



Bochum

9 June 2022

# Pacemaker – a tool for atomic cluster expansion fitting

Lysogorskiy Yury, Anton Bochkarev, Matteo Rinaldi, Sarath Menon,  
Matous Mrovec, Ralf Drautz

Interdisciplinary Centre for Advanced Materials Simulation (ICAMS)

Ruhr-Universität Bochum, Germany

[yury.lysogorskiy@rub.de](mailto:yury.lysogorskiy@rub.de)

ICAMS

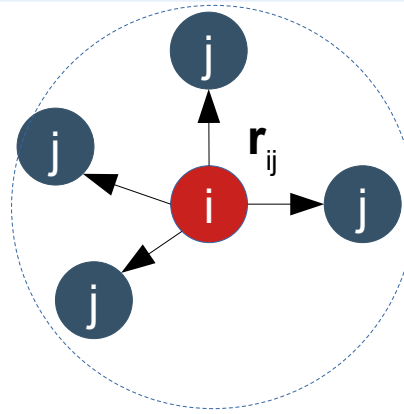
INTERDISCIPLINARY CENTRE FOR  
ADVANCED MATERIALS SIMULATION

RUHR  
UNIVERSITÄT  
BOCHUM

RUB

# Atomic cluster expansion - basics

1) Atom ("i") and its neighbors ("j") within cutoff  $r_c$



2) one-particle basis function for each bond  $\mathbf{r}_{ij}$ :

$$\phi_{\mu_i \mu_j n l m} = R_{nl}^{\mu_i \mu_j}(r_{ji}) Y_{lm}(\hat{\mathbf{r}}_{ji})$$

*translational invariance*

3) Atomic base A:  
(sum up over neighbors)

$$A_{i \mu n l m} = \sum_j \delta_{\mu \mu_j} \phi_{\mu_i \mu_j n l m}(\mathbf{r}_{ji})$$

$(n, l, m)$  – various indices

*permutation invariance*

4) A-product

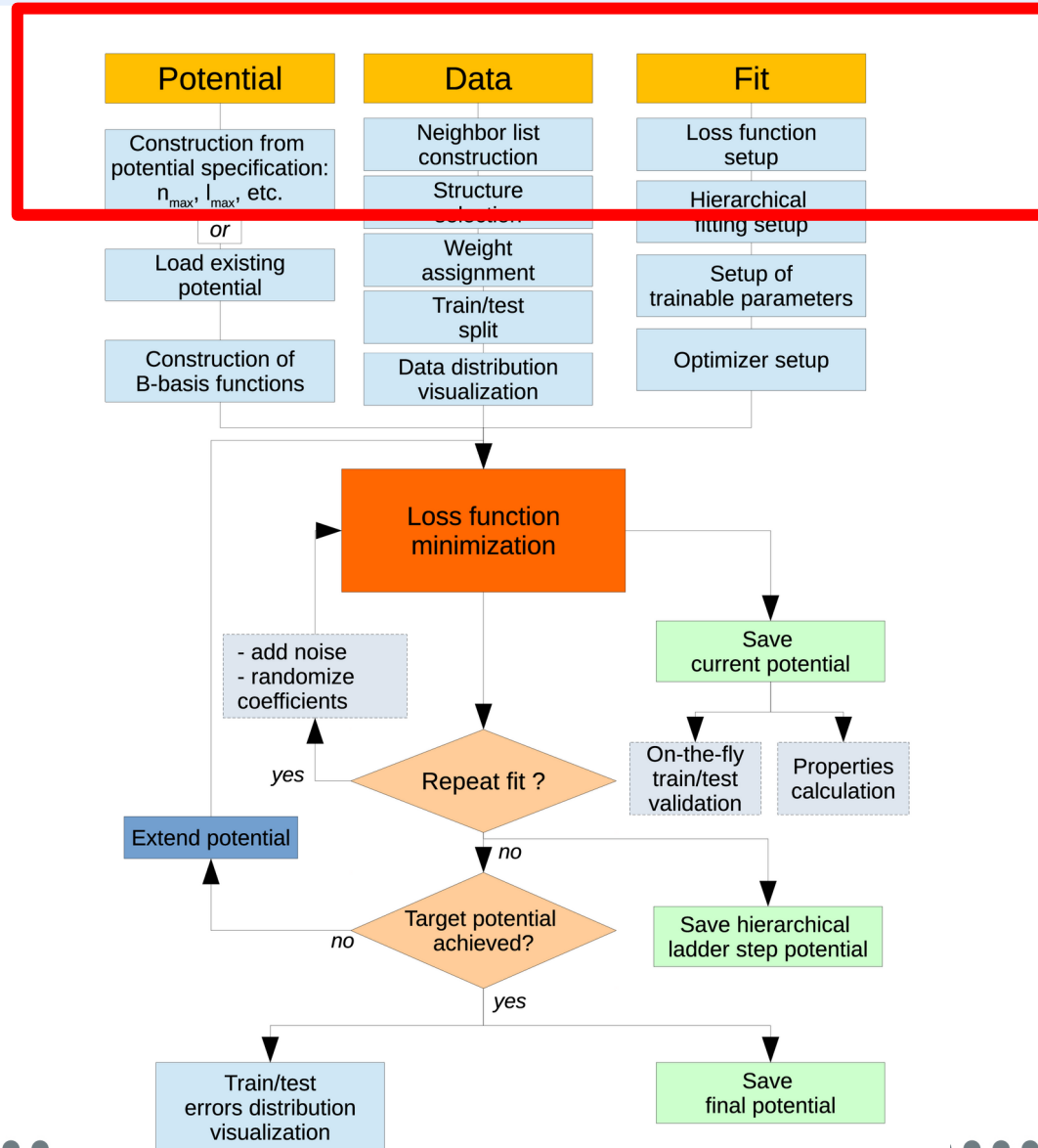
$$\mathbf{A}_{i \mu n l m} = \prod_{t=1}^{\nu} A_{i \mu_t n_t l_t m_t}$$

# Atomic cluster expansion – B-basis function

- B-function
$$B_{i\mu\mathbf{n}\mathbf{L}} = \sum_{\mathbf{m}} \begin{pmatrix} lm \\ LM \end{pmatrix} L_R = 0 A_{i\mu\mathbf{n}\mathbf{l}\mathbf{m}}$$
*rotational & inversion invariance*
- Atomic property
$$\varphi_i^{(p)} = \sum_{\mu\mathbf{n}\mathbf{L}} c_{\mu\mathbf{n}\mathbf{L}}^{(p)} B_{i\mu\mathbf{n}\mathbf{L}}$$
- Atomic energy
$$E_i = \varphi_i^{(1)} + \sqrt{\varphi_i^{(2)}}$$
*scale invariance*

## pacemaker workflow

# pacemaker workflow



## Potential setup

# Atomic cluster expansion – B-basis function

$$A_{i\mu nlm} = \prod_{t=1}^{\nu} A_{i\mu_t n_t l_t m_t}$$

$$B_{i\mu nlm} = \sum_{\mathbf{m}} \begin{pmatrix} lm \\ LM \end{pmatrix} L_R = 0 A_{i\mu nlm}$$

type: Mg Mg Mg Mg, nr: [1, 2, 2], nl: [0, 1, 1], lint: [1]

All possible/valid permutationally invariant combinations

# Atomic cluster expansion – B-basis functions

potential.yaml

```
1-order {
- {type: Mg Mg, nr: [1], nl: [0], c: [-0.38927436162521611, 0.20984551246156766]}
- {type: Mg Mg, nr: [2], nl: [0], c: [1.4094350400758542, 0.94050815622715633]}
- {type: Mg Mg, nr: [3], nl: [0], c: [2.9588990132794248, 2.1388349209791118]}
- {type: Mg Mg, nr: [4], nl: [0], c: [1.2851509007772666, 1.305175237712445]}
- {type: Mg Mg, nr: [5], nl: [0], c: [-1.653578093574122, -0.85822653649028668]}

2-order {
- {type: Mg Mg Mg, nr: [1, 1], nl: [0, 0], c: [-0.018946305647420548, 0.12680828730394764]}
- {type: Mg Mg Mg, nr: [1, 1], nl: [1, 1], c: [-0.1029318127695891, 0.02532028159817765]}
- {type: Mg Mg Mg, nr: [1, 1], nl: [2, 2], c: [0.028012606257895433, -0.0037383526619737695]}
- {type: Mg Mg Mg, nr: [1, 1], nl: [3, 3], c: [0.02084686763707377, -0.03440126264863106]}
- {type: Mg Mg Mg, nr: [1, 2], nl: [0, 0], c: [-0.046345361589086045, -0.14994621668165492]}
- {type: Mg Mg Mg, nr: [1, 2], nl: [1, 1], c: [-0.16609878751506357, -0.049233689797668574]}
- {type: Mg Mg Mg, nr: [1, 2], nl: [2, 2], c: [0.13828744540635127, -0.011442283987683413]}
- {type: Mg Mg Mg, nr: [1, 2], nl: [3, 3], c: [-0.055372900243592676, -0.040927484918434585]}
- {type: Mg Mg Mg, nr: [1, 3], nl: [0, 0], c: [0.04243842771959408, -0.60471926771396389]}
- {type: Mg Mg Mg, nr: [1, 3], nl: [1, 1], c: [0.10068839768226281, -0.17816850678190668]}
- {type: Mg Mg Mg, nr: [1, 3], nl: [2, 2], c: [-0.032123236319357241, -0.090506650246919992]}
- {type: Mg Mg Mg, nr: [1, 3], nl: [3, 3], c: [0.091615432810955202, 0.023297853506166682]}

3-order {
- {type: Mg Mg Mg Mg, nr: [1, 1, 1], nl: [0, 0, 0], lint: [0], c: [0.0041897229317814149, 0.0026578093935502112]}
- {type: Mg Mg Mg Mg, nr: [1, 1, 1], nl: [0, 1, 1], lint: [1], c: [0.018392688972029073, -0.0014558251705002606]}
- {type: Mg Mg Mg Mg, nr: [1, 1, 1], nl: [0, 2, 2], lint: [2], c: [-0.011581849261329474, -0.0032000055539503887]}
- {type: Mg Mg Mg Mg, nr: [1, 1, 1], nl: [1, 1, 2], lint: [2], c: [-0.026335649445420592, 0.055170961707895809]}
- {type: Mg Mg Mg Mg, nr: [1, 1, 1], nl: [2, 2, 2], lint: [2], c: [0.0026112158530509594, -0.003941844764793547]}
- {type: Mg Mg Mg Mg, nr: [1, 1, 2], nl: [0, 0, 0], lint: [0], c: [-0.0078783684025458406, 0.056324203669681029]}
```

Trainable coefficients

$$\varphi_i^{(p)} = \sum_{\mu \mathbf{n} \mathbf{L}} \overbrace{c_{\mu \mathbf{n} \mathbf{L}}^{(p)}}^{\text{Trainable coefficients}} B_{i\mu \mathbf{n} \mathbf{L}}$$

$$E_i = \varphi_i^{(1)} + \sqrt{\varphi_i^{(2)}}$$

Finnis-Sinclair embedding with 2 densities



# Atomic cluster expansion – potential

Three components to specify the B-basis potential

embeddings

npot: 'FinnisSinclairShiftedScaled'  
ndensity: 2  
fs\_parameters: [1, 1, 1, 0.5]

bonds

radbase: ChebExpCos  
rcut: 5

functions

nradmax\_by\_orders: [15, 3, 2, 2, 1]  
lmax\_by\_orders: [ 0, 2, 2, 1, 1]

*Specify maximum n/l for each order*

check *pacemaker* documentation for more details

## Fit setup

# Atomic cluster expansion: Computational workflow



## ACE

Radial functions

$$R_{nl}^{\mu_i \mu_j}(r_{ji}) = \sum_k c_{nlk}^{\mu_i \mu_j} g_k(r_{ji})$$

One-particle  
basis function

$$\phi_{\mu_i \mu_j n l m} = R_{nl}^{\mu_i \mu_j}(r_{ji}) Y_{lm}(\hat{\mathbf{r}}_{ji})$$

Atomic base

$$A_{i\mathbf{v}} = \sum_j \delta_{\mu \mu_j} \phi_{\mathbf{v}}(\mathbf{r}_{ji})$$

A-product

$$A_{i\mu \mathbf{n} \mathbf{l} \mathbf{m}} = \prod_{t=1}^{\nu} A_{i\mu_t n_t l_t m_t}$$

B-basis function

$$B_{i\mu \mathbf{n} \mathbf{L}} = \sum_{\mathbf{m}} \begin{pmatrix} l\mathbf{m} \\ L\mathbf{M} \end{pmatrix} \begin{matrix} L_R = 0 \\ \end{matrix} A_{i\mu \mathbf{n} \mathbf{l} \mathbf{m}}$$

Atomic property

$$\varphi_i^{(p)} = \sum_{\mu \mathbf{n} \mathbf{L}} c_{\mu \mathbf{n} \mathbf{L}}^{(p)} B_{i\mu \mathbf{n} \mathbf{L}}$$

Atomic energy

$$E_i = \mathcal{F}(\varphi_i^{(1)}, \dots, \varphi_i^{(P)})$$

**trainable parameters:**  $\Theta = \{c_{nlk}^{\mu_i \mu_j}, c_{\mu \mathbf{n} \mathbf{L}}^{(p)}\}$

Atomic configuration

$$\{\mathbf{r}_{ij}\}$$

Interatomic potential

$$E_i = \mathcal{F}(\{\mathbf{r}_{ij}\}, \Theta)$$

Energy

$$E = \sum_i E_i$$

Force

$$\mathbf{F}_i = -\nabla_i E$$

Loss function

$$\mathcal{L} = \sum_n (E_n - E_n^{\text{DFT}})^2 + \sum_{n,i} (\mathbf{F}_{ni} - \mathbf{F}_{ni}^{\text{DFT}})^2$$

$$\frac{\partial \mathcal{L}}{\partial \Theta}$$

**Minimizer**

automatic gradients

A Bochkarev, Y Lysogorskiy, S Menon, M Qamar, M Mrovec, R Drautz Physical Review Materials 6 (1), 013804

# Loss function

fitting = minimization of loss function

$$\begin{aligned}\mathcal{L} = & (1 - \kappa) \sum_{n=1}^{N_{\text{struct}}} w_n^{(E)} \left( \frac{E_n^{\text{ACE}} - E_n^{\text{ref}}}{n_{\text{at},n}} \right)^2 & \left. \vphantom{\sum_{n=1}^{N_{\text{struct}}}} \right\} & \text{Energy loss} \\ & + \kappa \sum_{n=1}^{N_{\text{struct}}} \sum_{i=1}^{n_{\text{at},n}} w_{ni}^{(F)} (\mathbf{F}_{ni}^{\text{ACE}} - \mathbf{F}_{ni}^{\text{ref}})^2 & \left. \vphantom{\sum_{n=1}^{N_{\text{struct}}}} \right\} & \text{Force loss} \\ & + \Delta_{\text{coeff}} + \Delta_{\text{rad}}, & \left. \vphantom{\sum_{n=1}^{N_{\text{struct}}}} \right\} & \text{Regularization}\end{aligned}$$

## Loss function – energy/force contribution

fitting = minimization of loss function

$$\begin{aligned} \mathcal{L} = & (1 - \kappa) \sum_{n=1}^{N_{\text{struct}}} w_n^{(E)} \left( \frac{E_n^{\text{ACE}} - E_n^{\text{ref}}}{n_{\text{at},n}} \right)^2 \\ & + \kappa \sum_{n=1}^{N_{\text{struct}}} \sum_{i=1}^{n_{\text{at},n}} w_{ni}^{(F)} \left( \mathbf{F}_{ni}^{\text{ACE}} - \mathbf{F}_{ni}^{\text{ref}} \right)^2 \\ & + \Delta_{\text{coeff}} + \Delta_{\text{rad}}, \end{aligned}$$

relative force contribution:

- $\kappa = 0$ : energy-only fit
- $\kappa = 1$ : force-only fit

## Loss function - weights

fitting = minimization of loss function

$$\begin{aligned} \mathcal{L} = & (1 - \kappa) \sum_{n=1}^{N_{\text{struct}}} w_n^{(E)} \left( \frac{E_n^{\text{ACE}} - E_n^{\text{ref}}}{n_{\text{at},n}} \right)^2 \\ & + \kappa \sum_{n=1}^{N_{\text{struct}}} \sum_{i=1}^{n_{\text{at},n}} w_{ni}^{(F)} (\mathbf{F}_{ni}^{\text{ACE}} - \mathbf{F}_{ni}^{\text{ref}})^2 \\ & + \Delta_{\text{coeff}} + \Delta_{\text{rad}}, \end{aligned}$$

## Energies per-structure and forces per-atom weights:

- uniform weights
- energy-based weights
- custom weights

## Loss function - regularization

fitting = minimization of loss function

$$\begin{aligned} \mathcal{L} = & (1 - \kappa) \sum_{n=1}^{N_{\text{struct}}} w_n^{(E)} \left( \frac{E_n^{\text{ACE}} - E_n^{\text{ref}}}{n_{\text{at},n}} \right)^2 \\ & + \kappa \sum_{n=1}^{N_{\text{struct}}} \sum_{i=1}^{n_{\text{at},n}} w_{ni}^{(F)} \left( \mathbf{F}_{ni}^{\text{ACE}} - \mathbf{F}_{ni}^{\text{ref}} \right)^2 \\ & + \Delta_{\text{coeff}} + \Delta_{\text{rad}}, \end{aligned}$$

## L1/L2 regularization

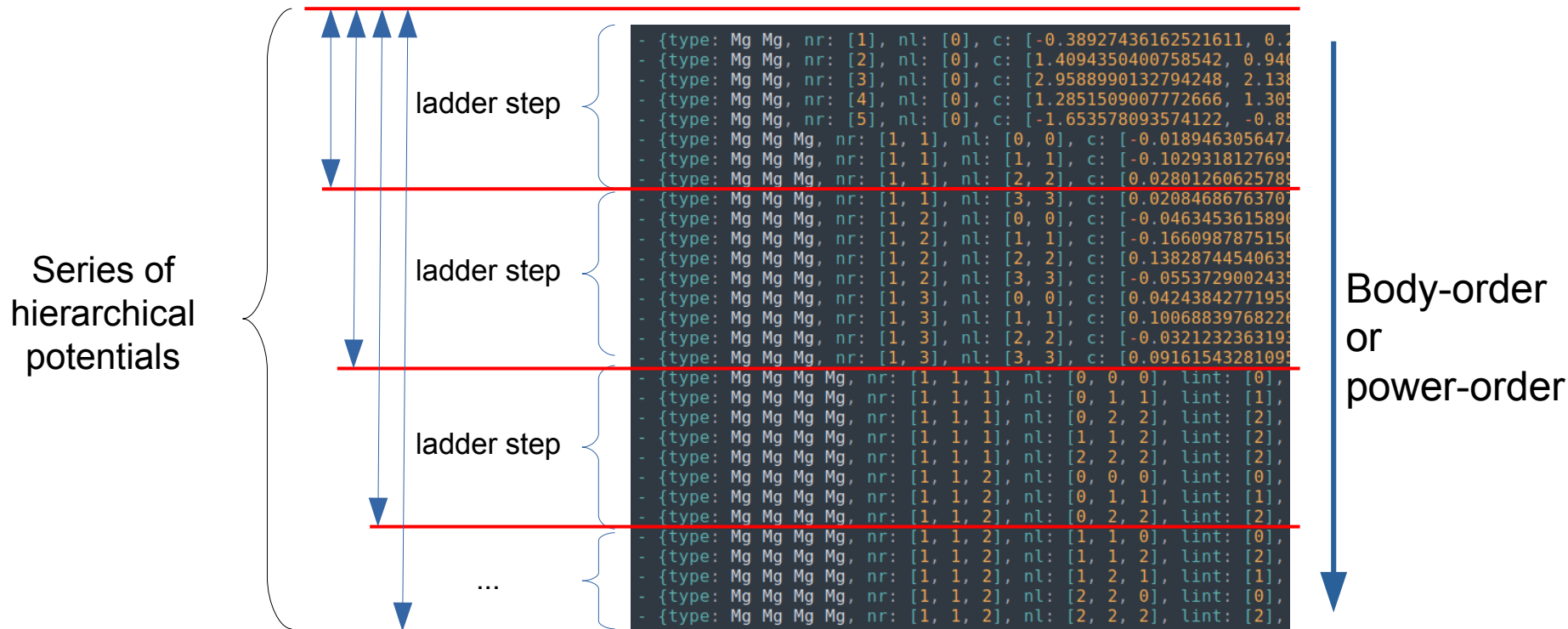
$$\Delta_{\text{coeff}} = L_1 \sum_{p\mu\mathbf{n}\mathbf{L}} \left| c_{\mu\mathbf{n}\mathbf{L}}^{(p)} \right| + L_2 \sum_{p\mu\mathbf{n}\mathbf{L}} \left| c_{\mu\mathbf{n}\mathbf{L}}^{(p)} \right|^2$$

# Hierarchical basis extension ( = LADDER fit)

$$E_i = \varphi_i^{(1)} + \sqrt{\varphi_i^{(2)}}$$

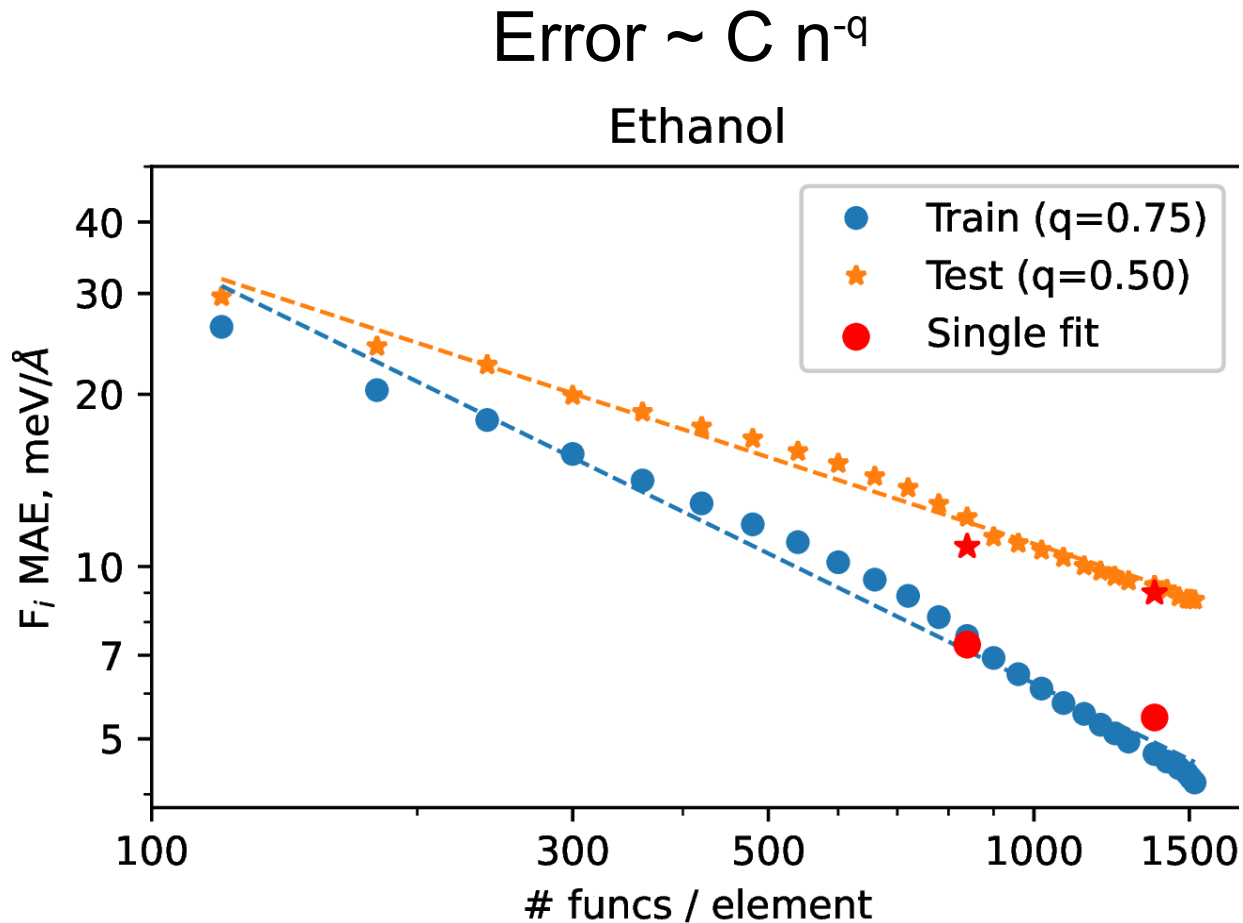
$$\varphi_i^{(p)} = \sum_{\mu \mathbf{n} \mathbf{L}} c_{\mu \mathbf{n} \mathbf{L}}^{(p)} B_{i \mu \mathbf{n} \mathbf{L}}$$

*new B-functions could be seamlessly added with 0-coeffs*





# Hierarchical basis extension: convergence exponent



## Data setup

## DFT reference data

## Where from get the DFT data?

## Find dataset in Internet

Generate by yourself

# DFT reference data: from Internet

Where from get the DFT data?

Find dataset in Internet

Generate by yourself

!! check Supplementary Materials / Data Availability/ etc. sections of papers !!

Examples: *archive.materialscloud.org*

## Revised MD17 dataset

### Files

File name	Size	Description
<a href="#">rmd17.tar.bz2</a> <small>MD5</small>	1016.9 MiB	Tarfile containing the data in NPZ and CSV format
<a href="#">readme.txt</a> <small>MD5</small>	2.0 KiB	Readme file

## CA-9, a dataset of carbon allotropes for training and testing of neural network potentials

### Files

File name	Size	Description
<a href="#">Readme.txt</a> <small>MD5</small>	2.4 KiB	Readme file
<a href="#">scripts.zip</a> <small>MD5</small>	3.4 KiB	Python scripts used to read data from VACP and train neural network potentials
<a href="#">datasets.zip</a> <small>MD5</small>	453.4 MiB	Datasets for training and testing of neural network potentials
<a href="#">NNPs.zip</a> <small>MD5</small>	13.1 MiB	The best trained neural network potentials for each dataset

# DFT reference data: generate by yourself

Where from get the DFT data?

Find dataset in Internet

Generate by yourself

Any (high-throughput) DFT calculation solutions:

- BASH scripts
- Python/ASE
- pyiron
- etc.

**IMPORTANT!**

**energy\_corrected = DFT energy – N<sub>(at)</sub> \* DFT free atom(s) energy**



Special columns names:

	ase_atoms	energy_corrected	forces
0	(Atom('Ti', [3.866588, 2.3427139999999995, 2.3...	-2.098939	[[ -0.0809379287038194, -0.296866532922296, -0....
1	(Atom('Ti', [0.01561, 0.047543, 0.016239], ind...	-45.733959	[[ -0.189867494784043, -0.327421600829841, -0.3...
2	(Atom('Ti', [5.5931969999999999, 0.002012000000...	-104.175058	[[ 0.15376244749841, -0.067939845038046, -0.195...
3	(Atom('Ti', [0.0, 0.0, 0.0], index=0), Atom('T...	-8.891530	[[ -2.18770164697502e-10, -3.05044330444982e-09...
4	(Atom('Ti', [0.0, 0.0, 0.0], index=0))	-3.032699	[[ 0.0, 0.0, 0.0]]

ASE atoms

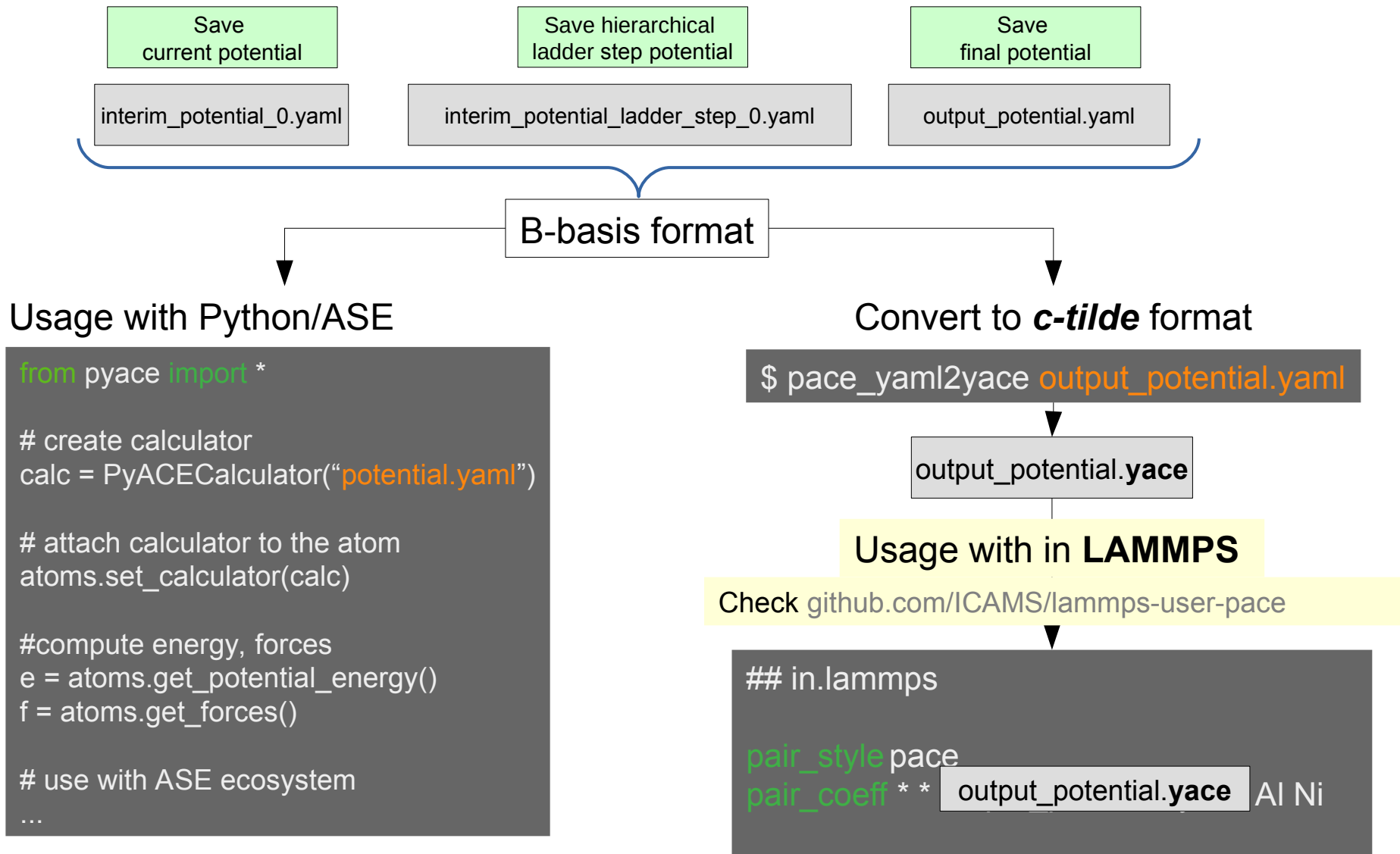
Cohesive energy (eV)

Forces (eV/Å): [n\_at, 3]

Save to file as: `df.to_pickle("my_dataset.pckl.gzip", compression="gzip", protocol=4)`

## Usage of the potential

# Usage of the potential



# PACE: LAMMPS implementation performance

- C++ implementation for CPU

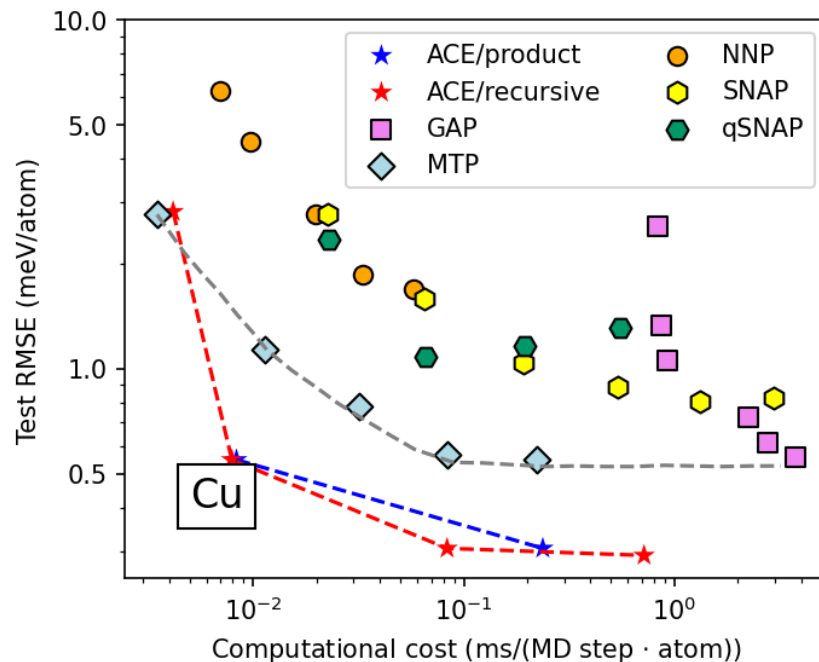
*by ICAMS team and C. Ortner (UBC Math)*

**100-500 microsec / atom / CPU core**

- KOKKOS implementation for GPU

*by Stan Moore (SNL), with helpful discussions with Evan Weinberg (NVIDIA) and Yury Lysogorskiy (ICAMS)*

**4-20 microsec / atom/ GPU (x30 faster)**

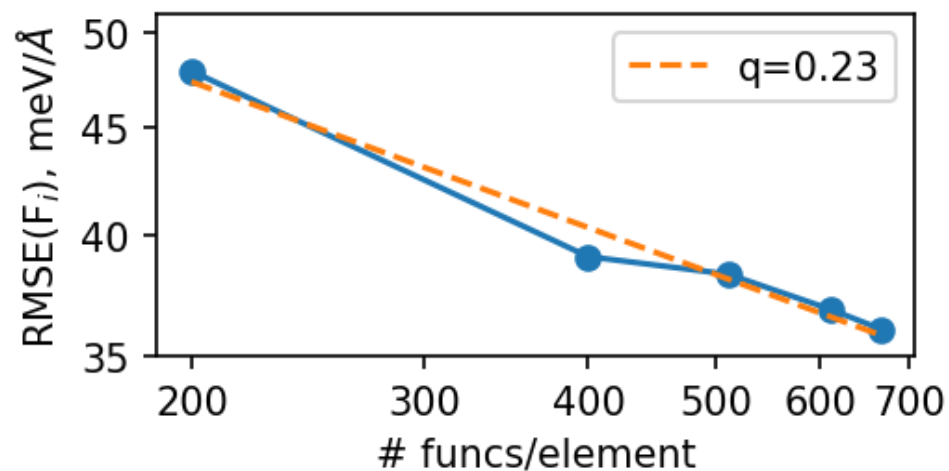
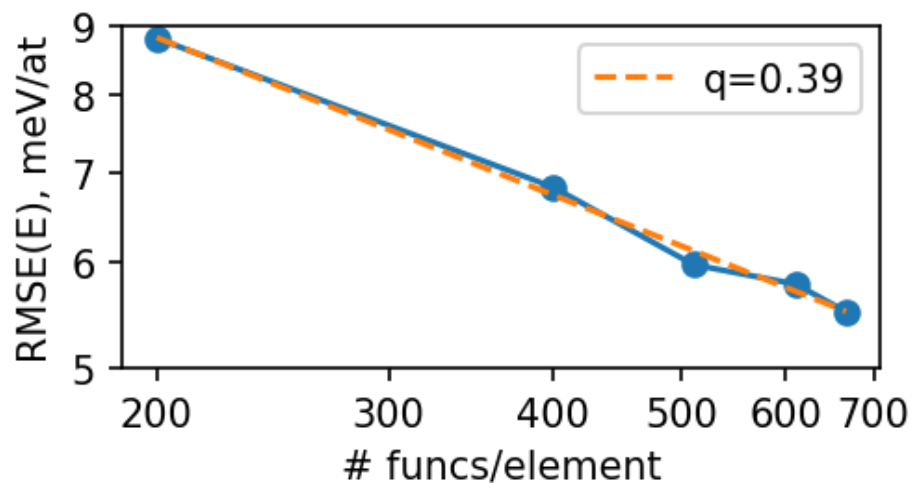




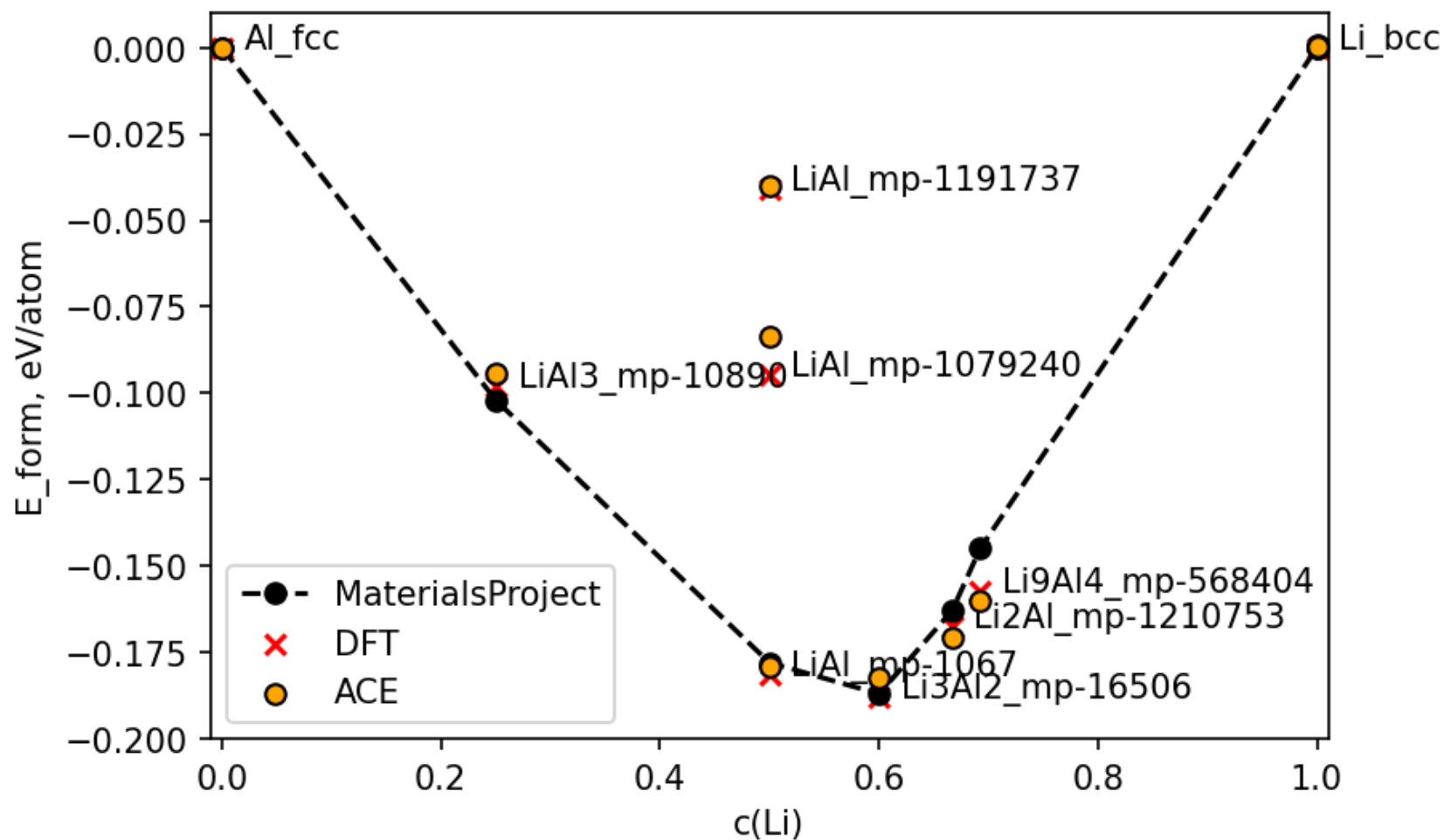
## Al-Li potential fit: details

# Al-Li potential validation: feature curves

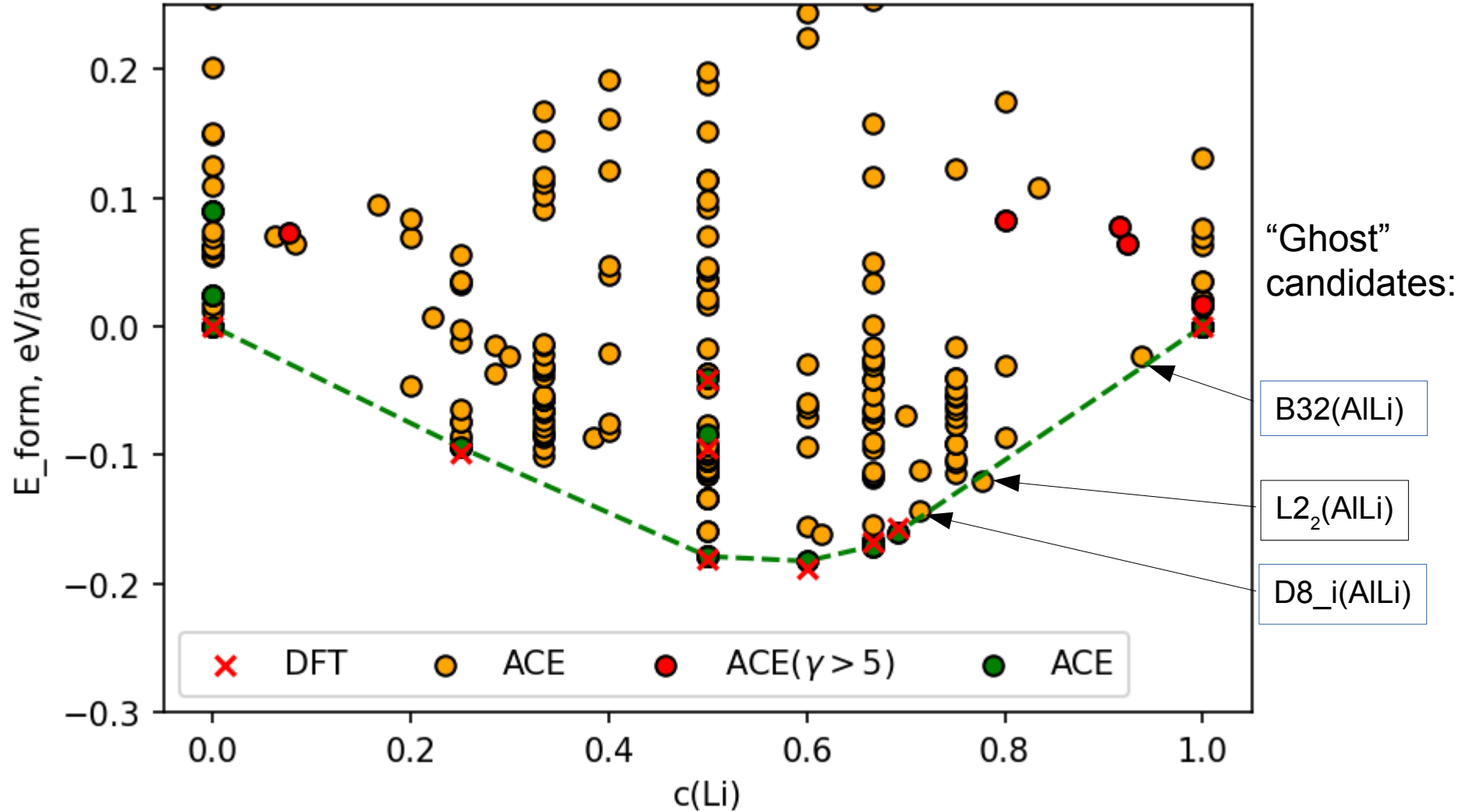
$$\text{Error} \sim C n^{-q}$$



# Al-Li potential validation: Convex hull



# Al-Li potential validation: Convex hull (more prototypes)



**Thank you for attention**

# PACE: Product and recursive evaluator

## Product evaluator:

$$\rho = \underbrace{c_1 A_1 + c_2 A_2 + \dots + c_n A_n}_{1\text{-order}} + \underbrace{c_{12} A_1 A_2 + c_{13} A_1 A_3 + \dots}_{2\text{-order}} + \underbrace{c_{123} A_1 A_2 A_3 + \dots}_{3\text{-order}} + \underbrace{c_{1234} A_1 A_2 A_3 A_4 + \dots}_{4\text{-order}} + \dots$$

## Recursive evaluator: (by C. Ortner)

