Modular Software for Generating and Modelling Diverse Polymer Databases

Alejandro Santana-Bonilla,[*,†] Raquel López-Ríos de Castro,[‡,¶] Peike Sun,[¶] Robert M. Ziolek,[¶] and Christian D. Lorenz[*,¶]

†*Department of Physics, King's College London, London, WC2R 2LS, United Kingdom*

‡*Department of Chemistry, King's College London, London, SE1 1DB, United Kingdom*

¶*Biological Physics and Soft Matter Group, Department of Physics, King's College London, London, WC2R 2LS, United Kingdom*

E-mail: alejandro.santana_bonilla@kcl.ac.uk; chris.lorenz@kcl.ac.uk

Abstract

Machine learning methods offer the opportunity to design new functional materials on an unprecedented scale however building the large, diverse databases of molecules on which to train such methods remains a daunting task. Automated computational chemistry modelling workflows are therefore becoming essential tools in this data-driven hunt for new materials with novel properties, since they offer a workflow by which to create and curate molecular databases without requiring significant levels of user input. This ensures well-founded concerns regarding data provenance, reproducibility and replicability are mitigated. We have developed a versatile and flexible software package, PySoftK (Python Soft Matter at King's College London), that provides flexible, automated computational workflows to create, model, and curate libraries of polymers with a minimal user intervention. PySoftK is available as an efficient, fully-tested, and easily installed Python package. Key features of the software include the wide

range of different polymer topologies that can be automatically generated and fully parallelized library generation tools. It is anticipated that PySoftK will support the generation, modelling and curation of large polymer libraries to support functional materials discovery in nano- and bio-technology.

Introduction

The diverse chemical, mechanical and electronic properties of polymers underlie their application in relevant technological areas spanning structural materials, cosmetics, pharmaceutical formulation, electronics, and biotechnology.[1–10] In order to optimize their aforementioned properties for this range of applications, the role of polymer constitution and topology has been widely explored, with a range of architectures including homopolymers, block copolymers, branched and ring polymers.[11–17] As a result of contemporary research activities, the chemical and structural domains of synthetic polymers are continuously growing.[18–20]

Polymeric materials have been the focus of a significant amount of scientific research for many decades. In more recent years, access to ever-growing computational power has allowed the materials modelling community to carry out increasingly complex investigations of polymeric materials.[21–28] Modern computer simulation studies provide predictive understanding of molecular interactions and mechanisms that determine the bulk-scale properties of polymers of interest. In tandem with experimental data, this combined insight yields rational design principles for new polymers with enhanced target properties.

While there are well-established computer simulation techniques to support the investigation of polymers, often the most technically difficult part of a polymer simulation workflow is actually generating models of the polymer(s) of interest. As a result, multiple computational platforms have been developed in recent years with the aim of making the generation of polymer models more straightforward.[29–33] Several of these packages have been developed to allow users to build molecular models of polymers for use in molecular dynamics simulations. These packages generally require the user to provide a detailed description of the underlying

chemistry of, and connectivity between, constituent monomers to output a topology of the desired polymer. Some allow the user to input forcefield information to be used in generating input files for classical molecular dynamics simulations. All of the referenced codes allow for the user to build homopolymers while some allow the user to build heteropolymers with different architectures and topologies.[29–32]

More recently, Polymer Structure Predictor (PSP)[34] has reduced the amount of information required from the user to describe the chemistry of the monomers by allowing a SMILES string to be used as the input. The code can assign forcefield parameters from the CHARMM, AMBER and OPLS generalized forcefields[35–37] and integrates with the LAMMPS classical simulation engine[38] and the pysimm package[29] for optimization of polymer structures. While the amount of the predefined information required to build the polymers is less in PSP than is required by other packages, the polymers that can currently be built are limited to homopolymers with either linear or cyclic topologies.

Here, we present PySoftK, a modular and versatile code to model polymer structures with different topologies. We present the various modules that currently exist within PySoftK to build different polymer structures, and show how they can be combined in order to build highly complex polymer topologies in an automated way. We subsequently demonstrate the various functionalities that are found within the code to facilitate high-throughput molecular modelling calculations. PySoftK can also be used in the parameterization of dihedral terms in conjugated polymers, which are commonly poorly defined by standard classical forcefields.[28,39] Finally, we briefly review the steps that have been taken when developing the code to ensure it can be successfully used in a broad range of applications.

Software overview

PySoftK is a modular Python package that generates molecular models of polymers with a diverse range of topologies and chemistries with minimal input from the user. It also

has various tools to facilitate high-throughput simulations of polymers. The code utilises RDKit to generate the molecular models of the various polymers.[40] The modular nature of this package will allow the scientific community to easily expand the code base to include polymer architectures that are not included in this first release of the library.

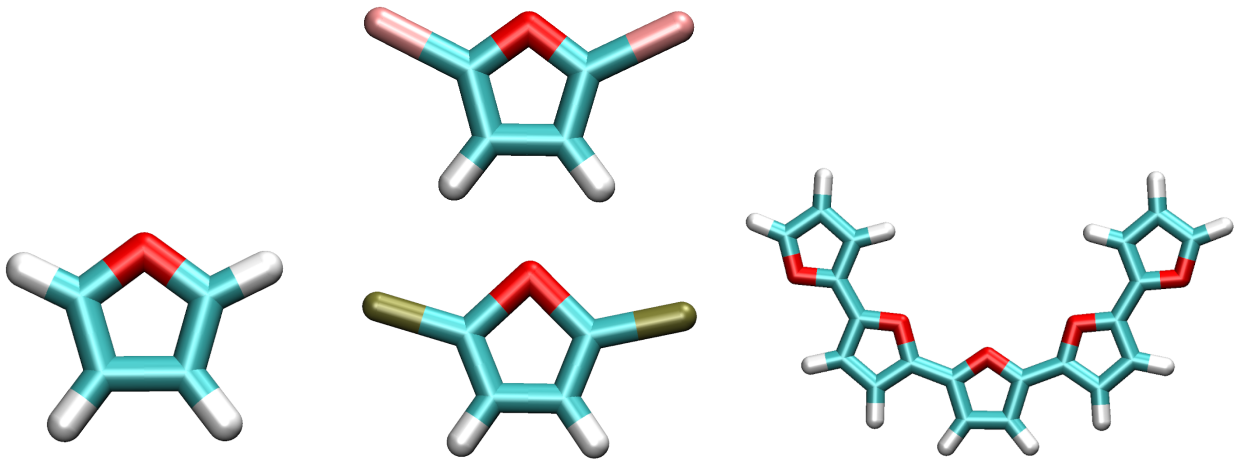
Generating diverse polymer architectures

Inputting monomers. A key functionality of PySoftK is the automated generation of a diverse range of polymer architectures. The user-inputted monomers, in which the sites that link each monomer together are predefined, are the main topological descriptor used by PySoftK. The chemical structure and connectivity of the monomer(s) that make up the polymer can be inputted using all formats supported by RDKit, such as SMILES strings and .pdb, .mol and .xyz files, as shown in Listing 1. Examples of inputs to input a furan monomer are shown in Figure 1a. In order to construct a polymer from its constituent monomers, a placeholder atom is used to indicate where the bond formation takes place during polymerization. In the examples presented in this manuscript, bromine (Br) atoms or platinum (Pt) are used as these placeholder atoms (see Fig. 1b). It is important to note that this choice is entirely arbitrary: any atom can be used as the placeholder atom.

```
from rdkit import Chem
from rdkit.Chem import AllChem
# SMILES. MOL and PDB formats
mol_1=Chem.MolFromSmiles('c1(ccc(o1)Br)Br')
mol_2=Chem.MolFromPDBFile('furan_pysoftk.pdb')
mol_3=Chem.MolFromMolFile('furan_pysoftk.mol')
```

Listing 1: Code snippet showing the different input foramts that PySoftK accepts.

Generating polymers with different monomer distributions. Once the monomers are defined, then PySoftK can be used to describe the distribution of the monomers within the polymer and the overall architecture of the polymer. The code is structured such that



(a) (b) (c)

Figure 1: Creation of a linear polymer. (a) Undecorated single furan molecule. (b) Decorated furan molecule (using a Br or Pt atom) highlighting the region where intermolecular bonds are formed between monomers. (c) Linear polymer with a 5 monomers and optimized atomic positions at the RDKit **MMFF** force-field level.[40]

specific modules are provided to generate homopolymers, diblock copolymers and random polymers. Block and random copolymers with more than two types of monomers, as well as alternating, periodic and statistical copolymers of any number of monomers, can be built using the `patterned` module within PySoftK. The modular structure of the software allows users to develop their own codes to design additional polymer topologies. In the following sections, examples of the different modules to build the various polymer topologies with different distributions of the monomers are provided.

Generating polymers with different topologies. Users can generate models of polymers with three different general topologies: (i) linear, (ii) cyclic and (iii) branched. In the following sections, examples showing how to use the various modules to build each block and random copolymers with each topology will be presented. Finally, we will present the `patterned` module, which provides the user more flexibility in defining the distribution of the monomers within the polymer and can be used in combination with the various topology modules to build any of these different polymer architectures.

Linear polymers. Linear polymers consist of individual monomers which are joined

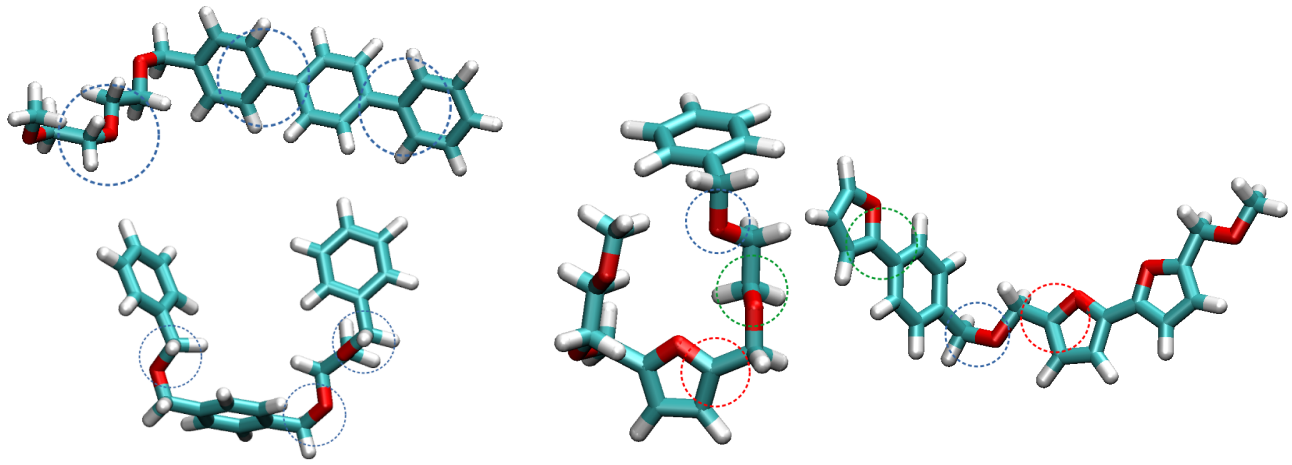
together end-to-end forming a single molecule. In order to build a linear homopolymer in PySoftK, the command `Lp(mol,atom,n_copies,shift).linear_polymer(FF,iter_ff)` is used to initiate a two-step process. First, a monomer (defined by the variable `mol`) is copied and translated along a predefined axis, where the distance between each monomer is defined by the `shift` variable in the module call, which results in an unconnected chain of `n_copies` monomers. Subsequently, a merging step between end-to-end monomers is performed employing a user-designated atomic placeholder (defined by the variable `atom`) indicating to PySoftK the selected site on the monomer where a bond is formed. Finally, to provide a realistic initial structure, PySoftK utilizes the **MMFF** or **UFF** force-field parameterizations (as chosen with the `FF` parameter) as implemented in RDKit[40] in order to generate an energy minimized molecular conformation after `iter_ff` steps of minimisation. This process is displayed in Figure 1, where a single furan molecule is decorated with a placeholder atom (in this case bromine) shown in pink and extended to form a linear polymer containing five monomers. The full block of code required to generate this homopolymer is found in Listing 2.

The `Db` module within PySoftK allows users to build linear diblock copolymers. The command `Db(ma,mb,atom).diblock_copolymer(len_block_A,len_block_B,FF,iter_ff)` is used to generate a diblock copolymer that consists of a block containing `len_block_A` monomers of `ma` and a block containing `len_block_B` monomers of `mb`. Again the monomers are inputted with atomic placeholders (`atom`) that identify the polymerisation site on each monomer, and the resultant structure undergoes `iter_ff` steps of energy minimisation using the `FF` forcefield. Listing 2 shows an example of how to practically apply this function.

```
from pysoftk.linear_polymer.linear_polymer import *
from pysoftk.topologies.diblock import *

# Build linear homopolymer
# Input monomer with SMILES format
mol_1=Chem.MolFromSmiles('c1(ccc(o1)Br)Br')
```

```
AllChem.EmbedMolecule(mol_1)
new=Lp(mol_1,"Br",5,shift=1.25).linear_polymer("MMFF",1)

# Build linear diblock copolymer
# Input ethylene oxide monomer with PDB format
mol_1 = Chem.MolFromPDBFile('eo.pdb')
# Input propylene oxide monomer with PDB format
mol_2 = Chem.MolFromPDBFile('po.pdb')
# Build a polymer with 5 eo monomers & 7 po monomers which is minimised
   using the MMFF forcefield over 10 iterations
poly = Db(mol_1,mol_2,"Br").diblock_copolymer(5,7,'MMFF',10)
```

Listing 2: Code snippet showing the creation of a linear homopolymer and a linear diblock copolymer.



(a) (b)

Figure 2: Creation of a random polymer with user defined probabilities. (a) Diblock copolymer architecture, where a single probability (P_1) of being linked is 0.4. The linkages are highlighted with a blue circle. Two different polymeric topologies are obtained using the same probability. (b) Triblock copolymer architecture employing a user-defined probabilities P_1 (represented in green) and P_2 (displayed in red). Two different moieties are obtained based on user defined probabilities (0.4 and 0.2), respectively.

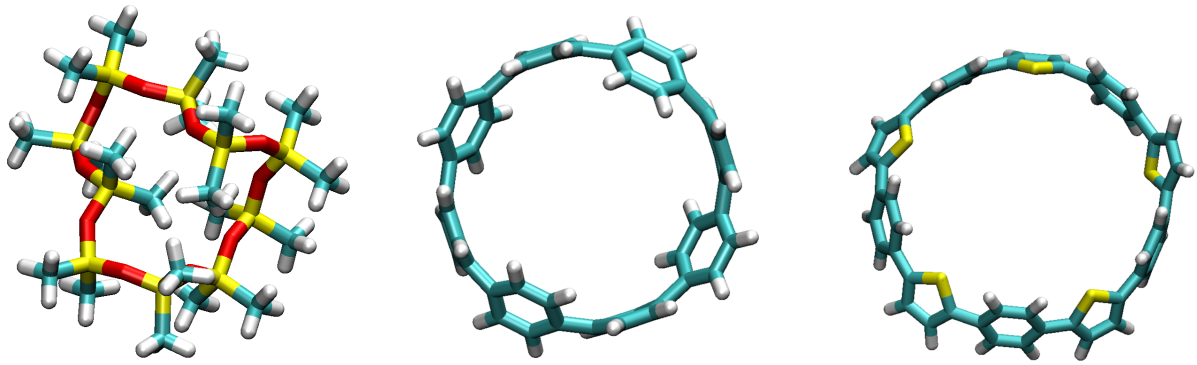
Additionally, PySoftK supports the construction of random linear copolymers which consist of two or three different monomers. The module `Rnp(mol_1,mol_2,atom).random_ab_copolymer`

`(len,pA,iter_ff,FF)` is used to build a random copolymer of length `len` that contains two monomers, `mol_1` and `mol_2` which are decorated with atoms of type `atom` to indicate where the polymerisation occurs. The number of `mol_1` and `mol_2` monomers in the resulting polymer are determined from `pA*len` and `(1-pA)*len`, respectively. Similarly PySoftK can be used to generate random linear copolymers consisting of three different monomers via the module `Rnp(mol_1,mol_2,atom).random_abc_copolymer(mol_3, len, pA, pB, iter_ff, FF)`. In this case, the resulting polymer will contain `len` monomers, such that the number of `mol_1`, `mol_2` and `mol_3` monomers is `pA*len`, `pB*len` and `(1-pA-pB)*len`, respectively. A snippet showing the usage of the functions is displayed in Listing 3, and examples of the polymers produced by that code are presented in Figure 2.

```
from pysoftk.topologies.ranpol import *

mol_1=Chem.MolFromSmiles('BrCOCBr')
mol_2=Chem.MolFromSmiles('c1(ccc(cc1)Br)Br')
mol_3=Chem.MolFromSmiles('c1cc(oc1Br)Br')

#diblock random polymer
dia = Rnp(mol_1, mol_2,"Br").random_ab_copolymer(5, 0.4, 10)
Fmt(dia).xyz_print("dia.xyz")

#triblock random polymer
tri = Rnp(mol_1, mol_2,"Br").random_abc_copolymer(mol_3, 5, 0.4, 0.2, 10)
Fmt(tri).xyz_print("tri.xyz")
```

Listing 3: Code snippet showing the usage of the random copolymer module functions. The functions `random..ab__copolymer` and `random..__abc__copolymer` allowing the definition of the user-supplied linking probabilities.

Ring polymers. Current experimental techniques allow a precise control of polymer architectures enabling the creation of new topologies. One interesting example is ring, or cyclic, polymers, which can be described as closed macromolecular structures with no beginning or



(a) (b) (c)

Figure 3: Creation of a ring polymers using (a) silanol[42] and (b) benzene ring. (c) Alternating copolymer (benzene and thiol monomers combined via the `Sm` module). This example illustrates the capabilities of PySoftK to utilize previously developed algorithms for creating new architectures.

end.[41] PySoftK enables the creation of ring homopolymers with the module `Rn(mol_1,atom).pol_ring(len,FF,iter_ff)` that generates a ring polymer with `len` `mol_1` monomers by first employing the linear polymer module to generate a polymer chain and then initial and final atomic placeholders (`atom`) are used to create a bond, which generates a ring topology, as displayed in Figure 3(a,b). A snippet of code utilising this command is shown in Listing 4. It is worth mentioning that PySoftK enables the combination of modules to create new structures. This is demonstrated in the bottom part of Listing 4, where the `Sm` module is used to create a linear diblock polymer that is then converted to a ring polymer topology using the `Rn` module, with the resultant ring polymer shown in Figure 3(c).

```
from pysoftk.topologies.ring import *
from pysoftk.linear_polymer.super_monomer import *

# Silanol definition using SMILES and PySoftK ring function
mol_1 = Chem.MolFromSmiles('C[Si](OBr)(C)Br')
rn2=Rn(mol_1,'Br').pol_ring(10, "MMFF", 250)

#Using the Sm module of PySoftK to generate a diblock cyclic polymer
mol_1=Chem.MolFromSmiles('c1cc(sc1Br)Br')
```

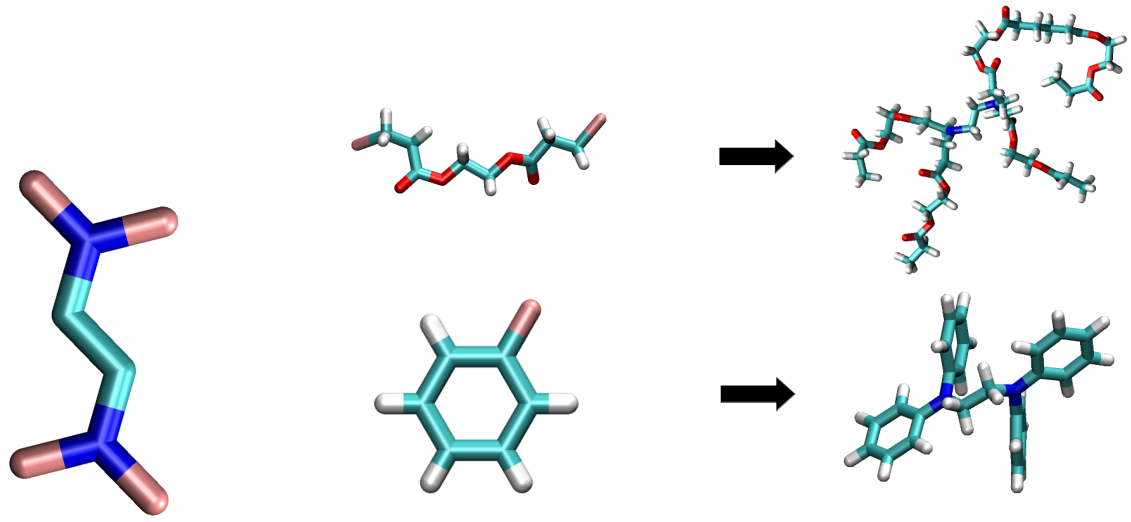
```
mol_2=Chem.MolFromSmiles('c1(ccc(cc1)Br)Br')

a=Sm(mol_1,mol_2,"Br").mon_to_poly()
cyc=Rn(a,'Br').pol_ring(5, FF="UFF", iters=2000)
```

Listing 4: Code snippet showing the creation of a cyclic polymer

Branched Polymers. Branched polymers are another topology that has been enabled in PySoftK. Generally, branched polymers are a set of secondary polymer chains linked to a primary backbone. PySoftK builds this topology by employing the user-supplied atomic placeholder as an indicator of the number of branch points from the primary backbone present in this structure. Thus, for instance, a backbone with four placeholders will generate a model where four branches (arms) are attached at the regions on the backbone indicated by the user, as shown in Figure 4. In PySoftK, the module `Bd(core,arm,atom).branched_polymer(FF, ff_iter)` is used to generate a branched polymer with a backbone of `core` and arms described by `arm` (Listing 5). The placeholder atoms are defined by `atom`. The inputted structures for `core` and `arm` can be simple monomers and therefore inputted as previously discussed or they can be resultant structures from any of the other commands and then inputted into the branched function. Therefore, the branched architecture not only illustrates the capabilities of PySoftK to create new architectures from scratch but also the versatility of PySoftK to be extended. In this case, this has been done by creating the module `topologies` where all the previous functions have been organized.

```
from pysoftk.topologies.branched import *

# Core and Arm molecules
core=Chem.MolFromSmiles('BrN(Br)CCN(Br)Br')
arm=Chem.MolFromSmiles('C(CC(=O)OCCOC(=O)CCBr)Br')
final = Bd(core, arm, "Br").branched_polymer()
```

Listing 5: Code snippet showing the creation of a branched polymer

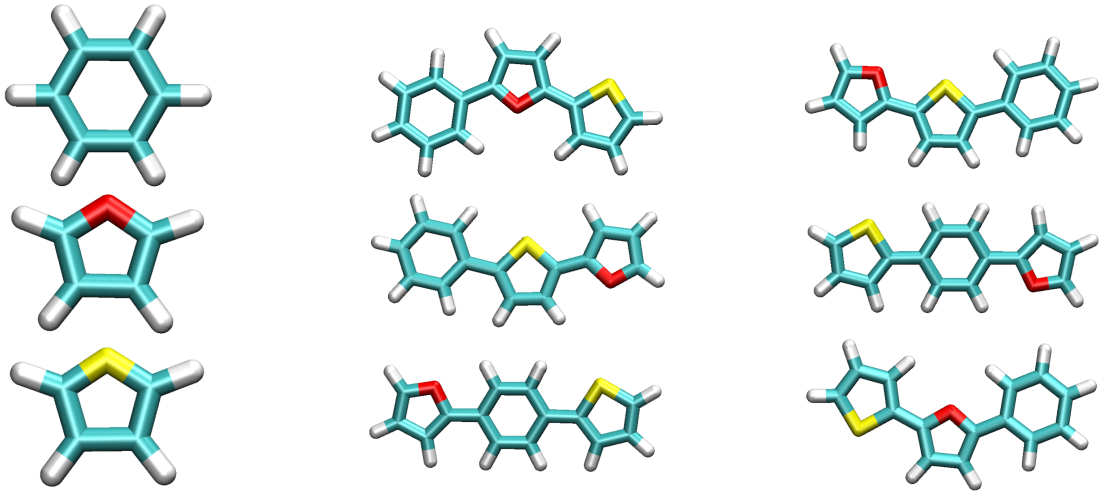


(a) (b) (c)

Figure 4: Creation of a branched polymers using (a) ethylenediamine as the primary backbone and (b) benzene or polyester oligomer as branches. (c) Branched polymer with provided core and arms moieties. In this case, only one placeholder atom is used to signal PySoftK the region where a bond would be formed.

Specifying absolute monomer sequences. Any linear polymer can be described by listing the combination of monomers in a specific pattern (e.g. ABBACCABBACC, ABCABABCBA). Therefore, PySoftK offers the option to build polymers with a specific pattern of monomers using an alphabet-based pattern expressed in a single string followed by a list of RDKit[40] molecular objects as presented in Listing 6. In Figure 5, thiophene, furan and benzene monomers are used to present the different possibilities provided by this function to construct all unique permutations.

```
from pysoftk.topologies.diblock import *

# Core and Arm molecules
mols=[Chem.MolFromSmiles('c1(ccc(cc1)Br)Br'),
      Chem.MolFromSmiles('c1cc(oc1Br)Br'),
      Chem.MolFromSmiles('c1cc(sc1Br)Br')]

patt="ABC"
```



(a) **A/B/C** (b) **ABC/ACB/BAC** (c) **BCA/CAB/CBA**

Figure 5: Figure (4) Patterned construction of a polymer based on a user-defined pattern. (a) An user created list is mapped to an alphabetic string representing the position of the molecule in the list. (b) Possible unique permutations for an arbitrary list containing 3 elements.

```
9 a=Pt(patt, mols, "Br").pattern_block_poly()
```

Listing 6: Code snippet showing the creation of a patterned polymer

This function can also be used in conjunction with the `linear_polymer` module of PySoftK to construct polymeric macromolecules that have other polymers as their monomeric units.

Facilitating High-Throughput Calculations

Folder creation and organization. The module `pysoftk.folder_manager.folder_creator` has been designed to create a user-defined number of folders with unique names. In the case of high-throughput calculations (HTC), usually many different systems are created in a single folder and then relocated to enable calculations or post-processing analyses to be conducted. As shown in Listing 7, this workflow can be carried out utilizing the function `Fld().file_to_dir` implemented in PySoftK. The result of the code in Listing 7 is that two new, uniquely named directories would be created and each of the two `.smi` files (`mol_1.smi` & `mol_2.smi`) would be moved into one of these new directories, such that each new directory

would contain one `.smi` file.

```
from pysoftk.folder_manager.folder_creator import *

print('c1cc(sc1Br)Br',file=open("mol_1.smi", 'w'))
print('c1(ccc(cc1)Br)Br', file=open("mol_2.smi",'w'))

# Test 3: Seek for .smi files and create the needed folders.
Fld().file_to_dir("smi",2)
```

Listing 7: Code snippet showing the automatic creation of folder to organize files based on an user-defined extension.

The aforementioned function is able to search for files with a given file extension, and relocate them within individually created folders. Likewise, for cases where many files are present, this command can be executed in parallel. The creation of automated workflows (as displayed in Figure 6) can be achieved by combining many different functions in PySoftK enabling the modeling of thousands of polymers using a single script. An example demonstrating how this can be done can be found in the SI. Alternatively, these modules can be used separately as part of another workflow that generates other types of directory structures or simply a folder where many different files are located.

Automatic torsional angle detection for conjugated polymers. PySoftK offers a tool to perform analysis of the torsion angles within conjugated polymers that connect the ring subunits that are commonly found in their backbone. These dihedrals are generally poorly captured by the existing parameters within classical forcefields.[28,39] Thus having the capability to rapidly identify those dihedrals then allows the user to easily set up the necessary *ab initio* simulations required to determine the potential energy landscape of those dihedrals, which then can be used to reparameterize the forcefield for those dihedrals.

The module `pysoftk.torsional` automatically detects and reports the atoms involved in the torsional angles found within planar conjugated polymers. PySoftK also enables the creation of molecular sketches highlighting and reporting the atom numbers that form a

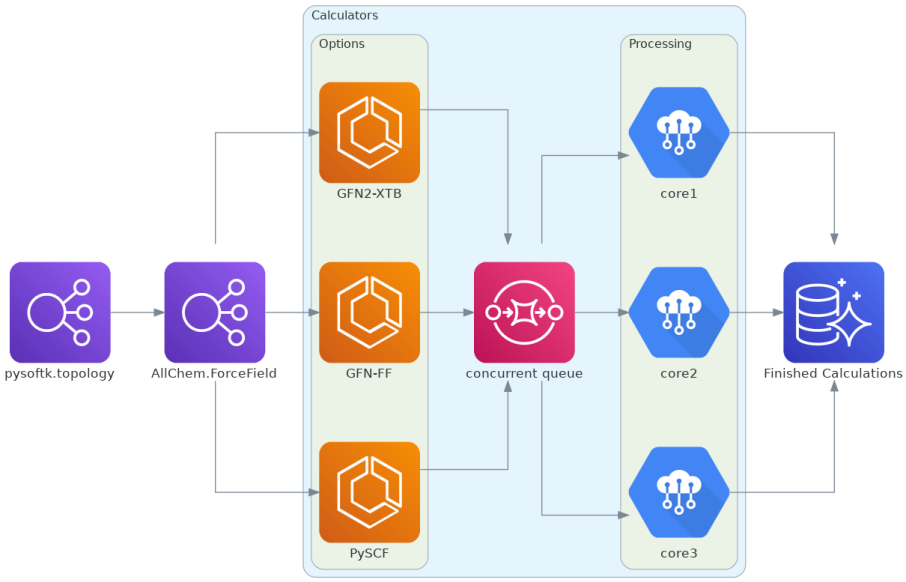
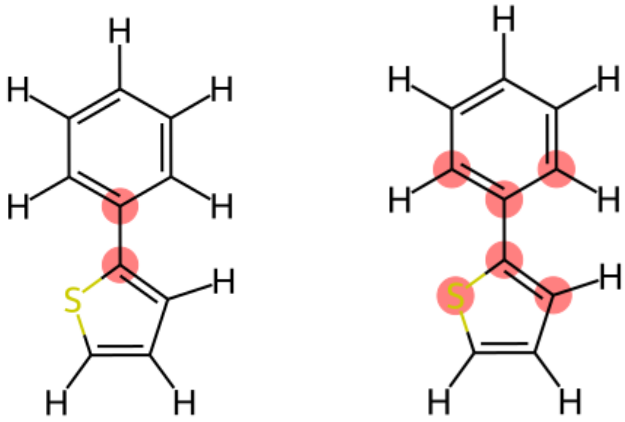


Figure 6: PySoftK workflow generation. Topology objects (polymers) are created and automatically parsed to RDKit force-field optimization.[40] External programs such as GFN2-XTB, GFN-FF or PySCF are linked using internal modules which facilitates the automation process.[43,44] Concurrent parallelization allow the creation of events in which subsequent parallelization strategies can be used. File organization is performed.



(a) (b)

Figure 7: Graphical explanation of the **P-Graph** description and algorithm used for automatically detecting intermolecular torsional angles. (a) Bond linking atoms are detected. (b) All connecting paths are computed where previously detected atoms are used as starting and end points.

torsional angle as shown in Figure 7. The topological fingerprint used in this module relies on the description of molecules as graphs where the atoms are nodes and the covalent bonds

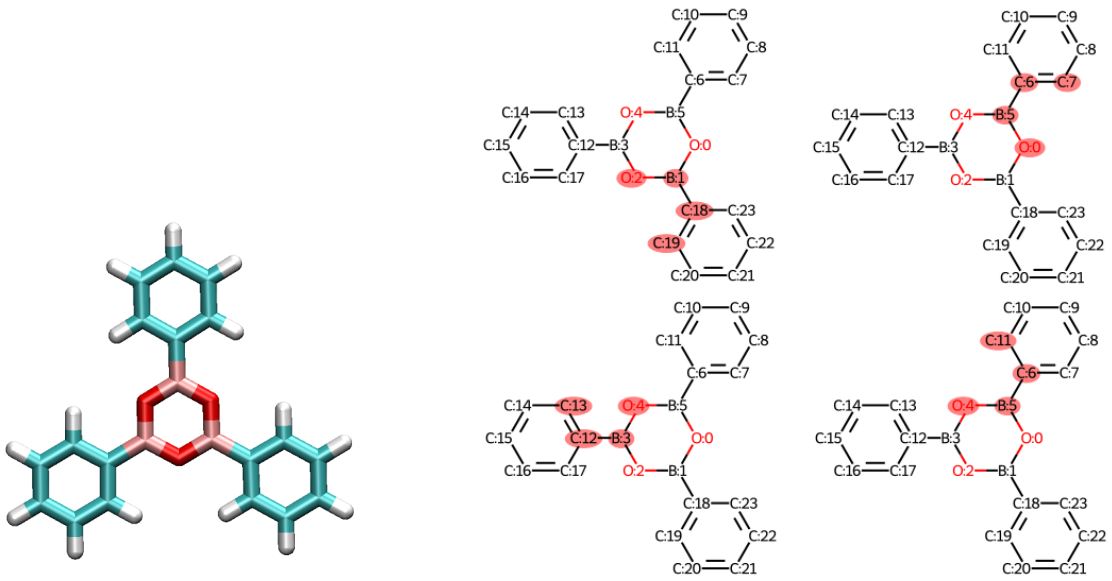
are the edges.[45] Based on the idea to convert a molecular Graph (**M-Graph**) into a path Graph (**P-Graph**), we have been able to label connections between edges in the **M-Graph** providing a nomenclature to name the **P-Graph**.[46]

In order to identify the torsions between ring subunits within the backbone of a conjugated polymer, we have identified two conditions that the atoms (e.g. nodes) making up the central bond of the torsion must meet. First, an atom must have three neighbours which are from the inner structure (such as rings) and one provided from the next monomer (as displayed in Fig. 7a). However, this condition can be also be found in atoms that are embedded within aromatic rings. To avoid this, we have used the ring detection function provided by RDKit to remove potential bonds within a ring, and therefore identifty the bonds which connect the ring moieties in the polymer (Fig. 7a). After identifying all of these important bonds within the polymer, we then identify all of the torsions that include these bonds as the central bond and therefore can provide the relevant atomic labels involved in these torsional angles as shown in Figure 7b.[47]

This module then reports all of these important torsional angles within the molecule, and generates 2D molecular sketches of each one. Figure 8 shows an example of this output for a boroxine polymer.[48]

Data Provenance and Software Development

One of our main objectives while creating PySoftK is to provide tools to the community that allow users to utilize high-performance computational facilities to automate the creation of databases for polymeric structures as easily as possible. To do so, we have greatly simplified the installation process of PySoftK by employing the `pip` command strategy. PySoftK has been tested in Linux and Mac operating systems based on python 3.6 or more recent versions. Parallel strategies have been developed in parts of the code to enhance the scalability in tasks such as HTC or general organization of molecular databases. In this sense, we have developed



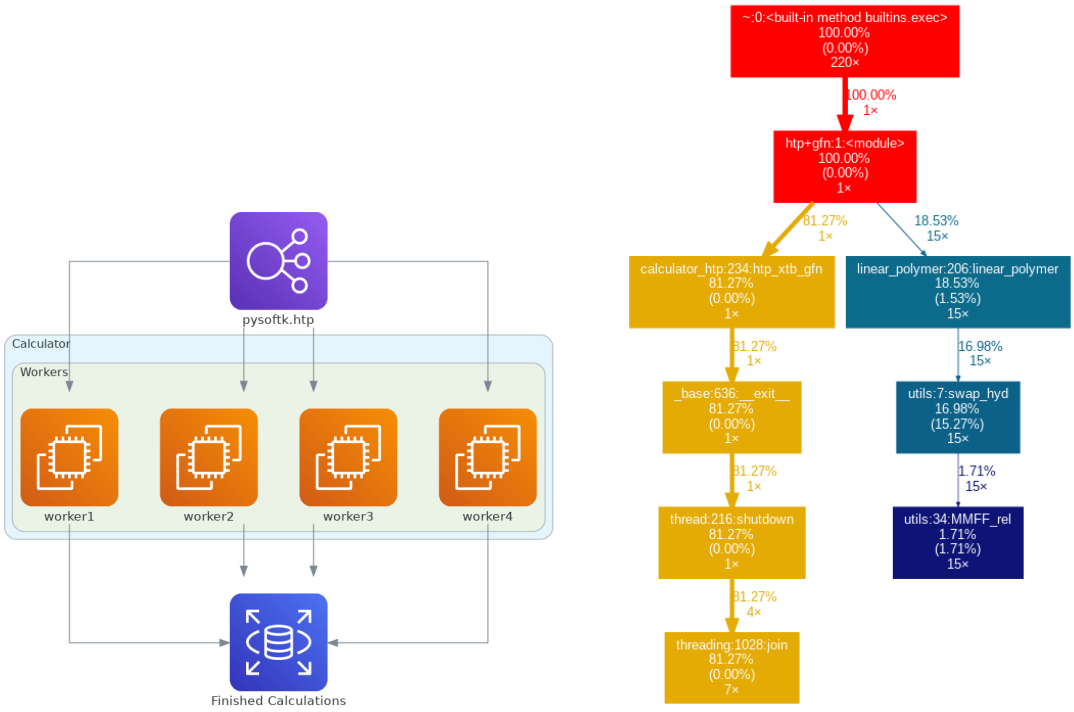
(a) (b)

Figure 8: Automatic recognition of intermolecular torsional angles within a planar polymer. (a) Boroxine planar polymer. (b) 2D depiction of selected torsional angles highlighting the atoms and its numerical labels.

modules of PySoftK using concurrency where independently executing tasks are created and queued to use the available resources. This approach ensures that the `calculator` module can run utilizing all available resources combining task assignment and core usage for parallel codes such as GFN2-xTB[43] while efficiently avoiding bottlenecks, as displayed in Figure 9.[49] In this sense, the user can maximize the parallel performance of the code by tuning variables such as number of available processors against simultaneous task management.

PySoftK has been constructed using principles of modern code development practices. Thus, continuous integration (CI) strategies have been incorporated to probe the code integrity against suggested changes. In our case, these tests have been designed not only as a showcase of all of the capabilities of PySoftK, but also with the aim of maximum code coverage achieved with our current test set. This design ensures that new commits are compatible with the majority of the modules before a new version is integrated into the main branch and released.

Continuous deployment (CD) is achieved once all previous tests have passed enabling



(a) (b)

Figure 9: Parallelization strategies implemented in PySoftK. (a) Schematic diagram presenting the distribution of user defined workers (threads). Each worker performs a calculation using the *pysoftk.calculator* object where further parallelization schemes can be used (cores). (b) Usage report of a calculation performed using an Intel(R) Core(TM) i9-900 CPU 3.10 GHz computer employing 16 polymers created on-the-fly in a $4x4$ scheme where 4 threads are performing task management while 4 cores are used to perform *ab-initio* calculations at the GFN2-xTB level of theory.[43,49,50]

an agile updating of PySoftK. At this point, we would like to comment, that a successful CD is only achieved when the corresponding explanatory examples are included in the documentation. This process is greatly facilitated since the documentation is also part of PySoftK building process. Finally, a detailed documentation (which is also part of our CI/CD workflow) has been developed showing working examples (based on our tests) alongside of tutorials for all the different features enabled in PySoftK. They have been tested by members of our community and feedback has been incorporated into the latest version of the code, ensuring a clear and concise approach for new users.

Conclusions

PySoftK is a modular and versatile software that has been designed to facilitate high-throughput calculations of polymeric systems. The code contains a range of modules that allow its users to generate polymers with a uniquely broad range of topologies and compositions. Additionally, the code has unique tools to assist in the file and directory structure management which is inherent in high-throughput calculations. All of the functionality within PySoftK has been developed to be embedded as part of user-defined workflows enabling steps such as modeling or computing using the specifically defined modules. In a complementary fashion, PySoftK allows the user to keep a track-record of the data produced, facilitating post-processing analysis that can be also part of the same workflow. An ample set of testing scripts has been created aiming at a high-coverage of the code which ensures that future algorithm developments are preserving the code structure. Further documentation and tutorials are available on the PySoftK website (https://alejandrosantanabonilla.github.io/pysoftk/#). The modular nature of the code allows for the user community to easily contribute code that complements the existing functionality of this published version of the code.

Acknowledgement

We are grateful to the UK Materials and Molecular Modelling Hub, which is partially funded by EPSRC (EP/P020194/1 and EP/T022213/1) and the UK HPC Materials Chemistry Consortium, which is also funded by EPSRC (EP/R029431) for providing us access to computational resources. This work also benefitted from access to the King's Computational Research, Engineering and Technology Environment (CREATE) at King's College London.[51] R. L.-R. D. C. acknowledges the support by the Biotechnology and Biological Sciences Research Council (BB/T008709/1) via the London Interdisciplinary Doctoral Programme (LIDo). R. M. Z. and C. D. L. acknowledge the Engineering and Physical Sciences Research Council (EPSRC) for funding (EP/V049771/1).

Supporting Information Available

Results of a computational benchmark for testing the Htp module of PySoftK altogether with the pictures of all bonds detected by Boroxine are displayed in the electronic supporting information.

References

(1) Foglia, F. et al. Disentangling water, ion and polymer dynamics in an anion exchange membrane. *Nat. Mater.* **2022**, *21*, 555–563.

(2) Foglia, F.; Frick, B.; Nania, M.; Livingston, A. G.; Cabral, J. T. Multimodal confined water dynamics in reverse osmosis polyamide membranes. *Nat. Commun.* **2022**, *13*, 2809.

(3) Getzler, Y. D. Y. L.; Mathers, R. T. Sustainable polymers: Our evolving understanding. *Acc. Chem. Res.* **2022**, *55*, 1869–1878.

(4) Wang, S.; Zuo, G.; Kim, J.; Sirringhaus, H. Progress of conjugated polymers as emerging thermoelectric materials. *Prog. Polym. Sci.* **2022**, *129*, 101548.

(5) Wang, Q.; Huang, L.; Wang, Z.; Zheng, J.; Zhang, Q.; Qin, G.; Li, S.; Zhang, S. High conductive anion exchange membranes from all-carbon twisted intrinsic microporous polymers. *Macromolecules* **2022**, *55*, 10713–10722.

(6) Yang, X.; Bai, R.; Zhang, Z.; Liu, Y.; Yan, X. Mechanically tunable supramolecular polymer networks with different triblock backbones. *J. Polym. Sci.* **2022**, doi: 10.1002/POL.20220615.

(7) Li, Z.; Shen, Z.; Pei, Y.; shuang chao,; Pei, Z. Covalently bridged pillararene-based polymers: structures, synthesis, and applications. *Chem. Comm.* **2023**, doi: 10.1039/D2CC05594E.

(8) Coscia, B. J.; Shelley, J. C.; Browning, A. R.; Sanders, J. M.; Chaudret, R.; Rozot, R.; Léonforte, F.; Halls, M. D.; Luengo, G. S. Shearing friction behaviour of synthetic polymers compared to a functionalized polysaccharide on biomimetic surfaces: models for the prediction of performance of eco-designed formulations. *Phys. Chem. Chem. Phys.* **2023**, doi: 10.1039/D2CP05465E.

(9) Gouveia, M. G.; Wesseler, J. P.; Ramaekers, J.; Weder, C.; Scholten, P. B. V.; Bruns, N. Polymersome-based protein drug delivery – quo vadis? *Chem. Soc. Rev.* **2023**, doi: 10.1039/D2CS00106C.

(10) Zhang, Y.; Chen, J.; Shi, L.; Ma, F. Polymeric nanoparticle-based nanovaccines for cancer immunotherapy. *Mater. Horiz.* **2023**, doi: 10.1039/D2MH01358D.

(11) Oliveira, A. S.; Pereira, P.; Mendonça, P. V.; Fonseca, A. C.; Simões, S.; Serra, A. C.; Coelho, J. F. Novel degradable amphiphilic 4-arm star PLA-b-POEOA and PLGA-b-POEOA block copolymers: synthesis, characterization and self-assembly. *Polym. Chem.* **2023**, *14*, 161–171.

(12) Wang, Y.; Su, H.; Wang, Y.; Cui, H. Discovery of Y-shaped supramolecular polymers in a self-assembling peptide amphiphile system. *ACS. Macro. Lett.* **2022**, *14*, 1355–1361.

(13) Li, W.; Hadjigol, S.; Mazo, A. R.; Holden, J.; Lenzo, J.; Shirbin, S. J.; Barlow, A.; Shabani, S.; Huang, T.; Reynolds, E. C.; Qiao, G. G.; O'Brien-Simpson, N. M. Star-

peptide polymers are multi-drug-resistant hram-positive bacteria killers. *ACS Appl. Mater. Interfaces* **2022**,

(14) Grest, G. S.; Ge, T.; Plimpton, S. J.; Rubinstein, M.; O'Connor, T. C. Entropic mixing of ring/linear polymer blends. *ACS Polym. Au* **2022**, doi: 10.1021/ACSPOLYMERSAU.2C00050.

(15) Zeng, X. C.; Zhao, H.; Chen, C.; Weil, T. Cyclic polymers: synthesis, characteristics, and emerging applications. *Nanoscale Horiz.* **2022**, *2*, 1–66.

(16) Liu, M.; Blankenship, J. R.; Levi, A. E.; Fu, Q.; Hudson, Z. M.; Bates, C. M. Miktoarm star polymers: Synthesis and applications†. *Chem. Mater.* **2022**, *34*, 6188–6209.

(17) Kostka, L.; Kotrchová, L.; Randárová, E.; Ferreira, C. A.; Malátová, I.; Lee, H. J.; Olson, A. P.; Engle, J. W.; Kovář, M.; Cai, W.; Šírová, M.; Etrych, T. Evaluation of linear versus star-like polymer anti-cancer nanomedicines in mouse models. *J. Control. Release* **2023**, *353*, 549–562.

(18) Gauthier-Jaques, M.; Mutlu, H.; Theato, P. Cage-shaped polymers synthesis: A comprehensive state-of-the-art. *Macromol. Rapid Commun.* **2022**, *32*, 2100760.

(19) Jing, Y.; Zhu, X.; Maier, S.; Heine, T. 2D conjugated polymers: exploiting topological properties for the rational design of metal-free photocatalysts. *Trends Chem.* **2022**, *4*, 792–806.

(20) Muramatsu, Y.; Takasu, A. Synthetic innovations for cyclic polymers. *Polym. J.* **2021**, *54*, 121–132.

(21) Ziolek, R. M.; Omar, J.; Hu, W.; Porcar, L.; González-Gaitano, G.; Dreiss, C. A.; Lorenz, C. D. Understanding the pH-directed self-assembly of a four-arm block copolymer. *Macromolecules* **2020**, *53*, 11065–11076.

(22) Xue, Y. L.; Huang, J.; Lau, C. H.; Cao, B.; Li, P. Tailoring the molecular structure of crosslinked polymers for pervapration desalination. *Nat. Commun.* **2020**, *11*, 1461.

(23) Liu, J.; Zheng, Y.; Zhao, Z.; Yuan, M.; Tsige, M.; Wang, S.-Q. Investigating nature of stresses in extension and compression of glassy polymers via stress relaxation. *Polymer* **2020**, *202*, 122517.

(24) Ziolek, R. M.; Smith, P.; Pink, D. L.; Dreiss, C. A.; Lorenz, C. D. Unsupervised learning unravels the structure of four-arm and linear block copolymer micelles. *Macromolecules* **2021**, *54*, 3755–3768.

(25) Scherillo, G.; Mensiteri, G.; Baldanza, A.; Loianno, V.; Musto, P.; Pannico, M.; Correa, A.; Nicola, A. D.; Milano, G. Weak interactions between poly(ether imide) and carbon dioxide: A multiscale investigation combining experiments, theory and simulations. *Macromolecules* **2022**, *55*, 10773–10787.

(26) Mohottalalage, S. S.; Senanayake, M.; Clemmer, J. T.; Perahia, D.; Grest, G. S.; O'Connor, T. C. Nonlinear elongation flows in associating polymer melts: From homogeneous to heterogeneous flow. *Phys. Rev. X* **2022**, *12*, 021024.

(27) Wang, J.; O'Connor, T. C.; Grest, G. S.; Ge, T. Superstretchable elastomer from crosslinked ring polymers. *Phys. Rev. Lett.* **2022**, *128*, 237801.

(28) Ziolek, R. M.; Santana-Bonilla, A.; Castro, R. L.-R. D.; K uhn, R.; Green, M.; Lorenz, C. D. Conformational heterogeneity and interchain percolation revealed in an amorphous conjugated polymer. *ACS Nano* **2022**, *16*, 14432–14442.

(29) Fortunato, M. E.; Colina, C. M. *pysimm*: A python package for simulation of molecular systems. *SoftwareX* **2017**, *6*, 7–12.

(30) Demidov, A. G.; Perera, B. L. A.; Fortunato, M. E.; Lin, S.; Colina, C. M. Update 1.1.

to "pysimm: A python package for simulation of molecular systems". *SoftwareX* **2021**, *15*, 100749.

(31) Girard, M.; Ehlen, A.; Shakya, A.; Bereau, T.; de la Cruz, M. O. Hoobas: A highly object-oriented builder for molecular dynamics. *Comput. Mater. Sci.* **2019**, *167*, 25–33.

(32) Cummings, P. T. et al. Open-source molecular modeling software in chemical engineering focusing on the Molecular Simulation Design Framework. *AIChE J.* **2021**, *67*, e17206.

(33) Yan, X.; Chaudhuri, S. An interactive polymer building toolkit for molecular dynamics simulations: PolyMAPS. *arXiv* **2022**, *2204*, 14218.

(34) Sahu, H.; Shen, K.-H.; Montoya, J. H.; Tran, H.; Ramprasad, R. Polymer Structure Predictor (PSP): A Python toolkit for predicting atomic-level structural models for a range of polymer geometries. *J. Chem. Theory Comput.* **2022**, *18*, 2737–2748.

(35) Vanommeslaeghe, K.; A. D.M̃acKerell, J. Automation of the CHARMM General Force-field (CGenFF) I: Bond perception and atom typing. *J. Chem. Inf. Model.* **2012**, *52*, 3144–3154.

(36) Wang, J.; Wolf, R. M.; Caldwell, J. W.; Kollman, P. A.; Case, D. A. Development and testing of a general Amber force field. *J. Comput. Chem.* **2004**, *25*, 1157–1174.

(37) Dodda, L. S.; de Vaca, I. C.; Tirado-Rives, J.; Jorgensen, W. L. LigParGen Web Server: An automatic OPLS-AA parameter generator for organic ligands. *Nucleic Acids Res.* **2017**, *45*, W331–W336.

(38) Plimpton, S. J. Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.* **1995**, *117*, 1–19.

(39) Wolf, C. M.; Guio, L.; Scheiwiller, S.; Pakhnyuk, V.; Luscombe, C.; Pozzo, L. D.

Strategies for the development of conjuated polymer molecular dynamics force fields validated with neutron and x-ray scattering. *ACS Polym. Au* **2021**, *1*, 134–152.

(40) RDKit: Open-source cheminformatics. `https://www.rdkit.org`.

(41) You, Y.; Chen, J.; Zheng, A.; Wei, D.; Xu, X.; Guan, Y. Effect of silanol on the thermal stability of poly [methyl (trifluoropropyl) siloxane]. *J. Appl. Polym. Sci.* **2020**, *137*, 49347.

(42) Liu, Y.; Chaiprasert, T.; Ouali, A.; Unno, M. Well-defined cyclic silanol derivatives. *Dalton Trans.* **2022**,

(43) Bannwarth, C.; Ehlert, S.; Grimme, S. GFN2-xTB—An accurate and broadly parametrized self-consistent tight-binding quantum chemical method with multipole electrostatics and density-dependent dispersion contributions. *J. Chem. Theory Comput.* **2019**, *15*, 1652–1671.

(44) Sun, Q.; Zhang, X.; Banerjee, S.; Bao, P.; Barbry, M.; Blunt, N. S.; Bogdanov, N. A.; Booth, G. H.; Chen, J.; Cui, Z.-H., et al. Recent developments in the PySCF program package. *J. Chem. Phys.* **2020**, *153*, 024109.

(45) Gutman, I.; Estrada, E. Topological indices based on the line graph of the molecular graph. *J. Chem. Inf. Comput. Sci.* **1996**, *36*, 541–543.

(46) Hanser, T.; Jauffret, P.; Kaufmann, G. A new algorithm for exhaustive ring perception in a molecular graph. *J. Chem. Inf. Comput. Sci.* **1996**, *36*, 1146–1152.

(47) Hagberg, A.; Conway, D. NetworkX: Network Analysis with Python. *URL: https://networkx. github. io* **2020**,

(48) Cheng, F.; Jäkle, F. Boron-containing polymers as versatile building blocks for functional nanostructured materials. *Polym. Chem.* **2011**, *2*, 2122–2132.

(49) Kwon, M. Diagrams: Diagrams as code. https://diagrams.mingrammer.com/, 2022.

(50) J. Fonseca, e. gprof2dot. https://github.com/jrfonseca/gprof2dot, 2022.

(51) King's College London (2022). King's Computational Research, Engineering and Technology Environment (CREATE). https://doi.org/10.18742/rnvf-m076.

TOC Graphic