

MlipKit

a python wrapper for MLIPs

Samuel Longo - University of Liège, BE
samuel.longo@uliege.be

CoDTS - 14/10/2025 - Rennes (remote talk)

DISCLAIMER



Motivation

MLIPs have different
implementations
(command line, python)

Python implementations
are not uniform

It's hard to interface to
other programs (MD,
phony, etc...)



Lots of scripts

Hard to maintain

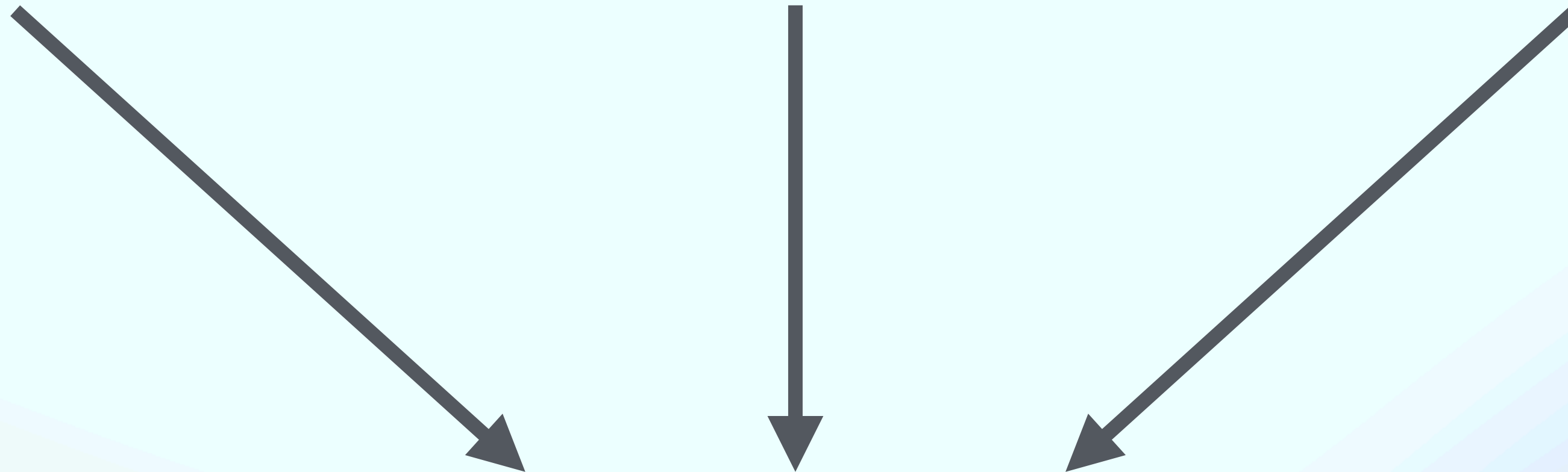
Inconsistencies

Solution

Wrap specific in general

Storable object with
fixed properties and
functions

Single interface to
external objects



MlipKit

github.com/chewinggram/mlipkit (soon)

MlipKit: the object

MlipModel object

- Training hyperparameters
- Training set
- Trained potential
- Parameters for prediction

train_model()

compute_properties()

evaluate()

save()

load()

Hash-based saving/
loading system to ensure
consistency

MlipKit: implementing a model

NewModel(MlipModel)

1. Mandatory and optional training hyperparameters

2. Mandatory and optional parameters for computing properties

3. List of computable properties

4. Files needed to define a trained version of the model

5. Methods to train, fine tune, compute properties etc...

```
mandatory_hyperparameters_names = ['mlip_bin',  
                                   'untrained_pot_file_dir',  
                                   'mtp_level',  
                                   'min_dist',  
                                   'max_dist',  
                                   'radial_basis_type',  
                                   'radial_basis_size']
```

```
optional_hyperparameters_names = ['ene_weight',  
                                  'for_weight',  
                                  'str_weight',  
                                  'sc_b_for',  
                                  'val_cfg',  
                                  'max_iter',  
                                  'cur_pot_n',  
                                  'bfgs_tol',  
                                  'weighting',  
                                  'init_par',  
                                  'skip_preinit',  
                                  'up_mindist',  
                                  'bin_pref']
```

MlipKit: implementing a model

NewModel(MlipModel)

1. Mandatory and optional training hyperparameters

2. Mandatory and optional parameters for computing properties

3. List of computable properties

4. Files needed to define a trained version of the model

5. Methods to train, fine tune, compute properties etc...

```
mandatory_parameters_compute_properties_names = ['mlip_bin']
```

```
optional_parameters_compute_properties_names = ['bin_pref']
```

MlipKit: implementing a model

NewModel(MlipModel)

1. Mandatory and optional training hyperparameters
2. Mandatory and optional parameters for computing properties

3. List of computable properties

4. Files needed to define a trained version of the model
5. Methods to train, fine tune, compute properties etc...

```
computable_properties_names = ['energy',  
                               'forces',  
                               'stress']
```

MlipKit: implementing a model

NewModel(MlipModel)

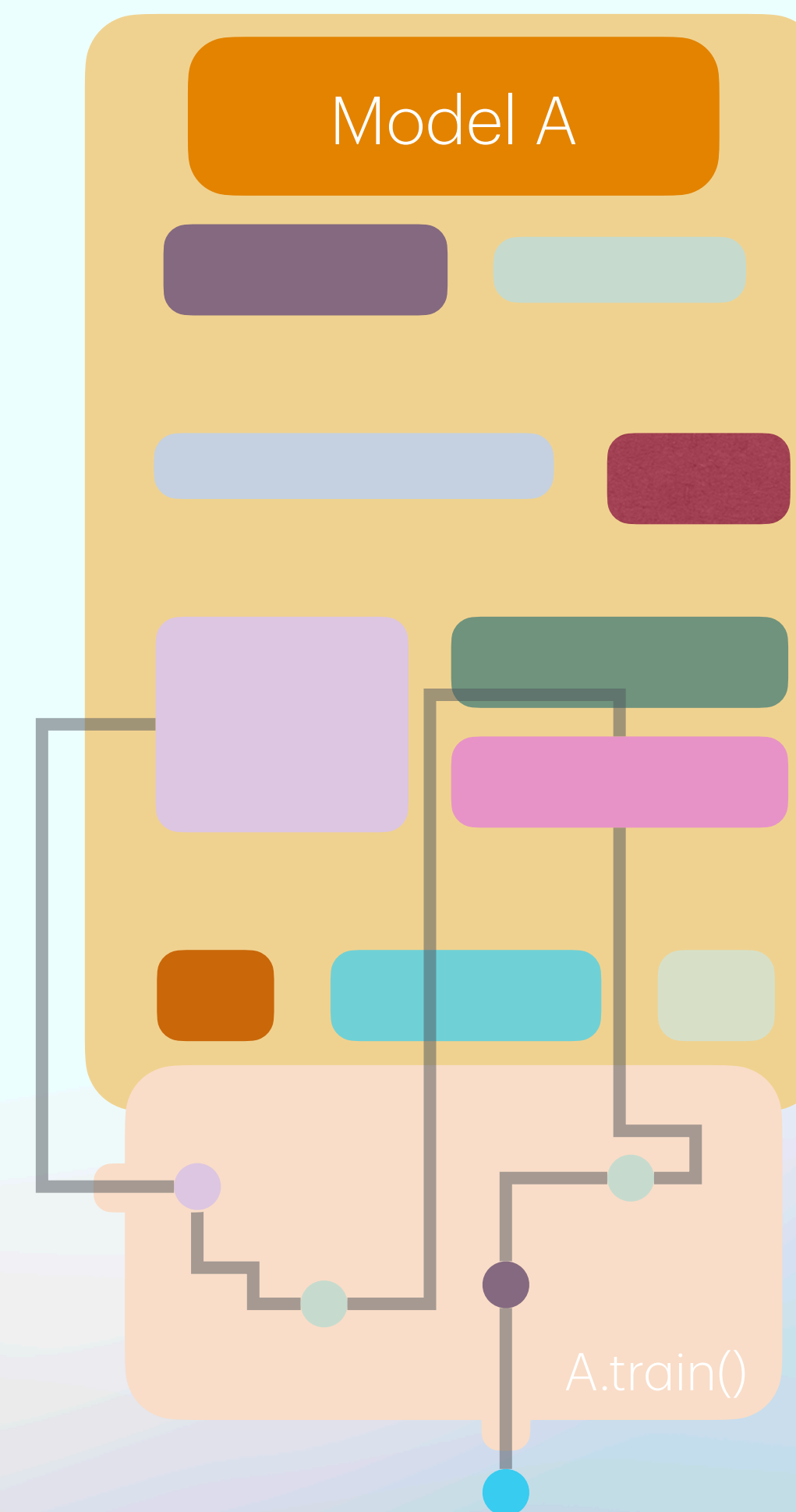
1. Mandatory and optional training hyperparameters
2. Mandatory and optional parameters for computing properties
3. List of computable properties
- 4. Files needed to define a trained version of the model**
5. Methods to train, fine tune, compute properties etc...

```
trained_pot_files = {'potential_file' : 'pot.mtp'}
```

MlipKit: implementing a model

`NewModel(MlipModel)`

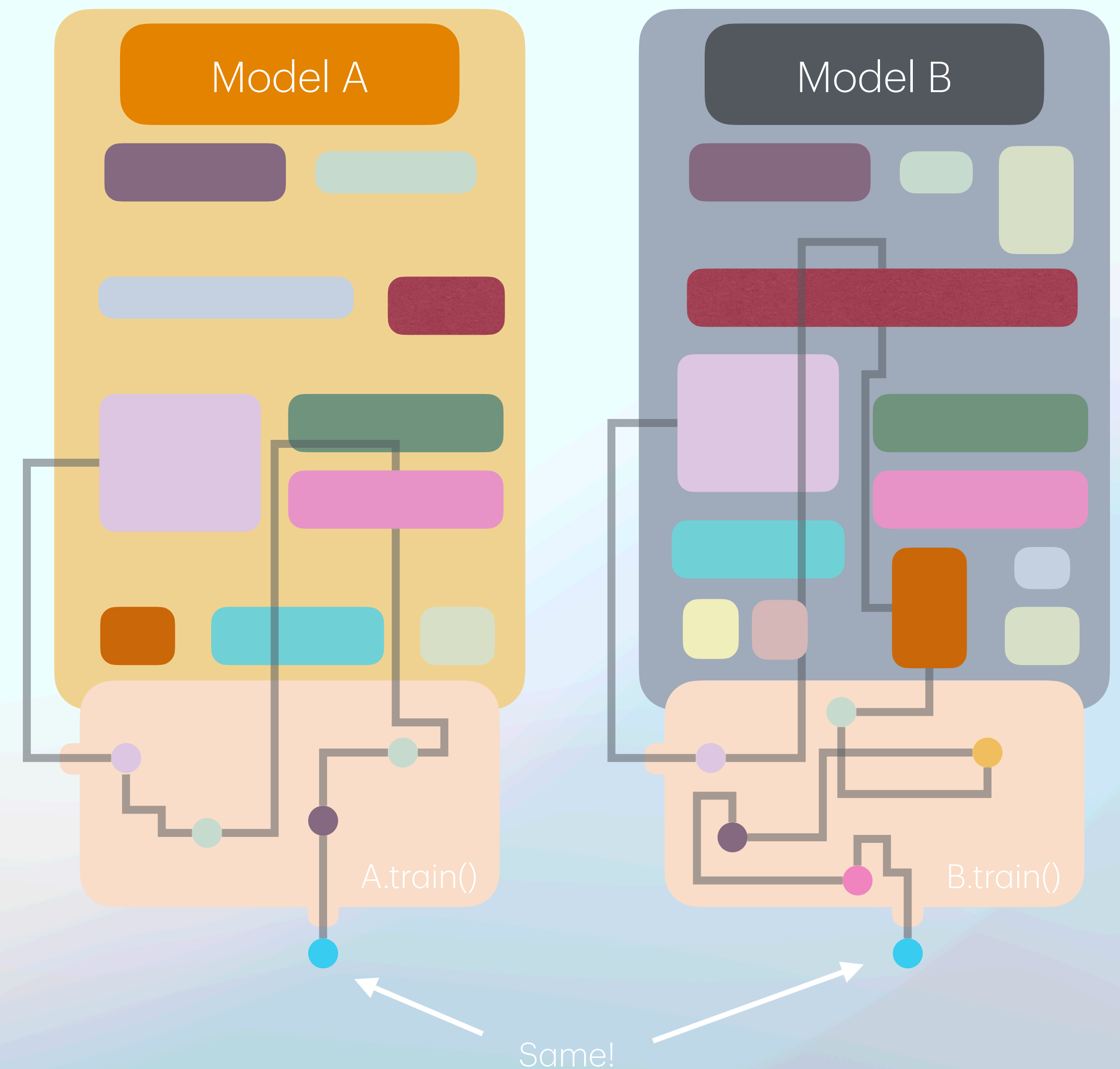
1. Mandatory and optional training hyperparameters
2. Mandatory and optional parameters for computing properties
3. List of computable properties
4. Files needed to define a trained version of the model
- 5. Methods to train, fine tune, compute properties etc...**



MlipKit: implementing a model

NewModel(MlipModel)

1. Mandatory and optional training hyperparameters
2. Mandatory and optional parameters for computing properties
3. List of computable properties
4. Files needed to define a trained version of the model
- 5. Methods to train, fine tune, compute properties etc...**



Hands on!

Example: sTDEP

Stochastic Temperature-Dependent Effective Potential: anharmonic phonons

