Supporting information for

# LAN: A Materials Notation for 2D Layered Assemblies

Georgios A. Tritsaris,[†] Yiqi Xie,[‡] Alexander M. Rush,[†,§] Stephen Carr,[¶] Marios Mattheakis,[†] and Efthimios Kaxiras[†,¶]

[†] *John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, Massachusetts 02138, USA*

[‡] *Institute for Applied Computational Science, Harvard University, Cambridge, Massachusetts 02138, USA*

[¶] *Physics Department, Harvard University, Cambridge, Massachusetts 02138, USA*

[§] *Current address: Cornell Tech, New York, NY, USA*

A basic parser for LAN for constructing atomistic structural models of twisted layered assemblies of nanoflakes (based on Python v3.7 and ASE v3.17).

```
#Import modules for implementing the LAN parser.
import ast
import operator
import simpleeval
import re

#Import modules for modeling graphene flakes using ASE.
#ref. Larsen, A. H. et al. J. Phys.: Condens. Matter 29, 273002 (2017).
from ase import Atoms
import ase.lattice.hexagonal
from ase.neighborlist import NeighborList
import numpy as np

def graphene_flake(radius, C_C = 1.42, vacuum = None, prune = True):
    """Create a circular graphene flake.

    Creates a finite circular graphene flake in the x-y plane.

    Parameters:

    radius: float
        The radius of the circular flake.

    C_C: float
        The length of C-C bonds. Default: 1.42 Angstrom.

    vacuum: float
        If specified adjust the amount of vacuum in the
        z-direction on each side. Default: 1.75 Angstrom.

    prune: bool
        If True remove dangling atoms on the edges.
        Default: True.

    """

    a = C_C * 3**0.5
    size = int(5 * radius)
    atoms = ase.lattice.hexagonal.Graphene(symbol = 'C', latticeconstant={'a':a, 'c':1}, size=(size, size, 1))
    atoms.set_pbc(False)

    center_atom = np.argmin(np.sum((atoms.get_positions() - atoms.get_center_of_mass())**2, axis = 1))
    center_pos = atoms.get_positions()[center_atom]

    atoms_remove = np.sum((atoms.get_positions() - center_pos)**2, axis = 1) > radius**2
    del atoms[atoms_remove]

    if prune:
        cutoff = 1.10 * C_C / 2.
        nl = NeighborList(len(atoms) * [cutoff,], self_interaction = False, bothways = True, skin = 0.)
```

```
            nl.update(atoms)
            atoms_remove = len(atoms) * [False,]
            for iatom, _ in enumerate(atoms):
                indices, _ = nl.get_neighbors(iatom)
                if len(indices) == 1:
                    atoms_remove[iatom] = True
            del atoms[atoms_remove]

        if vacuum is not None:
            atoms.center(vacuum)
        else:
            atoms.center(3.5 / 2.)

        return atoms

    def layered_assembly(notation, blocks = None, vacuum = None):
        """Create a layered assembly.

        Creates a layered assembly in the z-direction.

        Parameters:

        notation: string
             The string-based notation describing the layered assembly.
             For example, the string G/G@1.08 describes a bilayer
             with twist angle of 1.08 degrees. The symbols / and @ are
             used to describe the binary operations of the vertical
             stacking of a layer or layered (sub)structure on another,
             and counterclockwise rotation of a layer or layered
             (sub)structure by some angle about the stacking direction
             (in degrees; assuming the same absolute coordinate system),
             respectively. The grammar is described in detail in
             the manuscript.

        blocks: dictionary
             Symbols describing the building blocks (case-insesitive).
             Non-alphanumeric characters will be stripped.
             Default: 'G' for a circular graphene flake with
             radius of 50 Angstrom.

        vacuum: float
             If specified adjust the amount of vacuum when centering.
             Default: No vacuum.
        """

        def Build(m):
            m = blocks[m].copy()
            return m

        def Rotate(m, a):
            m.rotate(a, 'z')
            return m

        def Stack(m1, m2):
            z1, z2 = m1.get_cell()[2][2], m2.get_cell()[2][2]
            m2.positions += (0, 0, z1)
            m = m1 + m2
            m.set_cell((0,0, z1 + z2))
            return m

        if blocks is None:
            G = graphene_flake(50)
            G.translate(-G.get_center_of_mass())
            blocks = {'G': G}

        #Sanitize input
        notation = notation.upper()
        for symbol, atoms in blocks.copy().items():
            atoms = Atoms(atoms)

            del blocks[symbol]
            symbol = re.sub(r'\W+', '', symbol)
            symbol = symbol.upper()
            blocks[symbol] = atoms

        NAMES = {}
        for symbol in blocks:
            NAMES[symbol] = 'Build("%s")' % symbol

        #Implement Stacking and Rotation only, for demonstration purposes.
        #Clockwise rotations and parentheses-precedence are also supported.
```

```
    OPERATORS = {
        ast.Mult: lambda a,r: f'Rotate({a}, {r})',
        ast.Add: lambda a,b: f'Stack({a}, {b})',
        ast.USub: operator.neg
    }

    FUNCTIONS = {}
    s = simpleeval.EvalWithCompoundTypes(names = NAMES, operators = OPERATORS, functions = FUNCTIONS)

    notation = notation.replace('@', '*').replace('/', '+')
    operations = s.eval(notation)
    atoms = eval(operations, {'Build':Build, 'Rotate':Rotate, 'Stack':Stack})

    atoms.set_pbc(False)
    if vacuum is not None:
        atoms.center(vacuum)
    else:
        atoms.center(0)

    return atoms

#### Example of using LAN to build
#### atomistic models of arbitrarily layered assemblies
#### of finite graphene flakes using ASE.
from ase.visualize import view

for twist in [0.5, 5, 10, 20]:
    #Build a twisted double bilayer of graphene flakes.
    atoms = layered_assembly('(G/G)/(G/G)@%f' % twist, vacuum = 5.)
    view(atoms)
```