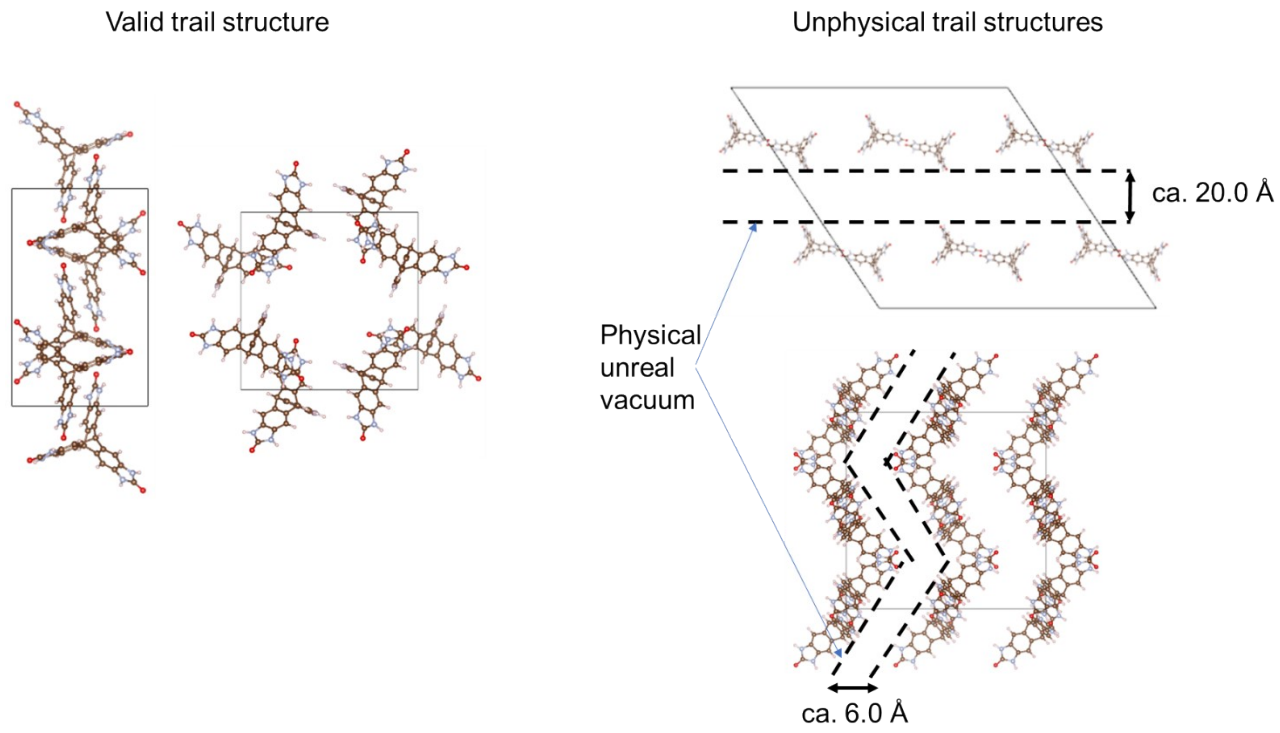

Figure S1: Example of generated trial structures exhibiting unphysical vacuum regions, which fail the validation test and are excluded from the geometry optimization step.

Table S1: Summary of speed-up test for the top 21 common space group.

Number of atoms	Number of crystals	Baseline(s)	Batchopt(s)	Speed-up factor
46	400	171.01	95.90	1.78
92	1200	654.89	281.95	2.32
184	3600	4282.04	1971.00	2.17
276	400	829.40	449.71	1.84
368	2400	8756.38	4444.64	1.97
736	400	6293.10	2448.02	2.57

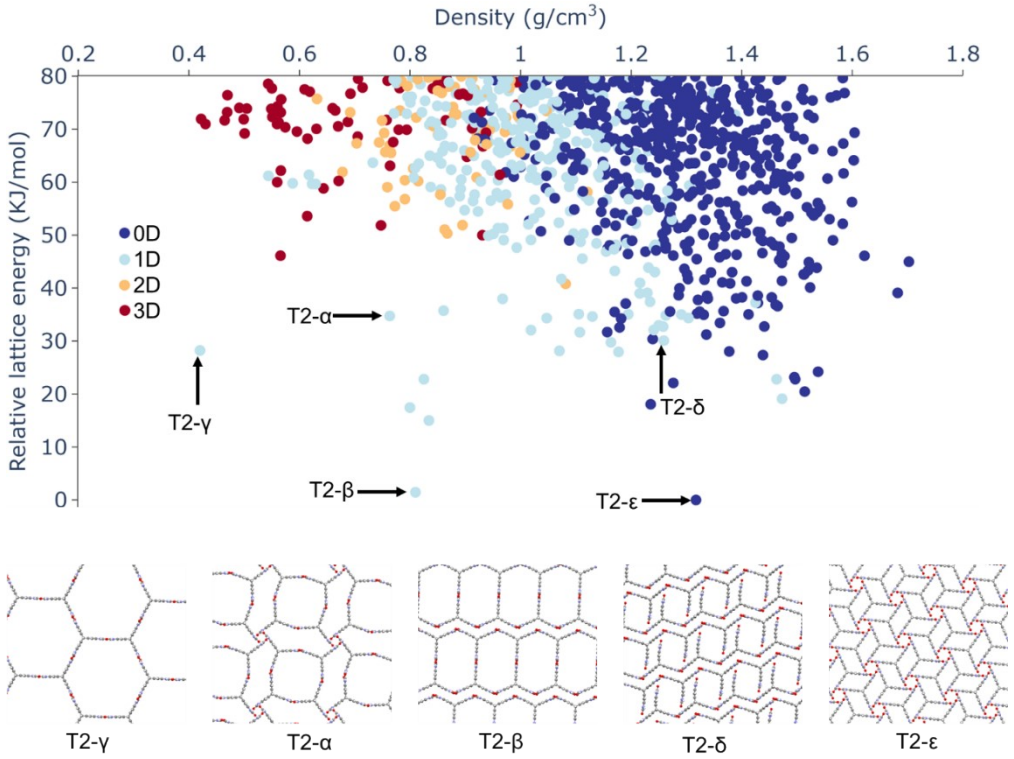

Figure S2: CSP energy–density plots for molecule T2, color-coded by channel dimension using a 1.7 Å probe (also referred to as an energy–structure–function map). All experimentally observed crystal structures are labeled on the plot. Observed energy spikes are consistent with those reported in previous studies, and their relative lattice energies fall within a reasonable range.

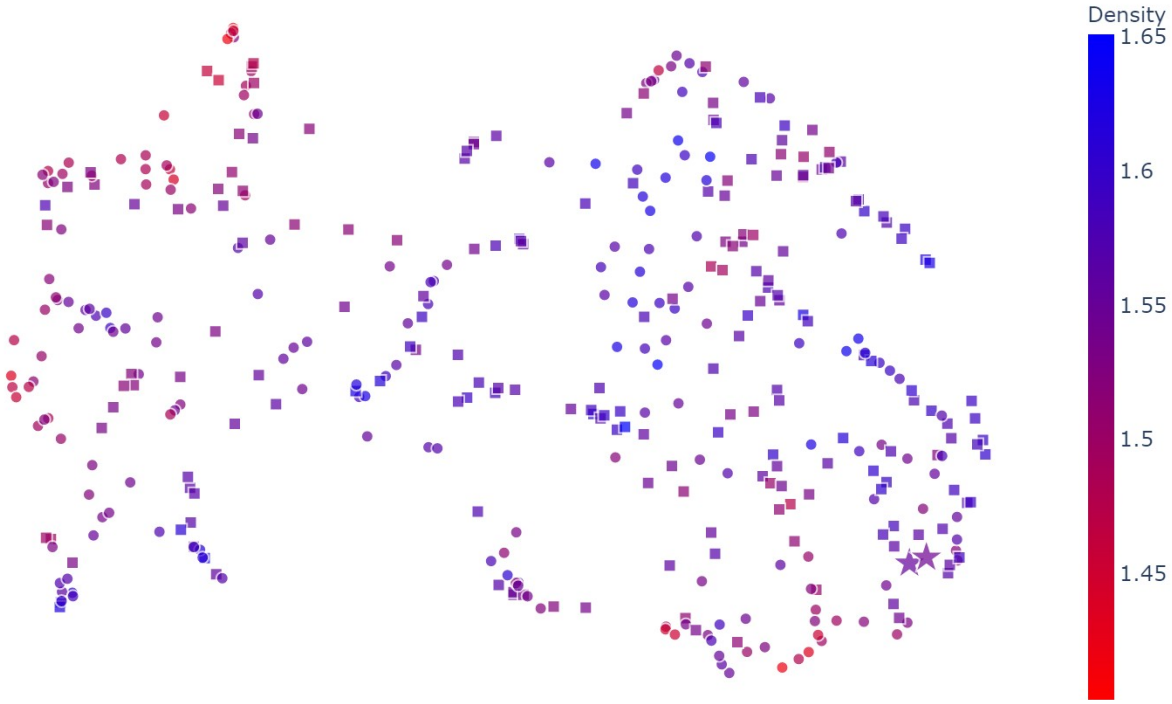

Figure S3: Two-dimensional UMAP embedding of the SOAP feature space for structure VI, color-coded by crystal density. Only the 200 lowest-energy structures from each model, after duplicate removal, are shown. Square symbols represent MACE-OFF-small optimized structures, while circle symbols represent SevenNet-0-D3 optimized structures. Structures that match the experimental crystal structures are highlighted with star symbols.

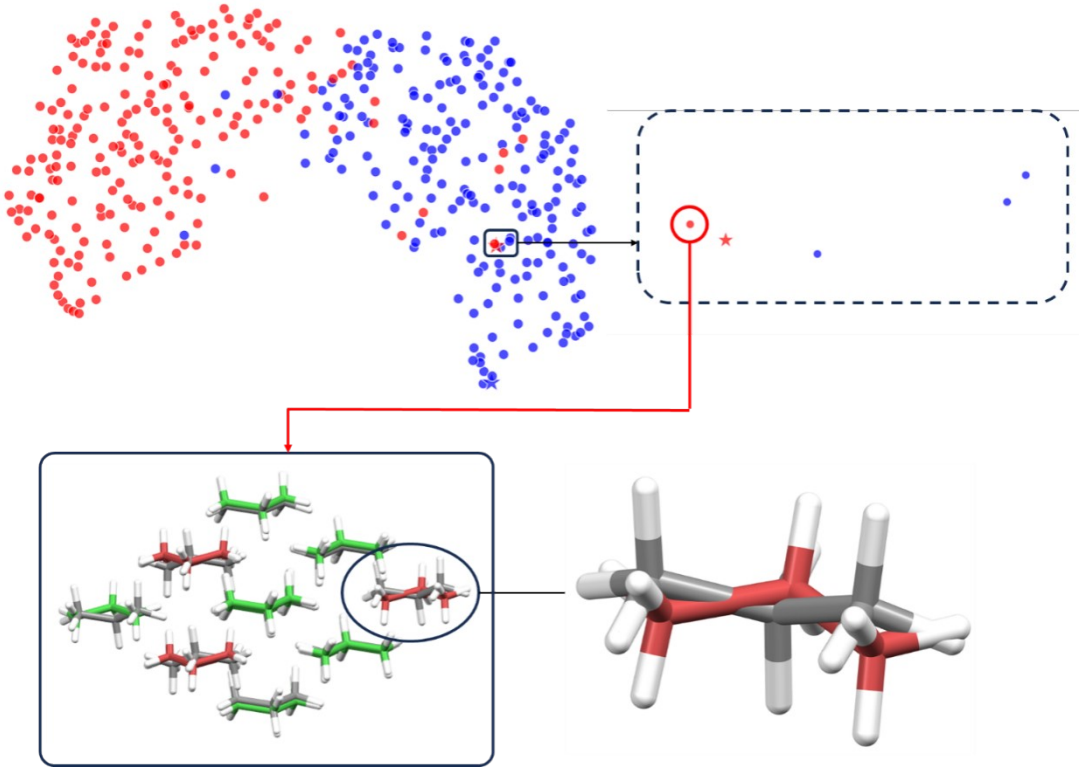

Figure S4: Packing similarity results for the predicted crystal structures optimized using the SevenNet-0-D3 model that exhibit the highest similarity to the experimental structure. In a 15-molecule cluster, matched molecules are shown in green and unmatched ones in red. Note that in the UMAP plot, the experimental structure (indicated by a red star) was added manually; therefore, no 15/15 packing match was identified in the CSP results obtained with SevenNet-0-D3 model, as reflected in the summary table.

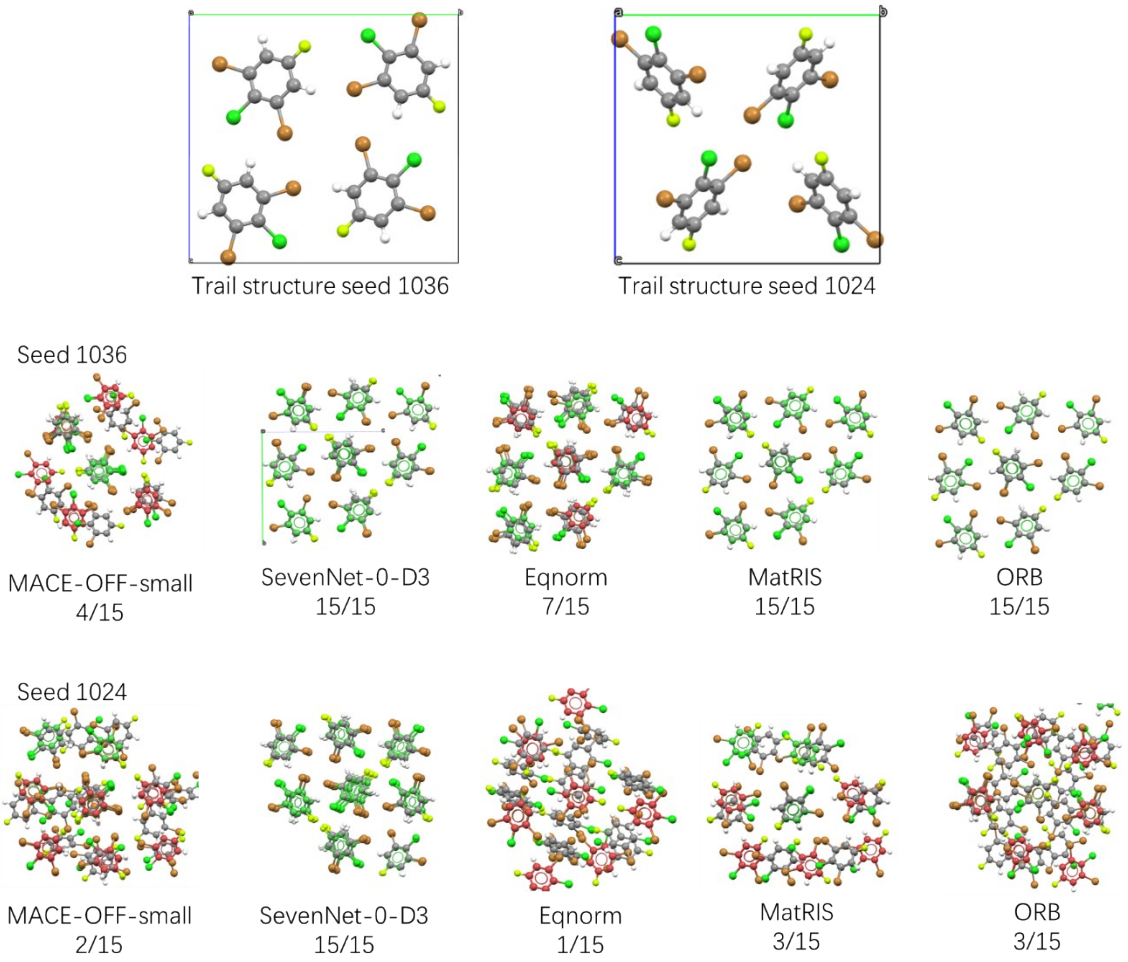

Figure S5: Packing similarity analysis for seed IDs from which at least one MLIP successfully recovers the experimental structure. Seed 1036 generates trial structure close to the final experimental packing, whereas seed 1024 generate trail structure far from the relevant local minimum on the energy landscape. The upper figures are the trail structures while the middle and bottom figures are structures after optimization by different MLIPs. The top panels show the trial structures, while the middle and bottom panels show the structures after optimization by different MLIPs.

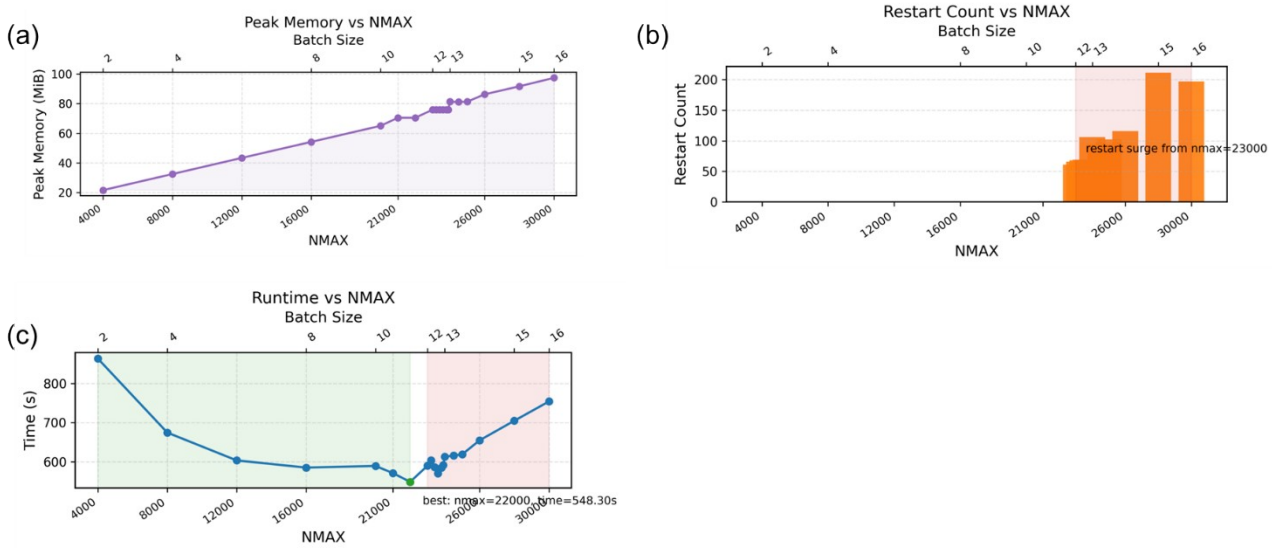

Figure S6: Sensitivity analysis of the atom-count cap N_{max}. N_{max} was varied from 4000 to 30000, corresponding to batch sizes from 2 to 16.

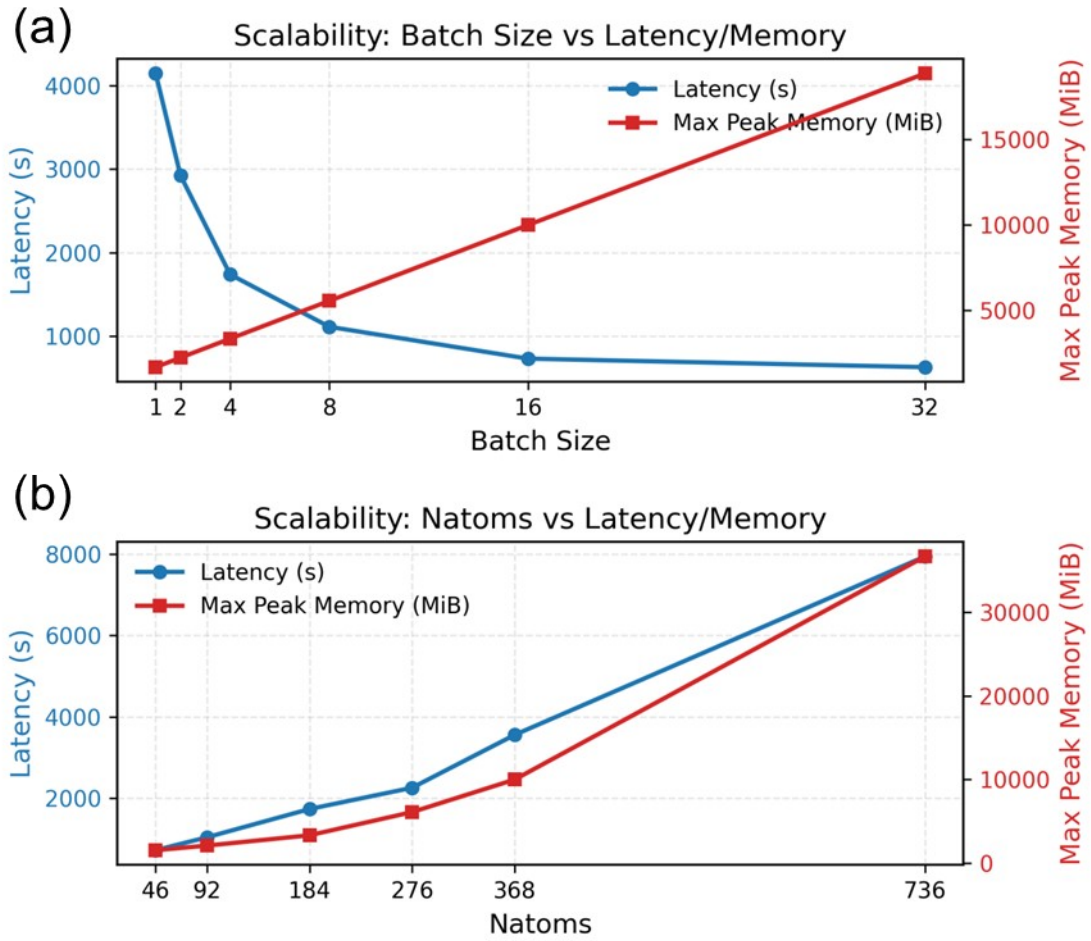

Figure S7: (a) Scalability with batch size. Latency and maximum peak GPU memory as a function of batch size for a fixed system size of 184 atoms. (b) Scalability with atom count. Latency and maximum peak GPU memory as a function of the number of atoms for a fixed batch size of 4.

Table S2: Summary of CSP results for 26 molecular systems using two different foundation atomistic models. "Exp." refers to the experimentally observed structures retrieved from the CCDC database, with corresponding Refcodes provided. Exp. Stability indicates whether the experimental structure remains stable during optimization with the respective foundation atomistic model. Exp. Hit Count denotes the number of times the experimental structure was predicted during the CSP process. Exp. Relative Energy represents the energy difference of the experimental structure relative to the lowest-energy predicted structure. Exp. Rank indicates the energy ranking position of the experimental structure among all predicted structures. Energies are given in KJ/mol.

		MACE-OFF-small				SevenNet-0-D3			
CCDC Refcode	Blind test refcode	Exp. Stability	Exp. Hit Count	Exp. Relative Energy	Exp. Rank (after deduplication)	Exp. Stability	Exp. Hit Count	Exp. Relative Energy	Exp. Rank (after deduplication)
XULDUD	I-1	Yes	11	6.82	76	Yes	5	9.26	69
XULDUD01	I-2	Yes	88	0	1	Yes	3	13.76	380
GUFJOG	II	Yes	5	11.21	76	Yes	21	2.24	11
QAMTAZ	III	No[a]	X	X	X	No	X	X	X
BOQQUT	IV-1	Yes	62	0	1	Yes	101	0.62	4
BOQQUT01	IV-2	Yes	0	0.44	2	Yes	0	0.75	7
BOQWIN	V	Yes	29	15.14	16	Yes	124	0.82	2
UJIRIO	VI	Yes	5	1.32	3	Yes	1	0	1
JAYDUI	VII	Yes	10	0	1	Yes	0	0	1
PAHYON01	VIII	Yes	23	0.36	3	Yes	117	0.49	2
XATMIP	IX	No	X	X	X	Yes	3	13.35	67
HAMTIZ01	X	Yes	0	14.01	161	Yes	2	11.06	181
XATMOV	XI	Yes	0	9.13	284	No	X	X	X
AXOSOW01	XII	Yes	306	0	1	Yes	536	3.94	23
SOXLEX01	XIII	Yes	0	12.85	222	Yes	8	7.09	88
WIDBAO	XIV	Yes	29	7.4	2	Yes	20	2.96	2
WICZUF	XV	Yes	0	8.98	3	Yes	0	18.07	306
OBEQUJ	XVI	Yes	76	1.41	2	Yes	11	12.94	236
OBEQOD	XVII	Yes	1	60.11	1014	Yes	13	4.56	36
OBEQET	XVIII	No	X	X	X	Yes	5	6.07	13
XATJOT	XIX	Yes	0	0	1	Yes	2	5.31	12
OBEQIX	XX	Yes	0	4.46	4	Yes	0	17.42	32
KONTIQ	XXI-1	Yes	2	17.25	14	Yes	2	0	1
KONTIQ01	XXI-2	Yes	1	7.71	5	Yes	0	18.32	35
KONTIQ03	XXI-3	Yes	1	0	1	Yes	0	15.1	22
NACJAF	XXII	No	X	X	X	No	X	X	X
XAFPAY	XXIII-1	Yes	1	16.74	84	Yes	0	20.59	379
XAFPAY01	XXIII-2	Yes	1	8.95	22	Yes	0	20.41	359
XAFPAY02	XXIII-3	Yes	0	4.68	4	Yes	0	15.87	134
XAFPAY03	XXIII-4	Yes	2	14.69	58	Yes	0	12.55	53
XAFPAY04	XXIII-5	Yes	0	18.33	111	Yes	0	23.53	612
XAFQON	XXIV	No	X	X	X	Yes	0	10.74	36
XAFQAZ	XXV	Yes	0	1.49	2	No	X	X	X
XAFQIH	XXVI	Yes	3	0	1	Yes	4	10.72	17

[a] The MACE-OFF-small MLIP did not include the boron element in its training set; therefore, this system was considered a failed prediction due to lack of elemental coverage.

Table S3: Summary of CSP results for selected systems using five different foundation models. "Exp." refers to the experimentally observed structures retrieved from the CCDC database, with corresponding Refcodes provided. Exp. Stability indicates whether the experimental structure remains stable during optimization with the respective foundation atomistic models. Exp. Hit Count denotes the number of times the experimental structure was predicted during the CSP process. Exp. Relative Energy represents the energy difference of the experimental structure relative to the lowest-energy predicted structure. Exp. Rank indicates the energy ranking position of the experimental structure among all predicted structures. Energies are given in KJ/mol.

		MACE-OFF-small				SevenNet-0-D3			
CCDC Refcode	Blind test refcode	Exp. Stability	Exp. Hit Count	Exp. Relative Energy	Exp. Rank (after deduplication)	Exp. Stability	Exp. Hit Count	Exp. Relative Energy	Exp. Rank (after deduplication)
SOXLEX01	XIII	Yes	0	12.85	222	Yes	8	7.09	88
KONTIQ	XXI-1	Yes	2	17.25	14	Yes	2	0	1
KONTIQ01	XXI-2	Yes	1	7.71	5	Yes	0	18.32	35
KONTIQ03	XXI-3	Yes	1	0	1	Yes	0	15.1	22

		Eqnorm				ORB			
CCDC Refcode	Blind test refcode	Exp. Stability	Exp. Hit Count	Exp. Relative Energy	Exp. Rank (after deduplication)	Exp. Stability	Exp. Hit Count	Exp. Relative Energy	Exp. Rank (after deduplication)
SOXLEX01	XIII	Yes	26	0.00	1	Yes	27	0.00	1
KONTIQ	XXI-1	Yes	1	0.00	1	Yes	0	0.00	1
KONTIQ01	XXI-2	Yes	0	15.31	26	Yes	0	16.51	26
KONTIQ03	XXI-3	Yes	0	13.72	19	Yes	0	13.65	18

		MatRIS			DFT	
CCDC Refcode	Blind test refcode	Exp. Stability	Exp. Hit Count	Exp. Relative Energy	Exp. Rank (after deduplication)	Exp. Relative Energy
SOXLEX01	XIII	Yes	12	0.09	2	-
KONTIQ	XXI-1	Yes	1	0.00	1	0.00
KONTIQ01	XXI-2	Yes	1	4.63	5	3.60
KONTIQ03	XXI-3	Yes	0	8.68	14	6.92