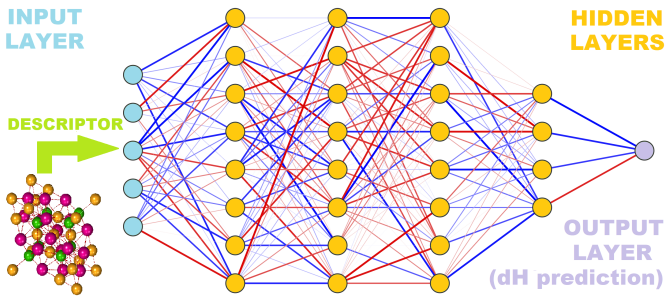
pySIPFENN MGF-PSU Workshop (Feb 2023)

This Jupyter notebook is a brief walkthrough covering core functionalities of the **pySIPFENN** or **py**(**S**tructure-**I**nformed **P**rediction of **F**ormation **E**nergy using **N**eural **N**etworks) package; available through the PyPI repository. A fuller description of capabilities is given at PSU Phases Research Lab webpage under *phaseslab.com/sipfenn*.



Install pySIPFENN

Installing pySIPFENN is simple and easy utilizing either **PyPI** package repository or cloning from **GitHub**. While not required, it is recommended to first set up a virtual environment using venv or Conda. This ensures that one of the required versions of Python (3.9+) is used and there are no dependency conflicts. To create one

```
conda create -n pysipfenn-workshop python=3.9 jupyter
conda activate pysipfenn-workshop
```

And then install pySIPFENN from PyPI with

```
pip install pysipfenn
```

Alternatively, you can also install pySIPFENN in editable mode if you cloned it from GitHub like

```
git clone https://github.com/PhasesResearchLab/pySIPFENN.git
```

Or by downloading a ZIP file. Then, move to the pySIPFENN folder and install in editable (-e) mode

```
cd pySIPFENN
pip install -e .
```

Starting pySIPFENN

To utilize pySIPFENN for straightforward calculations, only the Calculator class is needed. It allows for both fetching and identification of NN models and later running of them. For pretty printing we also import pretty print Python built-in library.

In [24]:
```
from pysipfenn import Calculator
from pprint import pprint # Just for pretty printing
```

Now initialize the Calculator. When run, this should display all models detected (e.g. ✓ SIPFENN_Krajewski2020 Standard Materials Model) and those not detected, but declared in the *modelsSIPFENN/models.json* file.

In [25]:
```
c = Calculator()
```

```
✓ SIPFENN_Krajewski2020 Standard Materials Model
✓ SIPFENN_Krajewski2020 Novel Materials Model
✓ SIPFENN_Krajewski2020 Light Model
✓ SIPFENN_Krajewski2022 KS2022 Novel Materials Model
Loading models:
100%|██████████| 4/4 [00:17<00:00,  4.30s/it]
*********  PySIPFENN Successfully Initialized  **********
```

If this is the first run of pySIPFENN and no models are available, one can fetch four default (as of Feb 2023) models from Zenodo with a simple:

In [26]:
```
#c.downloadModels()
#c.loadModels()
```

For the purpose of testing, a single model is sufficient and will be fetched faster. E.g. the lightweight model ('SIPFENN_Krajewski2020_NN24') can be acquired in about 1/30 of the time required to download all four.

In [27]:
```
#c.downloadModels(network='SIPFENN_Krajewski2020_NN24')
#c.loadModels()
```

Simple run from directory

The simplest and most common usage of pySIPFENN is to deploy it on a directory/folder containing atomic structure files such as POSCAR or CIF. To of so, one simply specifies its location and which descriptor / feature vector should be used. The latter determines which ML models will be run, as they require a list of specific and ordered features as input. Furthermore, while the exact model can be specified by the user, by default all applicable models are run, as the run itself is 1-3 orders of magnitude faster than descriptor calculation.

```
c.runFromDirectory(directory='myInputFiles', descriptor='KS2022')
```

In this demonstration, a set of test files shipped with pySIPFENN under **directory**

```
pysipfenn/tests/testCaseFiles/exampleInputFiles/
```

is used. However, feel free to change the directory to something with your structure files. Please note that the file extension (e.g. *.POSCAR*) is required for correct input.

Furthermore, you can specify whether pySIPFENN should run in series or parallel **calculation mode**. The parallel mode is generally faster, but uses more system resources and may be slower on low-power machines with not enough CPU cores. Serial mode is also preferred if there are less than 5 calculations/worker due to multiprocessing overheads.

In [28]:
```
c.runFromDirectory(directory='../pysipfenn/tests/testCaseFiles/exampleInputFiles/',
                   descriptor='KS2022',
                   mode='serial')
```

```
Importing structures...
100%|██████████| 32/32 [00:00<00:00, 571.42it/s]

Models that will be run: ['SIPFENN_Krajewski2022_NN30']
Calculating descriptors...
100%|██████████| 32/32 [00:10<00:00,  3.07it/s]
Done!
Making predictions...
Prediction rate: 183.2 pred/s
Obtained 32 predictions from:  SIPFENN_Krajewski2022_NN30
Done!
```

Now, all results are obtained and stored within the **c** Calculator object inside a few exposed conveniently named variables *predictions* and *inputFiles*. Also, the descriptor data is retained in *descriptorData* if needed. Let's look up the first 3 entries.

In [29]:
```
pprint(c.inputFiles[:3])
pprint(c.predictions[:3])
```

```
['0-Cr8Fe18Ni4.POSCAR', '1-Cr16Fe8Ni6.POSCAR', '2-Fe8Ni22.POSCAR']
[[0.17857088148593903], [0.22403044998645782], [0.07981749624013901]]
```

For user convenience, a few methods are provided for extracting the results. E.g., if pySIPFENN has been run from structure files, the _get_resultDictsWithNames()_ method is available to conveniently pass results forward in the code.

In [30]:
```
c.get_resultDictsWithNames()[:3]
```

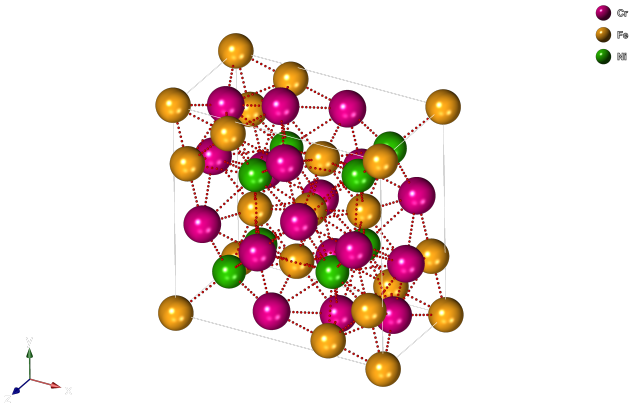
Out[30]:
```
[{'name': '0-Cr8Fe18Ni4.POSCAR',
  'SIPFENN_Krajewski2022_NN30': 0.17857088148593903},
 {'name': '1-Cr16Fe8Ni6.POSCAR',
  'SIPFENN_Krajewski2022_NN30': 0.22403044998645782},
 {'name': '2-Fe8Ni22.POSCAR',
  'SIPFENN_Krajewski2022_NN30': 0.07981749624013901}]
```

Alternatively, if results are to be preserved in a spreadsheet, they can be exported into a CSV.

In [31]:
```
c.writeResultsToCSV('myFirstResults_pySIPFENN.csv')
```

Sigma-Phase 5-sublattice model

In the previous example we went over a set of POSCAR files in a directory without performing any non-automated manipulation. This is what authors expect will be the most common use pattern. However, pySIPFENN can be effortlessly combined with other structure analysis and manipulation software to fulfill more advanced needs. Here, as an example, we will play with a fairly complex **topologically close packed (TCP) phase called Sigma** possesing 5 chemically unique sites, which will be automatically identified from one configuration, like the one in figure below. Then we will look at energetics of all possible occupancies by 3 elements.



For structure manipulation we will utilize pymatgen Structure. The Spglib library (spglib.github.io/spglib), accessed through pymatgen, will perform the symmetry analysis. We begin with import of appropriate modules.

In [32]:
```
from pymatgen.core import Structure
from pymatgen.analysis.structure_analyzer import SpacegroupAnalyzer
```

Next, we take any end-member of the atomic structure in question. Conveniently, one of them is already in the test case files. Then we replace all species in it with dummy ones.

In [33]:
```
baseStructure = Structure.from_file('../pysipfenn/tests/testCaseFiles/exampleInputFiles/0-Cr8Fe18Ni4.POSCAR')
for el in set(baseStructure.species):
    baseStructure.replace_species({el: 'dummy'})
print(baseStructure)
```

```
Full Formula (Dummy30)
Reduced Formula: Dummy
abc   :   8.547048   8.547048   4.477714
angles:  90.000000  90.000000  90.000000
pbc   :       True       True       True
Sites (30)
  #  SP              a         b         c
---  -------  --------  --------  --------
  0  Dummy0+  0.737702  0.063709  0
  1  Dummy0+  0.262298  0.936291  0
  2  Dummy0+  0.436291  0.237702  0.5
  3  Dummy0+  0.762298  0.563709  0.5
  4  Dummy0+  0.563709  0.762298  0.5
  5  Dummy0+  0.237702  0.436291  0.5
  6  Dummy0+  0.063709  0.737702  0
  7  Dummy0+  0.936291  0.262298  0
  8  Dummy0+  0         0         0
  9  Dummy0+  0.5       0.5       0.5
 10  Dummy0+  0.463029  0.129472  0
 11  Dummy0+  0.536971  0.870528  0
 12  Dummy0+  0.370528  0.963029  0.5
 13  Dummy0+  0.036971  0.629472  0.5
 14  Dummy0+  0.629472  0.036971  0.5
 15  Dummy0+  0.963029  0.370528  0.5
 16  Dummy0+  0.129472  0.463029  0
 17  Dummy0+  0.870528  0.536971  0
 18  Dummy0+  0.182718  0.182718  0.251726
 19  Dummy0+  0.817282  0.817282  0.748274
 20  Dummy0+  0.817282  0.817282  0.251726
 21  Dummy0+  0.317282  0.682718  0.751726
 22  Dummy0+  0.317282  0.682718  0.248274
 23  Dummy0+  0.182718  0.182718  0.748274
 24  Dummy0+  0.682718  0.317282  0.248274
 25  Dummy0+  0.682718  0.317282  0.751726
 26  Dummy0+  0.39991   0.39991   0
 27  Dummy0+  0.60009   0.60009   0
 28  Dummy0+  0.10009   0.89991   0.5
 29  Dummy0+  0.89991   0.10009   0.5
```

Then we use Spglib to find and group unique chemical sites in the structure and lists of their equivalents. In the case of Sigma-phase, there are 5 such sites, also called **sublattices**.

In [34]:
```
spgA = SpacegroupAnalyzer(baseStructure)
```

In [35]:
```
eqAtoms = spgA.get_symmetry_dataset()['equivalent_atoms']
uniqueDict = {}
for site, unique in enumerate(eqAtoms):
    if unique not in uniqueDict:
        uniqueDict.update({unique: []})
    uniqueDict[unique] += [site]
pprint(uniqueDict)
```

```
{0: [0, 1, 2, 3, 4, 5, 6, 7],
 8: [8, 9],
 10: [10, 11, 12, 13, 14, 15, 16, 17],
 18: [18, 19, 20, 21, 22, 23, 24, 25],
 26: [26, 27, 28, 29]}
```

Now, with unique sites identified, we need to find all possible occupancies in the chemical system in question. Here we look at the Cr-Fe-Ni ternary. We expect 3^5=243 possible permutations with repetition.

In [36]:
```
from itertools import product
allPermutations = list(product(['Fe', 'Cr', 'Ni'], repeat=5))
print(f'Obtained {len(allPermutations)} permutations of the sublattice occupancy\nE.g.:  {allPermutations[32]}')
```

```
Obtained 243 permutations of the sublattice occupancy
E.g.:  ('Fe', 'Cr', 'Fe', 'Cr', 'Ni')
```

In [37]:
```
structList = []
for permutation in allPermutations:
    tempStructure = baseStructure.copy()
    for unique, el in zip(uniqueDict, permutation):
        for site in uniqueDict[unique]:
            tempStructure.replace(site, el)
    structList.append(tempStructure)
print(structList[32])
```

```
Full Formula (Cr10 Fe16 Ni4)
Reduced Formula: Cr5(Fe4Ni)2
abc   :   8.547048   8.547048   4.477714
angles:  90.000000  90.000000  90.000000
pbc   :       True       True       True
Sites (30)
  #  SP           a         b         c
---  ----  --------  --------  --------
  0  Fe    0.737702  0.063709  0
  1  Fe    0.262298  0.936291  0
  2  Fe    0.436291  0.237702  0.5
  3  Fe    0.762298  0.563709  0.5
  4  Fe    0.563709  0.762298  0.5
  5  Fe    0.237702  0.436291  0.5
  6  Fe    0.063709  0.737702  0
  7  Fe    0.936291  0.262298  0
  8  Cr    0         0         0
  9  Cr    0.5       0.5       0.5
 10  Fe    0.463029  0.129472  0
 11  Fe    0.536971  0.870528  0
 12  Fe    0.370528  0.963029  0.5
 13  Fe    0.036971  0.629472  0.5
 14  Fe    0.629472  0.036971  0.5
 15  Fe    0.963029  0.370528  0.5
 16  Fe    0.129472  0.463029  0
 17  Fe    0.870528  0.536971  0
 18  Cr    0.182718  0.182718  0.251726
 19  Cr    0.817282  0.817282  0.748274
 20  Cr    0.817282  0.817282  0.251726
 21  Cr    0.317282  0.682718  0.751726
 22  Cr    0.317282  0.682718  0.248274
 23  Cr    0.182718  0.182718  0.748274
 24  Cr    0.682718  0.317282  0.248274
 25  Cr    0.682718  0.317282  0.751726
 26  Ni    0.39991   0.39991   0
 27  Ni    0.60009   0.60009   0
 28  Ni    0.10009   0.89991   0.5
 29  Ni    0.89991   0.10009   0.5
```

In [38]:
```
c = Calculator()
```

```
✓ SIPFENN_Krajewski2020 Standard Materials Model
✓ SIPFENN_Krajewski2020 Novel Materials Model
✓ SIPFENN_Krajewski2020 Light Model
✓ SIPFENN_Krajewski2022 KS2022 Novel Materials Model
Loading models:

100%|██████████| 4/4 [00:14<00:00,  3.50s/it]

*********  PySIPFENN Successfully Initialized  **********
```

In [39]:
```
predictions1 = c.runModels(structList=structList, descriptor='KS2022', mode='parallel', max_workers=6)
pprint(predictions1[:3])
```

```
Models that will be run: ['SIPFENN_Krajewski2022_NN30']
Calculating descriptors...

  0%|          | 0/243 [00:00<?, ?it/s]

Done!
Making predictions...
Prediction rate: 210.3 pred/s
Obtained 243 predictions from:  SIPFENN_Krajewski2022_NN30
[[0.17875494062900543], [0.164070263504982], [0.20459170639514923]]
```

In [40]:
```
results1 = c.get_resultDicts()
pprint(results1[:3])
```

```
[{'SIPFENN_Krajewski2022_NN30': 0.17875494062900543},
 {'SIPFENN_Krajewski2022_NN30': 0.164070263504982},
 {'SIPFENN_Krajewski2022_NN30': 0.20459170639514923}]
```

Run models utilizing different descriptor / feature vector

As alluded to before, pySIPFENN is built to enable rapid deployment of models based around single feature vector (or a subset of it) as descriptor calculation is by far the most costly component of structure-informed ML. However, the same feature vector cannot always be used and sometimes another one is needed. In this part of the tutorial, we combine previous predictions with a models based on a different descriptor (KS2017) used in the original SIPFENN paper.

In [41]:
```
predictions2 = c.runModels(structList=structList, descriptor='Ward2017', mode='parallel', max_workers=6)
pprint(predictions2[:3])
```

```
Models that will be run: ['SIPFENN_Krajewski2020_NN9', 'SIPFENN_Krajewski2020_NN20', 'SIPFENN_Krajewski2020_NN24']
Calculating descriptors...

  0%|          | 0/243 [00:00<?, ?it/s]

Done!
Making predictions...
Prediction rate: 223.7 pred/s
Obtained 243 predictions from:  SIPFENN_Krajewski2020_NN9
Prediction rate: 244.4 pred/s
Obtained 243 predictions from:  SIPFENN_Krajewski2020_NN20
Prediction rate: 1663.4 pred/s
Obtained 243 predictions from:  SIPFENN_Krajewski2020_NN24
[[0.07845493406057358, 0.07977385818958282, 0.036190714687108994],
 [0.0612497553229332, -0.011681136675179005, 0.057756152004003525],
 [0.05984886735677719, 0.06342127919197083, 0.074740797281265 26]]
```

In [42]:
```
results2 = c.get_resultDicts()
pprint(results2[:2])
```

```
[{'SIPFENN_Krajewski2020_NN20': 0.07977385818958282,
  'SIPFENN_Krajewski2020_NN24': 0.036190714687108994,
  'SIPFENN_Krajewski2020_NN9': 0.07845493406057358},
 {'SIPFENN_Krajewski2020_NN20': -0.011681136675179005,
  'SIPFENN_Krajewski2020_NN24': 0.057756152004003525,
  'SIPFENN_Krajewski2020_NN9': 0.0612497553229332}]
```

And finally combine two list of result dictionaries together.

In [43]:
```
resultsCombined = [res1 | res2 for res1, res2 in zip(results1, results2)]
pprint(resultsCombined[:2])
```

```
[{'SIPFENN_Krajewski2020_NN20': 0.07977385818958282,
  'SIPFENN_Krajewski2020_NN24': 0.036190714687108994,
  'SIPFENN_Krajewski2020_NN9': 0.07845493406057358,
  'SIPFENN_Krajewski2022_NN30': 0.17875494062900543},
 {'SIPFENN_Krajewski2020_NN20': -0.011681136675179005,
  'SIPFENN_Krajewski2020_NN24': 0.057756152004003525,
  'SIPFENN_Krajewski2020_NN9': 0.0612497553229332,
  'SIPFENN_Krajewski2022_NN30': 0.164070263504982}]
```

In [44]:
```
resultsCombinedLabeled = [{'configuration': '-'.join(permutation)} | result for
                          result, permutation in zip(resultsCombined, allPermutations)]
pprint(resultsCombinedLabeled[31:33])
```

```
[{'SIPFENN_Krajewski2020_NN20': 0.10104335844516754,
  'SIPFENN_Krajewski2020_NN24': 0.1215638816356659,
  'SIPFENN_Krajewski2020_NN9': 0.10402818769216537,
  'SIPFENN_Krajewski2022_NN30': 0.24544550478458405,
  'configuration': 'Fe-Cr-Fe-Cr-Cr'},
 {'SIPFENN_Krajewski2020_NN20': 0.0738341212272644,
  'SIPFENN_Krajewski2020_NN24': 0.05895533785223961,
  'SIPFENN_Krajewski2020_NN9': 0.06879021972417831,
  'SIPFENN_Krajewski2022_NN30': 0.19040240347385406,
  'configuration': 'Fe-Cr-Fe-Cr-Ni'}]
```

Add a new model!

Adding a new model that accepts one of the descriptors / feature vectors implemented in pySIPFENN is very easy! No matter if it is a re-trained model to fit a specific set of species, or entirely new architecture. It doesn't even need to be created in PyTorch, as pySIPFENN imports ONNX format which can be the export target of almost all ML frameworks. To add your model, you just need to put it in the **modelsSIPFENN directory** in the pySIPFENN location and add a brief definition to the **models.json file**, with field name matching model file name, descriptive name, and specify which descriptor has been used. E.g.,:

```
"SIPFENN_myNewModel": {
  "name": "SIPFENN_Krajewski2022 KS2022 Novel Materials Model - Retrained for Personal Needs",
  "descriptor": "KS2022"
}
```

Then just re-initialize the Calculator and everything should be loaded automatically!

In [45]:
```
c = Calculator()
```

```
✓ SIPFENN_Krajewski2020 Standard Materials Model
✓ SIPFENN_Krajewski2020 Novel Materials Model
✓ SIPFENN_Krajewski2020 Light Model
✓ SIPFENN_Krajewski2022 KS2022 Novel Materials Model
✓ SIPFENN_Krajewski2022 KS2022 Novel Materials Model - Retrained for Personal Needs
Loading models:

100%|██████████| 5/5 [00:16<00:00,  3.20s/it]

*********  PySIPFENN Successfully Initialized  **********
```

In [46]:
```
predictions3 = c.runModels(structList=structList, descriptor='KS2022', mode='parallel', max_workers=6)
results3 = c.get_resultDicts()
pprint(results3[:2])
```

```
Models that will be run: ['SIPFENN_Krajewski2022_NN30', 'SIPFENN_myNewModel']
Calculating descriptors...

  0%|          | 0/243 [00:00<?, ?it/s]

Done!
Making predictions...
Prediction rate: 243.5 pred/s
Obtained 243 predictions from:  SIPFENN_Krajewski2022_NN30
Prediction rate: 254.5 pred/s
Obtained 243 predictions from:  SIPFENN_myNewModel
[{'SIPFENN_Krajewski2022_NN30': 0.17875494062900543,
  'SIPFENN_myNewModel': 0.17875494062900543},
 {'SIPFENN_Krajewski2022_NN30': 0.164070263504982,
  'SIPFENN_myNewModel': 0.164070263504982}]
```

And append our new results to the previous ones!

In [47]:
```
resultsFull = [res12 | res3 for res12, res3 in zip(resultsCombinedLabeled, results3)]
pprint(resultsFull[:2])
```

```
[{'SIPFENN_Krajewski2020_NN20': 0.07977385818958282,
  'SIPFENN_Krajewski2020_NN24': 0.036190714687108994,
  'SIPFENN_Krajewski2020_NN9': 0.07845493406057358,
  'SIPFENN_Krajewski2022_NN30': 0.17875494062900543,
  'SIPFENN_myNewModel': 0.17875494062900543,
  'configuration': 'Fe-Fe-Fe-Fe-Fe'},
 {'SIPFENN_Krajewski2020_NN20': -0.011681136675179005,
  'SIPFENN_Krajewski2020_NN24': 0.057756152004003525,
  'SIPFENN_Krajewski2020_NN9': 0.0612497553229332,
  'SIPFENN_Krajewski2022_NN30': 0.164070263504982,
  'SIPFENN_myNewModel': 0.164070263504982,
  'configuration': 'Fe-Fe-Fe-Fe-Cr'}]
```

Great job! You have successfully completed the workshop!

Thank you for your attention! If you are following it in synchronous fashion, we will now head to Q&A session. If you are viewing it in your own time and have some questions, please feel free to reach out to Adam M. Krajewski (ak@psu.edu) (ak@psu.edu) or Zi-Kui Liu (zxl15@psu.edu) (zxl15@psu.edu)!

Type *Markdown* and LaTeX: α^2