Computational Design of Peptides and Proteins for Cyclic Peptide Applications

Meerit Said

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington

2023

Reading Committee:

David Baker, Chair

Justin Kollman

Dustin Maly

Program Authorized to Offer Degree:

Biochemistry

University of Washington

Abstract

Computational Design of Peptides and Proteins for Cyclic Peptide Applications

Meerit Said

Chair of the Supervisory Committee:
David Baker
Department of Biochemistry

Cyclic peptides are a fascinating class of compounds to use in multiple design applications due to their small size and ability to be chemically synthesized. Non-canonical amino acids can be easily incorporated into cyclic peptides to expand their chemistries and make them resistant to proteolysis. Additionally, cyclic peptides are modular and can be computationally designed to adopt specific membrane permeable conformations. During my PhD, I used computationally designed cyclic peptides for two main applications, 1) designing peptide metal-organic frameworks, and 2) designing peptide inducible dimeric proteins.

First, I explored the use of designed symmetric cyclic peptides to generate new metal-organic frameworks (MOFs). MOFs have shown great potential in a wide variety of applications such as small molecule separation, drug delivery, and catalysis. However, these materials have generally been limited to using small molecule linkers and short two to four

residue peptides. I aimed to computationally design peptide MOFs using symmetric cyclic peptides. Peptide based MOFs provide advantages since they can bind specifically and selectively to different substrates and can be used for catalysis. We used the Rosetta Software Suit to dock and design peptides into metal mediated 3D lattices, then experimentally screened them for crystal formation. We solved the structures of six peptide materials using single crystal X-ray diffraction. Although these structures do not match the design models, they demonstrate fundamental thermodynamic and kinetic rules that govern the formation of such materials. Our computational pipeline is the first step to computationally design peptide based metal-organic frameworks and our experimental data provides information to further refine the computational protocol for more accurate modeling.

Second, to gain insight into the effects of protein oligomerization on cellular functions, I employed cyclic peptides to induce the formation of protein oligomers. Most chemically inducible dimeric systems in the literature use clinically approved drugs to induce protein oligomerization. However, these drugs have off-target effects which makes the results ambiguous. In order to make CID systems that are orthogonal to cellular machinery, I used computationally designed *de novo* proteins and *de novo* cyclic peptides to make CIDs. I used Rosetta, Alphafold2, and ProteinMPNN to design and filter peptide binding proteins. The binding was validated using isothermal calorimetry, and equilibrium dialysis leading to the identification of multiple nanomolar and micromolar peptide binders. Finally, we can use the NanoBiT assay to optimize peptide-dependent protein oligomerization.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

First, I would like to thank my PhD mentor Dr. David Baker for his support and guidance. His enthusiasm and humility are always astounding to me. I'm especially grateful for all the feedback he has given me and his openness to take feedback in return. I would also like to thank my committee members Dr. Dustin Maly, Dr. Justin Kollman, Dr. Kim Woodrow, and Dr. Gojko Lalic. Dustin Maly and Justin Kollman are incredible rotation mentors whose feedback and advice were invaluable throughout my PhD. I would like to thank my undergraduate mentors without whom I would not have been able to go to graduate school. Thank you to Dr. Cynthia Chang who introduced me to research and is always inspiring to watch. Thanks to Dr. Lori Robins who introduced me to biochemistry. Her enthusiasm for the topic is why I became interested in biochemistry. Thanks to Dr. Nick Cox who taught me about peptides and guided me through the graduate school applications process.

The Baker lab is a large place that can be hard to navigate. I'm privileged to have had the best mentors in the lab, Dr. Christine Kang and Dr. Patrick Salveson. The impact they had on my personal and professional growth is indescribable. Christine's intelligence and wisdom are inspiring. I hope to one day be one tenth as wise at communicating with people and handling difficult situations as she is. Patrick is incredibly knowledgeable and enthusiastic about science. I learned how to communicate in a team and in presentations from him. They both made working in the lab genuinely fun! I would not have made it this far without their support and advice.

In addition to my main mentors in the lab, I was lucky to interact with many fantastic scientists in the lab. The peptide subgroup was my home in the lab for many years. Thank you to Dr. Parisa Hosseinzadeh, Dr. Gaurav Bhardwaj, Dr. Kris Deibler, Dr. Jacob O'Connor, Dr. Adam Moyer, and Dr. Pat Erickson. The peptide subgroup is the best subgroup thanks to all of these

people. As my PhD shifted to protein design, I had the pleasure of interacting with scientists outside of the peptide subgroup. Everybody in the lab is helpful and generous with their time. There are too many people to list, but I would like to especially thank Dr. Gyu Rie Lee, Dr. Derrick Hicks, Dr. Harley Pyles, Dr. Sherry Bermeo, Dr. Robby Divine, Dr. Stacey Gerben, Dr. Nate Ennist, Dr. Chris Norn, Dr. Sam Pellock, Dr. Linna An, and Susana Vazquez Torres. I would also like to thank everybody in the core groups who supported this work. Thank you to Dr. Inna Goreshnik, Dr. Asim Bera, Alex Kang, Hannah Nguyen, Ryanne Ballard, and Xinting Li. Thank you to everybody in the admin team at the IPD, especially Dr. Lance Stewart and Zari Magness.

Thank you to Rocky and Genie Higgins for their generous financial and moral support during my PhD. This work was supported with funds provided by the Audacious Project at the Institute for Protein Design (D.B., A.B., A.K., L.S., R.B.); the Juvenile Diabetes Research Foundation International grant # 2-SRA-2018-605-Q-R (D.B., P.J.S.), the Helmsley Charitable Trust Type 1 Diabetes Program Grant # 2019PG-T1D026 (D.B., X.L.); the Nordstrom Barrier Institute for Protein Design Directors Fund (P.J.S.); the Wu Tsai Translational Fund (P.J.S.), the Open Philanthropy Project Improving Protein Design Fund (D.B., A.B., H.N.), the Defense Threat Reduction Agency grant HDTRA1-19-1-0003 (D.B., M.Y.S., C.K., S.W.), the Higgins family (M.Y.S.), and the Howard Hughes Medical Institute (D.B., W.S.). We thank Dr. Brandi M. Cossairt, Dr. Andrew Ritchhart, Helen Larson, Dr. Tamir Gonen, Dr. Samantha Young, and Dr. Sara Weaver for their support and discussions. Crystallographic work is based upon research conducted at the Northeastern Collaborative Access Team beamlines, which are funded by the National Institute of General Medical Sciences from the National Institutes of Health (P30 GM124165). Part of this work was conducted at the Molecular Analysis Facility, a National

Nanotechnology Coordinated Infrastructure (NNCI) site at the University of Washington, which is supported in part by funds from the National Science Foundation (awards NNCI-2025489, NNCI1542101), the Molecular Engineering & Sciences Institute, and the Clean Energy Institute. This research used resources of the Advanced Photon Source, a U.S. Department of Energy (DOE) Office of Science User Facility operated for the DOE Office of Science by Argonne National Laboratory under contract no. DE-AC02-06CH11357.

I would like to thank the many friends I have made inside and outside of the lab, Naveen Jasti, Aditya Krishnakumar, Sidney Lisanza, Jeremiah Sims, Adam Chazin-Gray, and Fatima Davila. I would like to especially thank Sanaa Mansoor for being an incredible friend and always sharing fruit with me. Thank you to Ian Humphreys for always sharing his wrong opinions with me.

Thank you to all of my aunts, uncles, cousins and my grandma. Thank you to the Egyptian community that kept me grounded outside of research. I would like to especially thank Fr. Mina Salama and Fr. Danial Zaki for their kindness and support.

Finally, I would like to thank my parents, Feby Hanna and Yoakim Shenoda. Their kindness and support are unparalleled. They constantly keep me sane and motivated. I have the best parents and will forever be grateful to them.

DEDICATION

To my mom and dad.

Chapter 1: Introduction to cyclic peptides and their applications

From complex assemblies to catalytic reactions, proteins have evolved to perform many complicated tasks in biological systems. Over the past two decades, efforts in expanding available protein structures through *de novo* protein design have successfully generated new scaffolds used for assemblies,[1,2] logic,[3,4] and enzymology.[5,6] However, since proteins are recombinantly expressed in living cells, they are still mostly limited to the 20 canonical amino acids, with the ability to add a few non-canonical amino acids at once,[7] which limits the chemistry and backbone diversity available to these proteins.

Peptides provide many advantages unavailable to proteins. Due to their small size compared to proteins, they can be chemically synthesized to include both canonical and non-canonical amino acids. This expanded set of building blocks makes peptides less prone to proteolysis due to changes in the amino acid backbones which are not recognizable by proteases.[8] The use of non-canonical amino acids with diverse side chain chemistries also enhances the peptides' abilities to perform different tasks such as metal coordination and catalysis.[9] Finally, peptides can be used in conditions unavailable to proteins such as organic solvents and high temperatures without losing their stability. [10]

Cyclic peptides are a particularly interesting class of compounds. The cyclization leads to a limited set of conformations available, higher protease stability, and passive membrane permeability.[11] The Rosetta Software Suite has been shown to successfully model structured cyclic peptides by incorporating backbone constrained amino acids such as proline, D-proline, and 2- Aminoisobutyric acid and backbone hydrogen bonding.[10] Then, it samples hundreds of thousands of peptide conformations and predicts the lowest energy structure using Rosetta's

energy function. Predicted lowest energy conformations match NMR and crystal structures with low backbone RMSD.

Cyclic peptides' structural diversity, stability, and modularity makes them excellent compounds to generate self assembling materials and use as therapeutics.[12,13] In my PhD, I worked on using *de novo* designed cyclic peptides to build peptide metal-organic frameworks and design peptide based chemically inducible dimeric proteins (Figure 1). First, peptides' inherent chirality and surface area makes them attractive compounds to use in materials such as metal-organic frameworks. Their size and designability gives them the potential to bind substrates for functions such as small molecule binding and catalysis. Using Rosetta based computational modeling, we generated thousands of models of symmetric cyclic peptides in 3D metal mediated lattices. We empirically tested 48 peptides for crystal formation in the presence of transition metals and solved the structures of six peptide metal-organic frameworks to atomic resolution.[14] These structures elucidate the importance of dispersion interactions in the formation of peptide MOFs and greatly expand the available MOF ligands to include the cyclic peptide class of compounds.

Second, I worked on designing cyclic peptide binding proteins to generate *de novo* chemically inducible dimers (CIDs). The peptides' designability, orthogonality, and membrane permeability makes them attractive chemical inducers as opposed to clinically approved drugs. These CID systems can be used to examine signaling pathways with few off-target effects since they are *de novo* designed to be orthogonal to natural systems. To make *de novo* CIDs, I used Rosetta to redesign homodimeric proteins to bind C2 symmetric peptides. Then, I filtered models of binders using AlphaFold2 and Rosetta interface filtering.[15] Cyclic peptide binding was experimentally tested using equilibrium dialysis and isothermal calorimetry. This led to the

validation of multiple peptide binding proteins with nanomolar to micromolar affinity. The strongest peptide binding protein bound its respective cyclic peptide with 30nM Kd. Additionally, we solved the crystal structure of this peptide-protein complex which showed close agreement to the designed model with 0.9 Å Cα RMSD. We are currently mutating the homodimeric protein interface to optimize for peptide dependent complex formation.

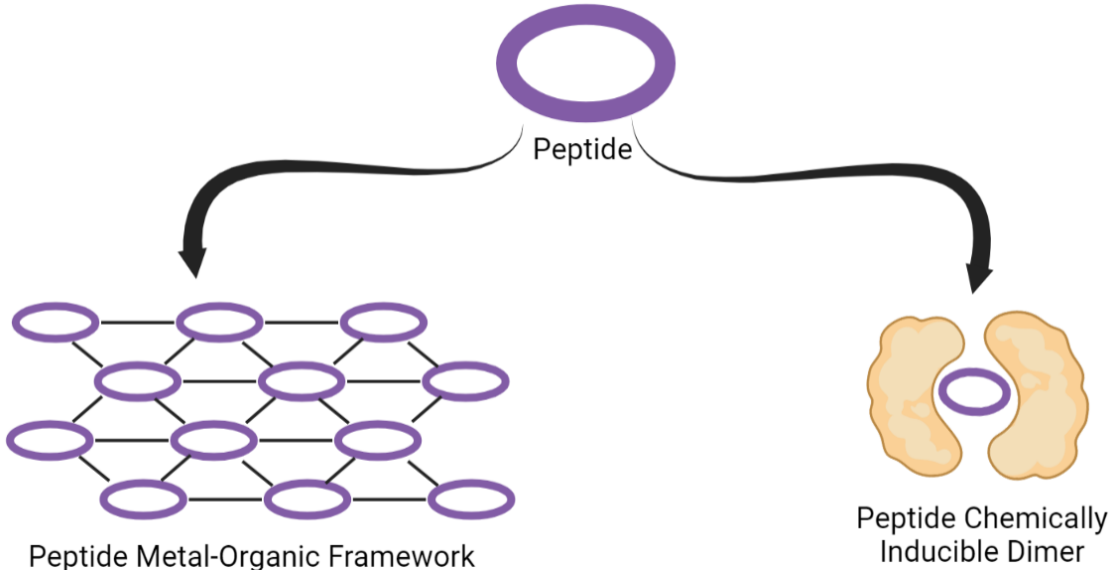


Figure 1. Computationally designed structured cyclic peptides are redesigned to form metal-organic frameworks and chemically inducible dimeric proteins.

Chapter 2: Design of peptide metal-organic frameworks

This chapter contains additional background (and limited reproduction of) content previously published as: Said, M. Y.; Kang, C. S.; Wang, S.; Sheffler, W.; Salveson, P. J.; Bera, A. K.; Kang, A.; Nguyen, H.; Ballard, R.; Li, X.; Bai, H.; Stewart, L.; Levine, P.; Baker, D. Exploration of Structured Symmetric Cyclic Peptides as Ligands for Metal-Organic Frameworks. *Chem. Mater.* 2022, **34** (21), 9736–9744.

2.1 Introduction

Guided by a set of topological and chemical principles, combinations of organic ligands and metals have been used to engineer crystals formed from a wide variety of coordination polymers, including metal−organic frameworks (MOFs).[16–21] Recently developed chemically tailorable organic linkers enable the modification of the pore geometry and internal surface chemistry of open-framework materials for applications such as adsorption, separation, and catalysis.[22–28] The most commonly explored MOF ligands utilize rigid conjugated aromatic linkers,[29,30] while the development of such materials using larger flexible ligands remains more limited. Peptidic ligands have recently emerged as an attractive class of MOF building blocks due to their intrinsic chirality, structural modularity, and biocompatibility.[12,31–34] However, the majority of reported metal-peptide frameworks to date involve short linear peptides (e.g., di- and tripeptides) identified through the large-scale experimental screening.[35–38] Use of longer peptides as organic linkers in this way has been challenging because of their greater conformational flexibility.

We previously demonstrated that Rosetta computational design methods could be used to design cyclic peptides with internally symmetric sequences and structures.[39] The crystal

structures of nine C2, C3, and S2 peptides were very close to the computational design models. Here, we set out to explore the design of MOFs using these symmetric cyclic peptides with well-defined backbone structures as metal ligands. These compounds have potential advantages over previous peptide ligands as they are more rigid and have internal symmetry axes that can be aligned with crystal lattice symmetry axes, and hence we reasoned that materials generated using them should be more programmable. We aimed to design specific MOF lattices using geometrically compatible symmetric peptides and metal sites; the combination of two symmetry elements in defined orientations generates regularly repeating lattices.[40] We reasoned that by combining the internal symmetries of the cyclic peptide backbones and metal coordination centers through defined rotations, translations, and dihedral angles associated with the peptide sidechains coordinating the metal, a wide variety of crystal lattices could be generated. Experimental characterization of a series of symmetric cyclic peptide-metal systems, which involve some of the largest peptidic ligands reported to form crystalline coordination polymers with transition metals to date, reveals limitations in our starting assumptions and highlights a set of principles underlying the structures of such assemblies.

2.2 Experimental and computational methods

2.2.1 Symmetric Backbone Generation.

Backbone structures were generated for C3 and S2 symmetric macrocycles by sampling the backbone dihedral angles of the asymmetric unit (3 residues for a nine-residue C3 peptide) using kinematic closure to drive chain closure with internal symmetry as described in Mulligan *et al*.[39] C2 symmetric macrocycles were generated by systematically sampling the space of conformations for the asymmetric unit, computing the rigid body transformation associated with

these, and selecting those for which duplication generates a closed structure (i.e., those for which the angle of rotation around the symmetry axis is 180°). The energy of the designed peptide conformation was calculated using AIMNet.[41]

2.2.2 Metal-Mediated Crystal Lattice Design.

Peptide backbones generated as described above were placed into metal-mediated lattices by choosing a set of metal binding sidechains and metal coordination geometries, and then, for each choice, placing a metal binding sidechain at each position in the asymmetric unit and sampling the chi angles in 1-degree steps and analytically computing rotation around the sidechain-metal bond that produces the correct angle between the peptide and metal symmetry axes, resulting in a macrocycle with 2 (C2 and S2) or 3 (C3) metal coordinating residues. The crystal lattice is finally generated through the placement of additional copies of the macrocycle to fill out each metal coordination sphere. Lattices containing clashes between neighboring macrocycle backbones were removed, and the amino acids not involved in the metal coordination designed using Rosetta (Listing S7) to favor the internal geometry of the macrocycle, the packing interactions between macrocycles, and the positioning of the metal coordinating residues. The resulting designed crystal lattices were filtered based on density (calculated using the script in Listing S9).

This design approach is similar in principle to that of King *et al*. and Hsia *et al*.,[42,43] wherein distinct symmetry elements are placed so they propagate into the desired assembly. A top-down approach was used by King *et al*., placing proteins with cyclic symmetry along the axes of the target cage symmetry, for example, C4 and C3 at the faces and corners of a cube, then sampling the rotations and translations along these axes that preserve symmetry.[42] A bottom-up approach was used in Hsia *et al*., fusing proteins with cyclic symmetry through helical repeat

linker elements and searching for fusions which place the symmetry elements relative to each other to form a target symmetry, for example, forming a cube with C4 and C3 elements 54.7° apart such that the axes intersect.[43] The bottom-up approach we use here to design crystal lattices starting with symmetric peptides and searching for possible binding geometries that attach a symmetric metal coordination site goes beyond the previous approaches in several ways. First, here the relationship between symmetry elements is defined by rotamer and metal binding geometry rather than protein−protein interactions or backbone−backbone fusion. Second, we design three-dimensional crystal assemblies requiring more complex geometric criteria, precision, and careful alignment to the unit cell. Third, we employ small peptide scaffolds with D- and L-amino acids rather than large all L-proteins. Fourth, we considered D2 symmetry elements as well as cyclic elements. Consideration of D-amino acids and D2 symmetry elements expands the space of possible symmetric assemblies and metal binding geometries but is otherwise straightforward. Placement of symmetry elements to form 3D crystals requires higher precision than in other symmetric design tasks, as small errors can propagate much further in the assembly before self reinforcement. For example, C4 elements on the faces of a cube require only three steps to come back on itself, while a $P2_13$ crystal requires 10 steps.

To calculate the void volume in each crystal structure, water was removed from the structures (Figure S3) and then the percent void in a unit cell was calculated using Mercury's default settings as described in Macrae *et al*.[44]

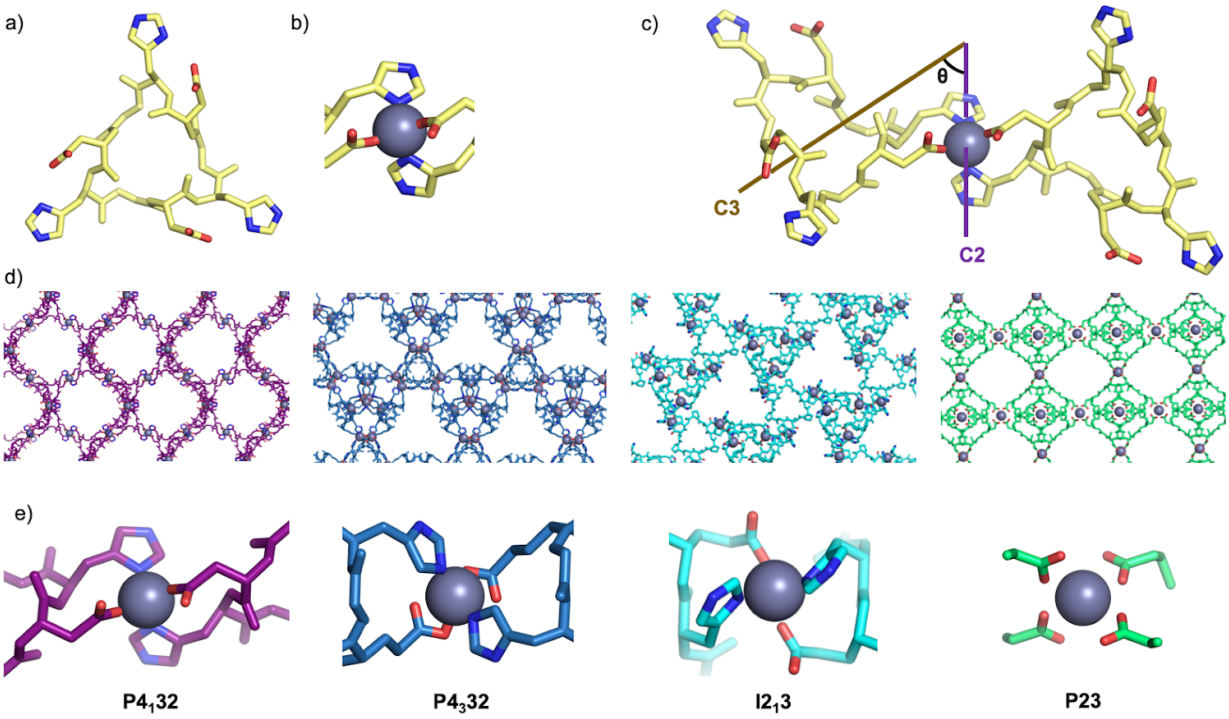


Figure 1. Computational method for designing metal mediated 3D lattices from rigid symmetric peptide building blocks. (a) Rotamers of metal coordinating residues such as histidines are sampled on symmetric peptide backbones. (b) Symmetric metal mediated interactions between pairs of peptides are sampled according to standard coordination geometry. (c) Peptide arrangements with dihedral angles between the axis of symmetry of the peptide and the axis of symmetry of the metal compatible with ideal lattice geometry for particular space groups are selected. (d) Examples of modeled lattices in space groups $P4_332$, $P4_132$, $P23$, and $I2_13$. (e) Close up view of the metal coordination for each lattice.

2.2.3 Peptide Synthesis and Purification.

All peptides were purchased from WuXi Apptec or synthesized in-house on a CEM Liberty Blue microwave synthesizer. All L- and D- amino acids were purchased from P3 Biosystems. Oxyma Pure was purchased from CEM, DIC was purchased from Oakwood Chemical, diisopropyl ethylamine (DIEA) and piperidine were purchased from SigmaAldrich. Dimethylformamide (DMF) was purchased from Fisher Scientific and treated with an Aldraamine trapping pack prior to use. Synthesis was done on a 0.1 mmol scale on CEM

Cl-TCP(Cl) resin. Five equivalents of each amino acid were activated using 0.1 M Oxyma with 2% (v/v) DIEA in DMF, 15.4% (v/v) DIC, and coupled on resin for 4 min with double coupling if needed. This was followed by deprotection using 5 mL of 20% piperidine in DMF for 2 min at 95 °C. Completed linear peptides were removed from resin while maintaining side chain protecting groups by 5 times 5 min incubations of the resin in 1% TFA in dichloromethane (DCM). The DCM was removed in vacuo and the protected peptides were subjected to lyophilization in a 1:1 water/acetonitrile (ACN) mixture. The protected peptides were resuspended in 70 mL of DCM in a 100 mL round bottom flask, treated with 1.1 equivalents of (7- azabenzotriazol-1-yloxy)tripyrrolidinophosphonium hexafluorophosphate (PyAOP), and stirred for 30 min before adding 0.2% (v/v) DIEA dropwise. The cyclization reaction proceeded for 16 h before removing DCM in vacuo and subjecting the peptide to a total deprotection solution consisting of TFA/H2O/DODT (3,6-dioxa1,8-octanedithiol)/triisopropylsilane (92.5:2.5:2.5:2.5) for 3 h. This deprotection mixture was precipitated in 30 mL of ice-cold ethyl ether, centrifuged and decanted, then washed twice more with fresh ether and dried under nitrogen to yield crude peptide for high pressure liquid chromatography (HPLC) purification.

The crude peptide was dried and dissolved in a mixture of ACN and water where the entire crude is soluble. This solution was purified on a C18 column in an Agilent HPLC instrument. A linear gradient of increasing ACN with 0.1% TFA was used to purify the samples. UV signal was monitored at 214 nm and all peaks were collected. Peaks were checked using ESI mass spectroscopy for the correct peptide mass. The purified peptide was then lyophilized for further use. All UPLC and mass spectra are included in the Supporting Information.

2.2.4 Crystal Screening.

Peptides were screened using 96 well plates using the conditions shown in Supporting Information Tables 1 and 2. Stocks of the peptides were made in water, methanol, acetonitrile, or DMF so that 1.25−5 mM are added to each well. The peptide samples were left to dry on the plate overnight, then 5 μL of the appropriate solvent was added to each well. Completed plates were incubated at 4°C overnight and then checked using a light microscope for crystal formation. If no crystals form, the plates would be placed in a convection oven at 80 °C. Conditions that grow crystalline material are optimized in polymerase chain reaction tubes through varying peptide concentration and solvent conditions. Once crystals formed, diffraction data were collected from a single crystal at synchrotron (on APS 24ID-C) and at 100 K. Unit cell refinement and data reduction were performed using the XDS and CCP4 suites.[45,46] The structure was identified by direct methods and refined by full-matrix least squares on F2 with anisotropic displacement parameters for the nonH atoms using SHELXL-2018/3.[47,48] Structure analysis was aided by using Coot/Shelxle.[49,50] The hydrogen atoms on heavy atoms were calculated in ideal positions with isotropic displacement parameters set to 1.2× Ueq of the attached atoms. Crystallographic structures were deposited into the Cambridge Structural Database (CSD), under deposition numbers 2160569 (C2-1), 2160570 (C2-2a), 2160571 (C2-2b), 2160572 (C3-1), 2160573 (C3-2), 2160589 (S2-1), and 2160766 (S2-2).

2.3 Results

We previously described a symmetric cyclic peptide design method that generates peptide sequences predicted to have single low-energy states with internal symmetry.[10,39] The method starts by generating large numbers of cyclic peptide backbones with internal symmetry, searches

for low energy sequences for these backbones, and then checks by folding simulations that the lowest energy conformation matches the designed conformation. In our previous work, we designed large numbers of such compounds *in silico*. We were able to solve the crystal structures of 12 of these and found that they were very close to the design models, including one peptide designed to switch from one conformation into another in the presence of zinc (both conformations were confirmed crystallographically).[39] To generate coordination polymers using these rigid symmetric structures as building blocks, we incorporated metal liganding amino acid side chains into the structures, confirming by *in silico* energy landscape mapping that the lowest energy predicted states were not affected by the amino acid substitutions (Figure S1).

We developed a computational method for docking and designing such symmetric cyclic peptides into crystal lattices with metal-mediated interfaces based on three simplifying assumptions. First, that the internal structures of the peptides would be maintained in the metal mediated crystal lattices; second, that the peptides would fully coordinate metals with preferred tetrahedral geometry such as Zn^{2+} ions; and third that all metal coordinating residues would be involved in the metal coordination (e.g., that peptides with one histidine and one aspartate residue would coordinate the metal in a two-His, two-Asp configuration, Figure 1a,b). We took a bottom-up approach, starting with symmetric peptides and searching through possible interaction geometries through symmetric metal coordination sites. The peptide and metal symmetry elements are placed relative to each other based on the coordinating residue position and rotamer, as well as the metal residue bond (for more detail, see Experimental Section). To form a 3D crystal, the axes of the component symmetry elements must be placed at precise dihedral angles (Figure 1c). In cases where the peptide to metal connection is a single residue, the metal-residue bond can be rotated to form the correct dihedral angle between the axis of symmetry of the

peptide and the axis of symmetry of the metal site. In other cases, the correct dihedral angle must be screened for (Figure 1c). We also considered a two-residue bidentate ASP-HIS binding motif, forming an overall C2 symmetric metal site around a tetrahedral metal center (Figure 1e). ASP-HIS pairs were precomputed and indexed, then superimposed on the peptide scaffolds. In this case, there is no rotatable metal-peptide bond, so not all structures have the appropriate dihedral angle between symmetry elements, and many must be discarded. To generate a crystal lattice in a specific crystal space group, in addition to component symmetry elements forming the correct dihedral angle between their axes, they must be placed at specific locations within the crystal unit cell, and there must not be clashes between symmetrically related copies; evaluating these properties is lattice-dependent. In the case of a C3 peptide and a C3 metal center, a $P2_13$ crystal can be formed with one C3 axis along [1,1,1] and intersecting the origin, and the other along [1,1,-1] and intersecting the [0,1,0] axis. The cell dimension is, in this case, defined by the distance from the origin to the [0,1,0] intersection. In the case of a C3 peptide and a bidentate C2 binding site, an $I2_13$ crystal can be formed in a similar manner. In the case of a C3 peptide and a tetrahedral metal site, the fully coordinated site has D2 local symmetry and can form a $P23$ crystal by placing the D2 element axis-aligned with the center along the [1,1,0] axis. The C3 element is aligned to [1,1,1], and the system scales such that the C3 axis intersects the $[\bar{2},1,1]$ axis. In this third case, the placement of the D2 and C3 elements each imply a unit cell dimension, and only systems where these cell dimensions agree are valid. This pipeline produced models in the $P4_332$, $P4_132$, $P23$, and $I2_13$ space groups (Figure 1d,e), which were designed using Rosetta (Supporting Information methods). Designed lattices with very low energies (as computed by Rosetta), cell dimensions less than 50 Å, and an approximate solvent fraction of less than 0.8 were selected for further analysis.[51,52]

To increase the diversity of structures that could be generated, in addition to the cyclic peptides whose structures were previously determined crystallographically (in the absence of metals), we included as potential building blocks the larger *in silico* set of designs predicted to adopt low-energy symmetric states. We selected 48 C3 peptide crystals generated from these compounds in the $I2_13$, $P23$, $P4_132$, and $P4_332$ space groups with Zn^{2+} as a metal-ligand for crystal assembly.[53] The cyclic peptide ligands were synthesized in-house using the previously described methods or obtained from WuXi AppTec.[39] To sample a wide condition space for crystallization and reduce the mass of peptide required for each individual reaction, we performed high-throughput screening experiments in 5 μL of volume using 96-well plates. In a typical experiment, 1 to 2.5 mM peptide was mixed with a metal source [e.g., $Zn(NO_3)_2$, $Fe(NO_3)_3$, $Cu(NO_3)_2$, or $Co(NO_3)_2$] at various molar ratios, in the presence of aqueous buffer solution (HEPES pH 7.0−8.5 or MES pH 5.0−7.0), or mixtures of organic solvents (DMF, DEF, MeOH, EtOH, and/or ACN) (Tables S1 and S2). The reaction mixtures were sealed and reacted for 24−48 h at either room temperature or at an elevated temperature (e.g., 65 or 80 °C) in a convection oven.

Crystallization studies reveal that many of the designed peptides formed aggregates in the presence of metals. However, two peptides crystallized, but the structures could not be solved due to their low resolution (Table S3 and Figure S4). We were able to solve the structure of one peptide C3-1 (EhPEhPEhP, Figure 2a), which in the designed crystal lattice ($P4_332$ space group) was intended to coordinate tetrahedral metals such as zinc with histidines and glutamates (Figure 2b). We were unable to crystallize the peptide using $Zn(OAc)_2$, $Zn(NO_3)_2$, or $ZnCl_2$, but in the presence of $Co(NO_3)_2$ in HEPES pH 8.2, crystals grew in the $P65$ space group over 4 weeks at room temperature (Figure 2d), and we were able to solve the structure at 0.86 Å resolution. The

peptide backbone conformation matches the design with a Cα root mean square deviation (RMSD) of 0.59 Å (Figure 2c). However, in the design model, the metal ion is fully coordinated by the glutamic acids and histidines, while in the crystal structure each Co^{2+} cation is octahedrally coordinated to three water molecules and three histidines from different peptides in a planar fashion (Figure 2b), and the glutamates do not participate in coordination but fill the crystal pores. This coordination geometry leads to the formation of 2D planes with 3-fold symmetry (Figure S6a), which stack at a 60-degree offset angle (Figure S6b) along the c-axis to form a 6-layer repeat unit (Figure S6b, teal dashed lines). The 3D lattice is stabilized by dispersion interactions and hydrogen bonding between the peptide planes and is more dense than the design model (void volume of 40% compared to 91%, see Experimental Section for void calculations).[44] Thus, while the internal conformation of the peptide matches the design model, the interactions between peptides are quite different than in the design model, with favorable peptide−peptide interactions outweighing the energetic gain from full metal coordination. These results suggest that our assumption that the lowest energy states would involve full metal coordination may not hold generally.

To gain further insight into the balance between peptide−peptide and peptide-metal interactions in determining MOF structures, we carried out a bottom-up exploration of peptides with variable symmetries (C2, C3, and S2), incorporated non-canonical metal coordinating residues [3-(4-pyridyl)-alanine, DOPA, or 4-carboxy-phenylalanine], and generated five additional structures, which we describe in the following sections.

A nine-residue peptide (DhmDhmDhm, C3-2, Figure 3a), crystallized in the $P4_12_12$ space group in the presence of 1 equivalent of $Zn(NO_3)_2$ in MES pH 6, at 80 °C for 24 h (Figure 3c). In contrast to the C3-1 crystal, in which the peptide conformation was nearly identical to the design

model, the C3-2 peptide conformation in the metal-mediated crystal is different from the original design model. This is due to a change in the torsional angle of the coordinating histidine (Figure 3b); such metal-induced changes have been observed previously.[39,54] The backbone conformation is still C3 symmetric, but the side chain rotamers are not symmetric. The zinc ion is internally coordinated with three histidines from one peptide and aspartic acid from an adjacent peptide. The crystal is composed of 1D metal-mediated peptide chains that intercross to form a dense 3D lattice (18% calculated void volume). In the crystal, two peptide-metal chains are intertwined via dispersion interactions (Figure S7a), and the other uncoordinated aspartic acid side chains form polar interactions with the peptide backbones (Figure S7a, purple dashed circle). Such interactions are also observed with the poly-proline-containing peptide-metal frameworks synthesized by Schnitzer and colleagues.[31]

To reduce the chance of backbone conformational changes and to explore a broader range of geometries and metal coordination ligands, we used a geometric hashing approach to design two pyridine-containing 6-mer peptides with AIMNet ground states having C2 symmetry (Figure S1a,b) and were able to obtain crystals with metal in multiple conditions after heating at 80 °C for 2 days. The structures of the crystals formed with 1 equivalent $Zn(NO_3)_2$ are shown in Figure 4. C2-1 (Figure 4a) formed crystals in the presence of HEPES pH 7.5 in the *P*1 (Figure 4c) space group, and C2-2 (Figure 4d) formed crystals in both the $C121/P12_11$ space groups (Figure 4f). In both cases, crystallization was driven by Zn pyridine interactions, which formed 1D metal-peptide chains (Figure S8a,b) that hierarchically thread into 3D crystals.

In the lattice formed by the C2-1 ligand [3-(4-pyridyl)-alanine−β-homoproline−α-aminobutyric acid−3-(4-pyridyl)-alanine−β-homoproline−

α-aminobutyric acid], each zinc ion is linked to two peptides through pyridine coordination, while two water molecules fill the other positions for full tetrahedral coordination (Figure 4c). The resulting peptide chains form 3D crystals through peptide stacking that is mediated by dispersion interactions and hydrogen bonding with participating water molecules. The 1D metal-peptide coordination chains grow along two different directions and intersect with each other, tiling the ab plane; non-covalent interactions mediate the stacking of these layers into 3D crystals. Water filled pores between the coordination chains make up 25% of the calculated unit cell volume. The internal hydrogen bonds in the peptide design model (Figure 4a) are broken in the crystal (Figure 4b); AIMNet calculates the crystal conformation to be 4.7 kcal/mol higher in energy than the designed conformation, suggesting that the lattice stabilizes the higher energy state (Figure S1; we cannot exclude the possibility that the AIMNet calculations are incorrect, but given the large energy difference, and the low expected error (1.1 kcal/mol)[41], crystal packing interactions are likely to distort the monomer ground state).

The crystal structure of peptide C2-2 (3-(4-pyridyl)-alanine−1,2,3,4-tetrahydroisoquinoline-3-carboxylic acid−3-aminobutanoic acid−3-(4-pyridyl)-alanine−1,2,3,4-tetrahydroisoquinoline-3-carboxylic acid−3-aminobutanoic acid) in the absence of metal in methanol matches that of the design model (Figure S2). However, crystal structures in the presence of Zn^{2+} reveal a different peptide conformation 3.4 kcal/mol higher than the designed conformation according to AIMNet (Figure 4d,e). The first crystal ($P12_11$ space group, Figure 4ftop) formed in HEPES pH 8.0 and 2% PEG2000 has a void volume of 35.3%. The second crystal ($C121$ space group, Figure 4f-bottom) formed in HEPES pH 8.0 and 2% PEP and has a void volume of 16%. As in the C2-1 case, in both crystals, the Zn^{2+} ions are tetrahedrally coordinated with two pyridine ligands and two water molecules and form 1D chains (Figure 4f),

but the packing is slightly different. For the *C*121 crystal, 1D chains first arrange in a parallel fashion into bilayer planes, which then stack to form the 3D crystal lattice. The $P12_11$ crystal shares an identical peptide-zinc coordination configuration, but the 1D chains stack at different angles between the adjacent 2D planes. Thus, for this peptide, metal mediated interactions generate 1D chains, and higher order structures such as 2D planes and 3D crystals are stabilized by dispersion interactions through extensive interchain peptide−peptide packing.

We next explored metal-mediated crystals built from achiral S2 symmetric peptides. These peptides have a two-fold improper rotation across their axis of symmetry, allowing access to centrosymmetric space groups, which increases the likelihood of crystallization.[55] Crystal structures determined in the absence of metal are very close to the design models.[39]

S2-1 (ppKvEPPkVe), is a 10 residue S2 symmetric cyclic peptide containing one lysine and one glutamate per asymmetric unit (Figure 5a). The apo structure matches the design to 0.53 Å RMSD. S2-1 formed crystals upon heating at 80 °C for 24 h in DMF with one equivalent of $Cu(NO_3)_2$ in the $P\bar{1}$space group (Figure 5b) with very small pores making up 7% of the unit cell volume. A single Cu^{2+} is coordinated between two peptides via two lysines and two glutamates in a square planar geometry (Figure 5b). Each peptide forms a bidentate interaction with two Cu^{2+} ions that assemble into a crystal through peptide backbone hydrogen bonding. Despite the copper coordination, the peptide backbone conformation matches that of the design and the apo structure. The crystal lattice also matches that of the apo crystal with an expansion of a and b axis by 1 Å each to allow for metal incorporation into the structure.[39]

The 12 residue S2-2 peptide (aNkhPeAnKHpE, Figure 5c) contains one lysine, one histidine, and one glutamic acid per asymmetric unit available for metal coordination. The apo structure of this peptide matches the design to 0.43 Å RMSD.[39] In the presence of 1 equivalent of

$ZnCl_2$, S2-2 crystallizes in isopropanol at room temperature in the *R*3 space group (Figure 5d). In the crystal structure, peptide−peptide interactions mediate crystal packing, while the large open channels along the c axis make up 40% of the unit cell volume (Figure S3f), which are filled with water-coordinated Zn^{2+} ions (Figure 5d). A comparison of the structure of S2-2 obtained in the absence and presence of metals indicates that the addition of Zn^{2+} did not change the overall crystal packing, since the Zn^{2+} ions occupied empty open channels in the crystal and are not coordinated to any of the aforementioned metal-binding residues. The peptide−peptide interactions in this crystal lattice are evidently more favorable than the metal coordination in the crystal conditions screened.

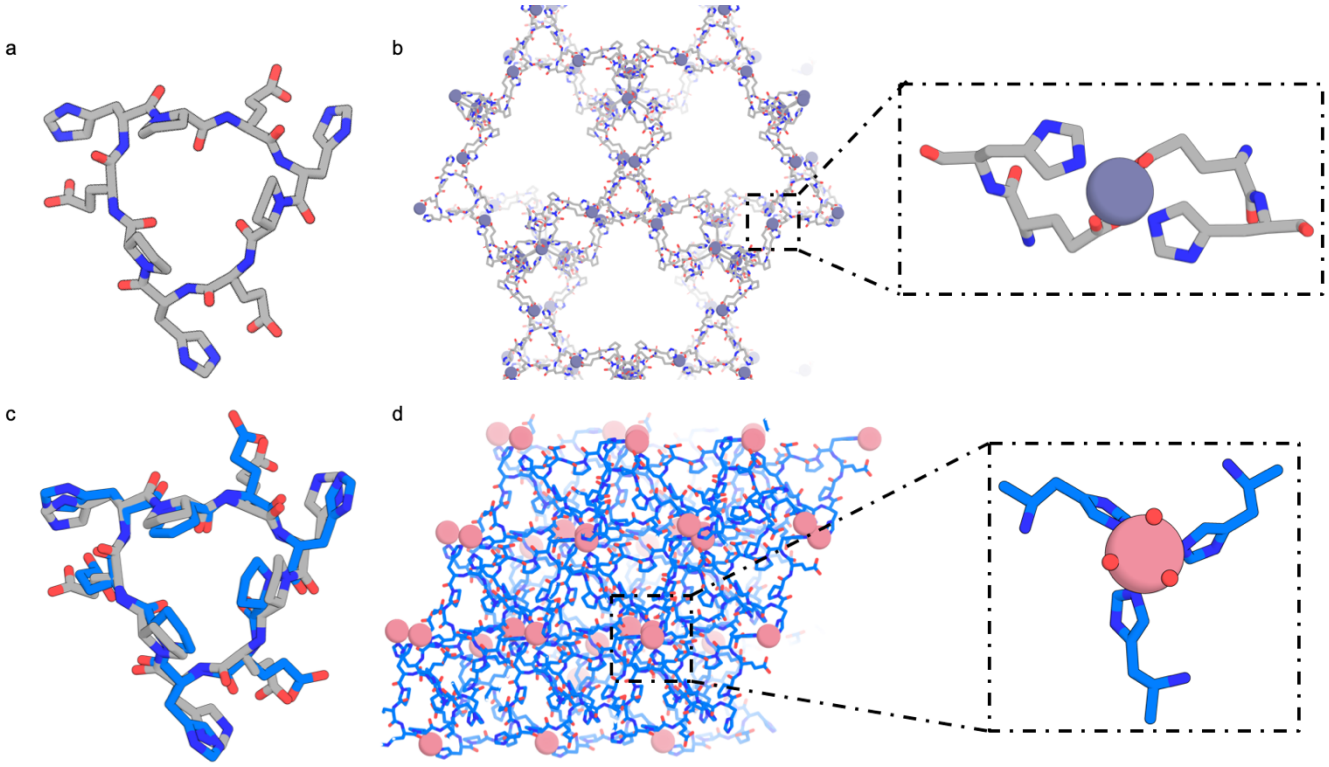


Figure 2. Structure of the C3-1 (EhPEhPEhP) - Co^{2+} crystal. (a) Computational design of the monomeric peptide. (b) Computational design of the peptide in $P4_332$ lattice. The inset shows the tetrahedral metal coordination in the design model. (c) Crystal structure of the C3-1 ligand (blue) aligned with the design model (gray) with a 0.59 Å Cα RMSD. (d) Crystal structure of C3-1 in *P*65 space group. The inset shows the three histidines and three water molecules coordination of C3-1 in the crystal structure.

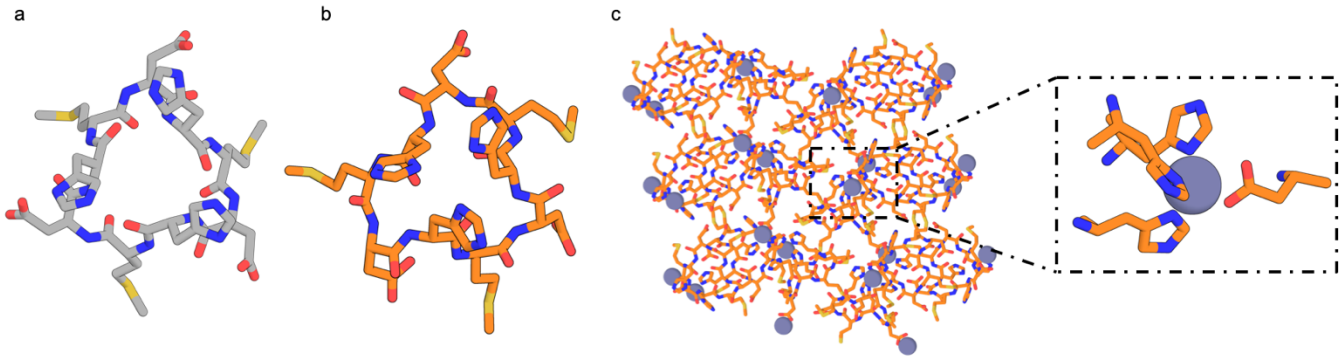


Figure 3. Structure of the C3-2 (DhmDhmDhm): Zn^{2+} crystal. (a) Design model of peptide C3-2. (b) Crystal structure of peptide C3-2. (c) Crystal structure of the peptide in $P4_12_12$ space group. Inset view shows tetrahedrally coordinated zinc in the crystal structure with three histidines from one peptide and one aspartate from an adjacent peptide.

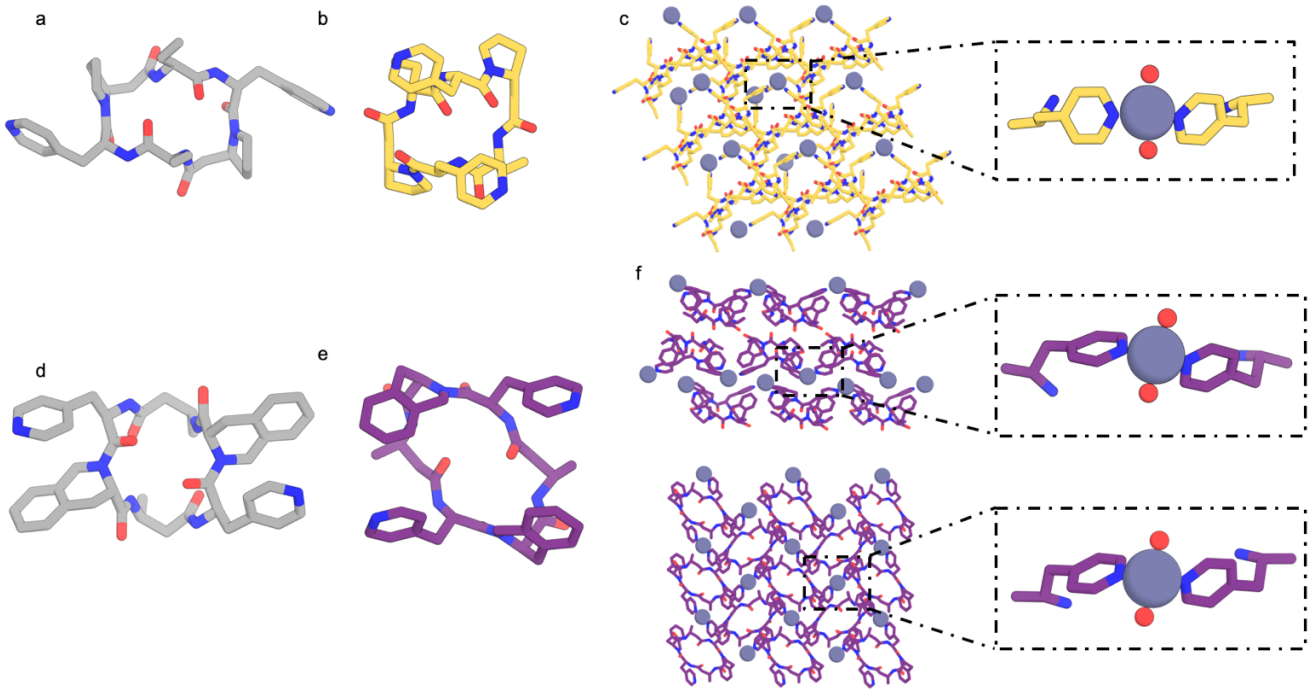


Figure 4. Structures of lattices formed by C2 pyridine-containing peptides. (a) Computational model of C2-1 [3-(4-pyridyl)-alanine−β-homoproline−α-aminobutyric acid–3-(4-pyridyl)-alanine−β-homoproline−α-aminobutyric acid] ligand. (b) Crystal structure of the C2-1 ligand. (c) Crystal structure of the peptide in the *P*1 space group. Inset view of the Zn^{2+}-C2-1 1D tetrahedral coordination using two pyridines and two waters. (d) Computational model of C2-2 [3-(4-pyridyl)-alanine–1,2,3,4-tetrahydroisoquinoline-3-carboxylic acid–3-aminobutanoic acid–3-(4-pyridyl)-alanine–1,2,3,4-tetrahydroisoquinoline-3-carboxylic acid–3-aminobutanoic acid] ligand. (e) Crystal structure of the C2-2 ligand. (f) Crystal structure of C2-2 in the $P12_11$ space group (top). Inset shows a zoomed view of the Zn^{2+}-C2-1 tetrahedral

coordination. The crystal structure of peptide C2-2 in the *C*121 space group is shown on the bottom. Inset is a zoomed view of the Zn^{2+}-C2-1 coordination.

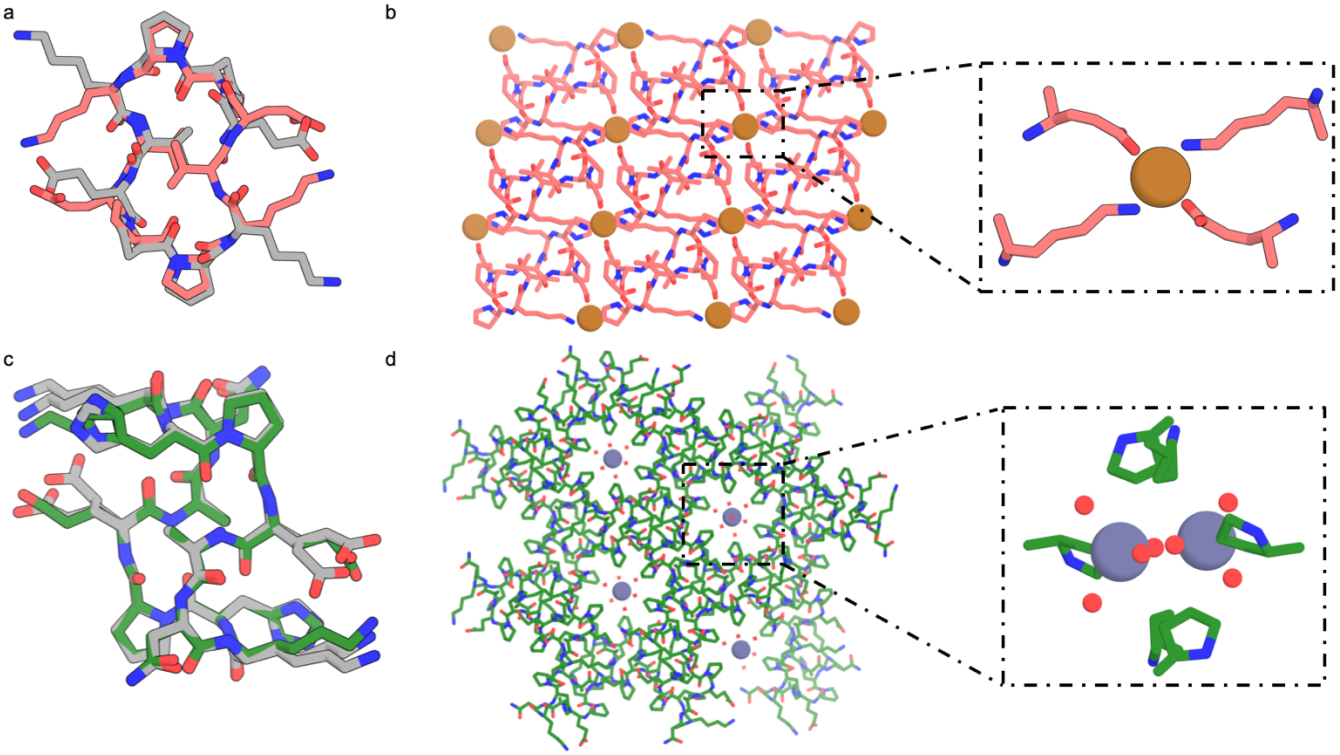


Figure 5. Lattices formed by cyclic peptides with S2 symmetry. (a) S2-1 (ppKvEPPkVe) ligand apo crystal (gray) overlaid with the metal coordinating crystal (pink) with a 0.53 Å Cα RMSD. (b) crystal structure of peptide S2-1 in *P*-1 space group. Inset shows a single Cu^{2+} ion coordinated with two peptides via two lysines and two glutamates in a square planar geometry. (c) S2-2 (aNkhPeAnKHpE) ligand apo crystal (gray) overlaid with the metal containing crystal (green) with a 0.43 Å Cα RMSD. (d) crystal structure of peptide S2-2 in *R*3 space group. Inset shows two zinc atoms coordinated by water molecules.

2.4 Discussion

Computational design provides a stringent test of the understanding of a physical or biological system: one formulates a set of hypotheses, implements a computational method based on these hypotheses, uses the method to design new molecular structures, and determines whether the experimental structures match the computational designs. Discrepancies between the computations and experimental data can then guide increases in understanding of the systems.

What can we learn from the discrepancies between the designs and the experiment observed here? While we designed and sought to crystallize 48 designs in the $I2_13$, $P23$, $P4_132$, and $P4_332$ space groups, only one crystallized and was in a different space group from the design. We cannot exclude the possibility that we did not find the appropriate crystallization conditions for the designs. But the simplest explanation, supported by the set of crystal structures we were able to obtain in our subsequent broad exploration of cyclic peptide ligands, is that the assumptions underlying our MOF design approach do not generally hold. First, we assumed that the designed peptide backbone conformation, in many cases supported by previous metal-free crystal structures, would be maintained in the metal mediated crystals. While this was true for some peptides, many adopted likely higher energy conformations stabilized by metal and dispersion crystal packing interactions. Second, we assumed that, as with smaller MOF ligands, the peptides would fully coordinate the metals with amino acid sidechains and that this coordination would drive the assembly into the crystal. Instead, we observed consistent partial coordination of the metals with water and direct peptide−peptide mediated crystal packing interactions. This likely occurs because our ligands are much larger than the aromatic small molecules commonly used in MOF synthesis and can pack against each other using multiple dispersion and hydrogen bonding interactions, which can outweigh purely metal-mediated interactions. Full metal coordination by peptide groups, while on its own more favorable than water coordination of the metals, is out-competed by dispersion and hydrogen bonding interactions between these large peptides in the crystal lattice. These observations suggest that for the successful design of cyclic peptide-mediated MOFs, it will be necessary to relax the fixed backbone assumption and allow backbone sampling along with rigid body and sidechain sampling, perhaps using approaches similar to the RIF docking approach used for designing protein−protein interactions.[56] It will also

be necessary to sample a wider range of different crystal packing arrangements, both those involving peptide metal coordination and those in which the primary crystal interactions are between the peptides, to more accurately determine whether the designed MOF is indeed the thermodynamically favorable structure. Development of improved computational design methods along these lines should enable a much more accurate design of macrocyclic based MOFs, which could have a wide variety of applications.

2.5 Conclusions

While MOFs have been previously described using short linear peptides, larger peptide ligands with internal symmetry have not, to our knowledge, been previously explored. We report the first structures of symmetric cyclic 6 to 12 residue peptide MOFs with both proper and improper symmetries (C2, C3, and S2), employing metal-chelation histidine, cysteine, aspartate, glutamate, and non-canonical amino acids containing pyridine and DOPA side chains. Our crystal structures of six peptide MOFs with different metals (Zn^{2+}, Co^{2+}, and Cu^{2+}) and space groups (*P*1, *P*65, *C*121, *P*12$_1$1, *R*3, *P*4$_1$2$_1$2, and *P*$\bar{1}$) contain a rich variety of 1D and 2D metal-mediated structures with pore shapes and sizes ranging from 7 to 40% void volume (some of these features have been observed in previous peptide-metal crystal structures, e.g., six residue poly-proline peptides can assemble into strings mediated by zinc and form dense frameworks through proline−proline packing).[31] The up to 12 residue cyclic peptide ligands studied here are to our knowledge, the largest peptidic ligands reported that form crystalline coordination polymers to date. An essentially unlimited number of rigid symmetric cyclic peptides can be designed using the methods described in Mulligan *et al*.,[39] and hence the crystal lattices described here are the first representatives of a very large class of new metal−organic crystals

that could provide new peptide materials for biocompatible, chiral, and catalytic applications. The large surface area and pore sizes of these peptide-metal lattices make them particularly interesting for downstream applications such as catalysis and sensing, and the wide variety of both natural and unnatural sidechains available allows facile customization of the chemistry lining the pores and other structural features of the crystals. The lattices frequently contain open metal coordination sites (Figure S5), around which substrate binding pockets could be built by further computational design, providing access to a new class of catalytic materials combining the features of MOFs and enzymes.

2.6 Supplemental information

2.6.1 Supplementary Figures

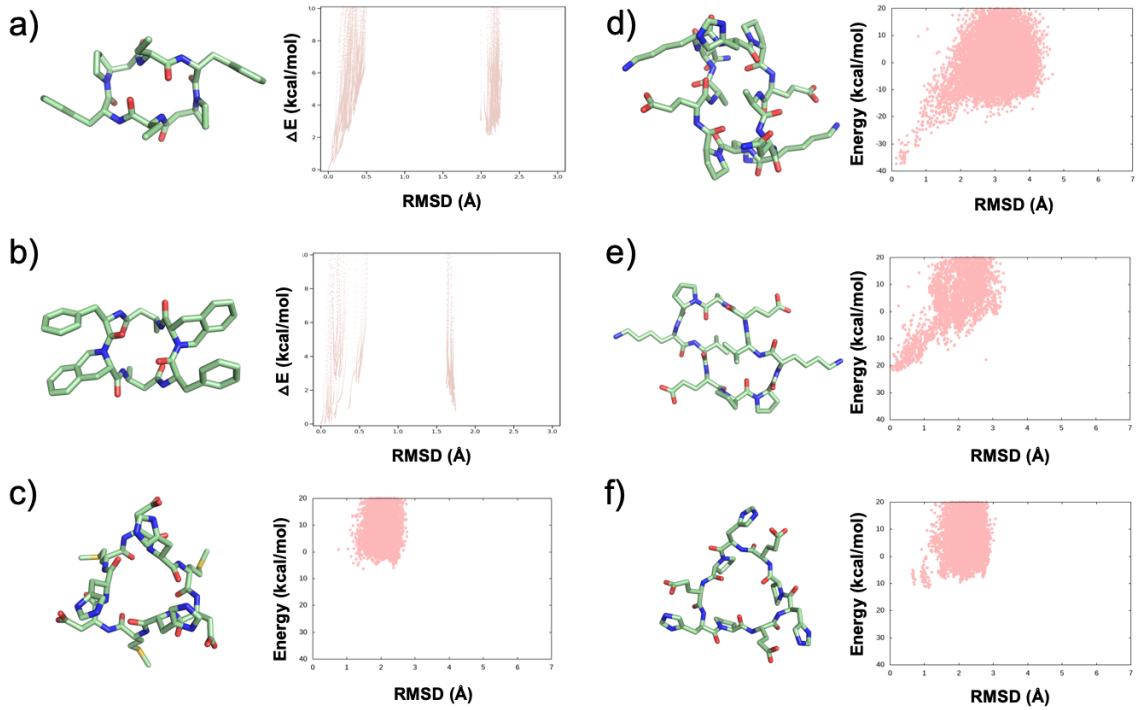


Figure S1. Predicted energy landscape for each designed peptide. The first column shows the designed conformation, while the second column shows the energy landscape calculated using either AIMNet (a-b) or Rosetta (c-f). Backbone RMSD is calculated to the designed conformation on the left. (a) AIMNet energy landscape for C2-1. (b) AIMNet energy landscape for C2-2. (c) Rosetta energy landscape for C3-2. (d) Rosetta energy landscape for S2-1. (e) Rosetta energy landscape for S2-2. (f) Rosetta energy landscape for C3-1.

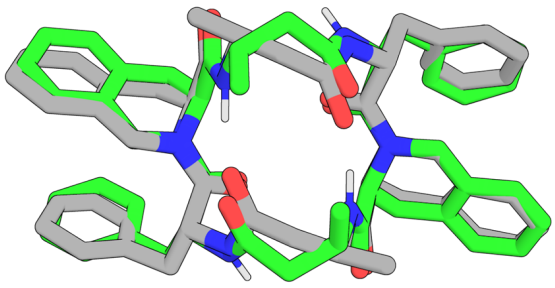


Figure S2. Alignment of C2-2 apo design (green) to crystal structure (gray) with 0.5 Å Cα RMSD.

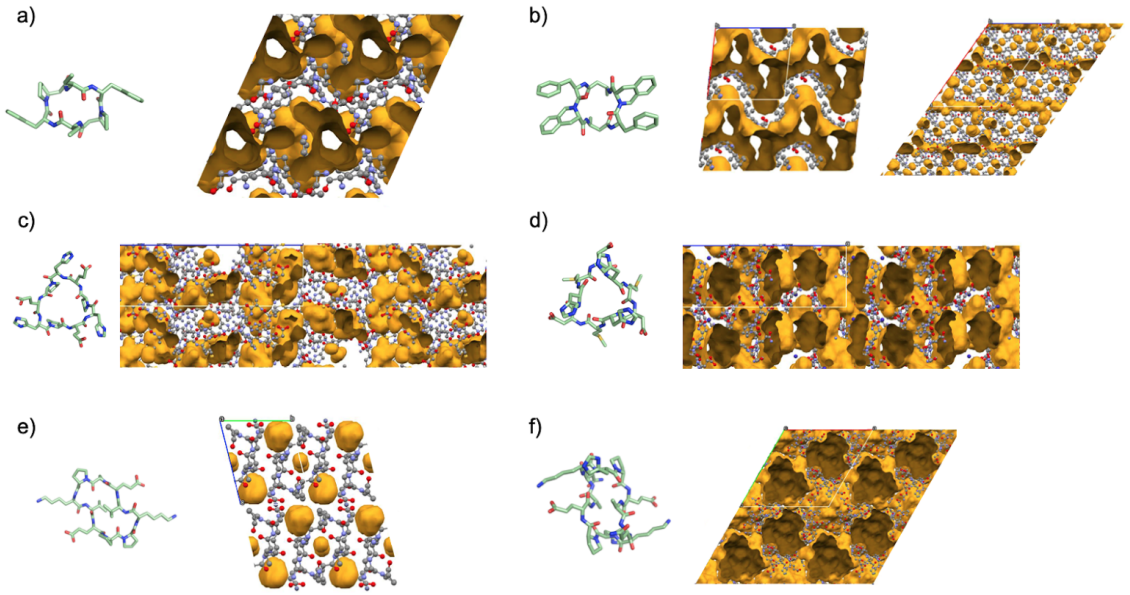


Figure S3. Mercury calculated void volume shown in yellow surface for each peptide crystal. (a) C2-1 crystal shown along the c axis. (b) C2-2 crystal 1 (left) and crystal 2 (right) shown along the b axis. (c) C3-1 crystal shown along the b axis. (d) C3-2 crystal shown along the b axis. (e) S2-1 crystal shown along the a axis. (f) S2-2 crystal shown along the c axis.

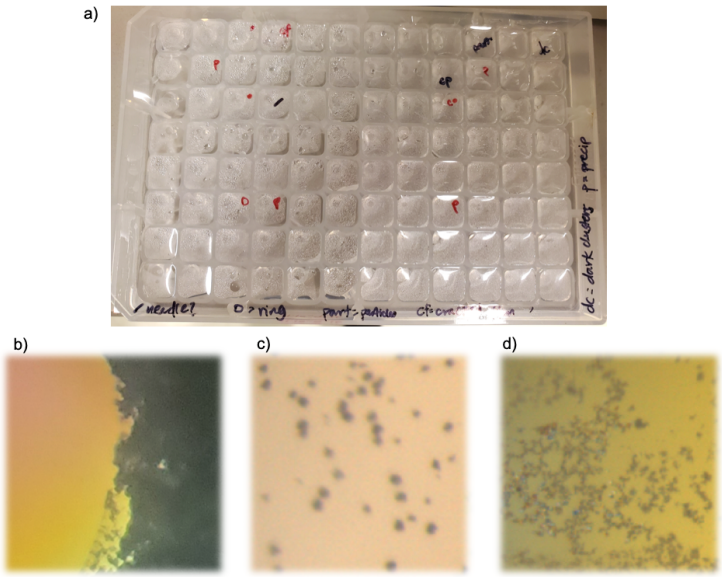


Figure S4. (a) Example of a 96 well plate used to screen one peptide. Wells that form precipitate are labeled with a 'P', dark particles are 'dc', crystalline materials are 'c'. (b) Example of a precipitated peptide. (c) Example of particles. (d) Example of crystalline peptide.

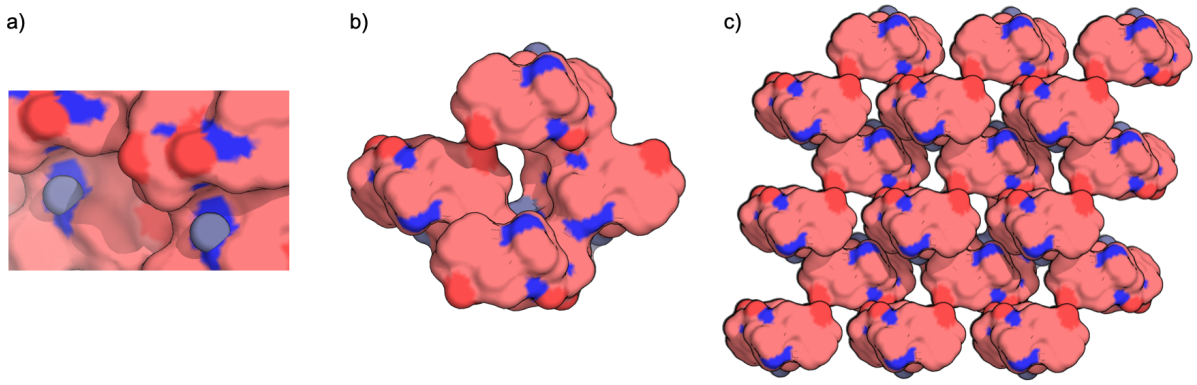


Figure S5. View along the b axis of C2-2 crystal. (a) Zoomed in view of the solvent accessible cavity with partially coordinated zinc ions (purple). (b) View of one pore going through the crystal lattice. (c) Multiple space units of C2-2 in the $P12_11$ space group.

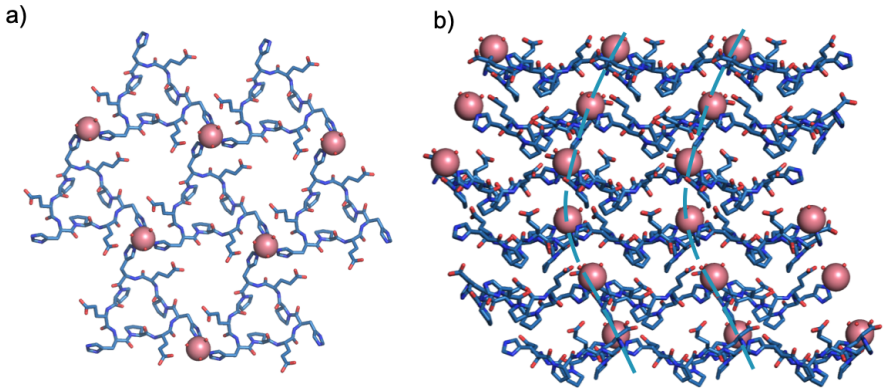


Figure S6. (a) View of a single layer of Co^{2+}-C3-1 2D sheet with C3 symmetry. (b) View along the *b* axis showing six layers of 2D planes stacked in a twisted way (teal dashed curves).

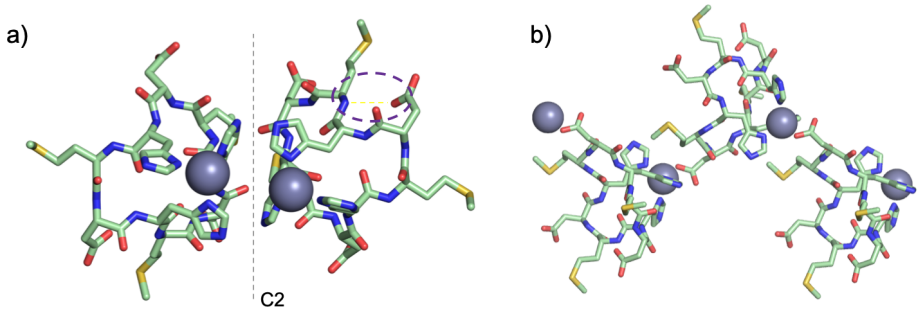


Figure S7. (a) Adjacent Zn-C3-2 1D chains interact via dispersion interactions. (b) 1D Metal coordinating peptide chain of C3-2.

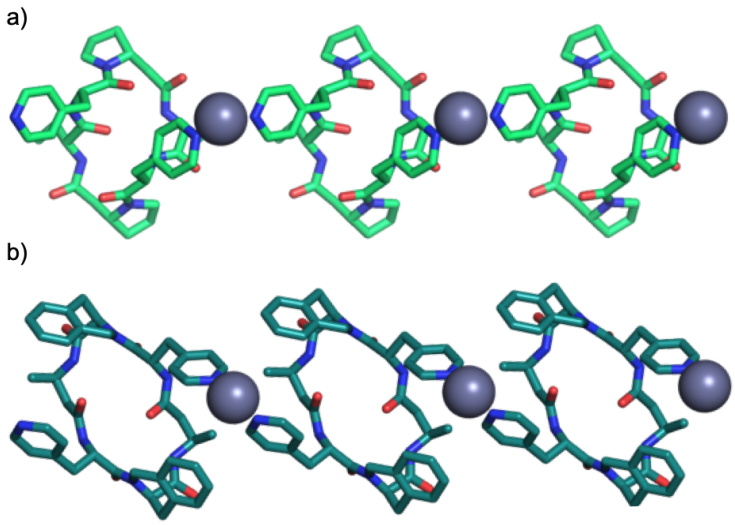


Figure S8. (a) view of the Zn^{2+}-C2-1 1D chain (two coordinated water molecules are omitted). (b) view of the Zn^{2+}-C2-2 1D chain (two coordinated water molecules are omitted).

2.6.2 Supplementary Tables

Table S1: Organic conditions to screen for crystal formation using a 96 well plate.

	5mM $Zn(NO_3)_2$	5mM $Cu(NO_3)_2$	5mM $Zn(OAc)_2$	5mM $Cu(OAc)_2$	5mM $ZnCl_2$	5mM $CuCl_2$	5mM $Zn(NO_3)_2$	5mM $Cu(NO_3)_2$	5mM $Zn(OAc)_2$	5mM $Cu(OAc)_2$	5mM $ZnCl_2$	5mM $CuCl_2$
	1	2	3	4	5	6	7	8	9	10	11	12
A	DMF	DMF	DMF	DMF	DMF	DMF	MeOH	MeOH	MeOH	MeOH	MeOH	MeOH
B	1:1 water:DMF	1:1 water:DMF	1:1 water:DMF	1:1 water:DMF	1:1 water:DMF	1:1 water:DMF	1:1 water:MeOH	1:1 water:MeOH	1:1 water:MeOH	1:1 water:MeOH	1:1 water:MeOH	1:1 water:MeOH
C	2:1:1 DMF:water: ACN	2:1:1 DMF:water: ACN	2:1:1 DMF:water: ACN	2:1:1 DMF:water: ACN	2:1:1 DMF:water: ACN	2:1:1 DMF:water: ACN	2:1:1 MeOH:water :ACN	2:1:1 MeOH:water: ACN	2:1:1 MeOH:water: ACN	2:1:1 MeOH:water :ACN	2:1:1 MeOH:water :ACN	2:1:1 MeOH:water: ACN
D	1:2:1 DMF:water: ACN	1:2:1 DMF:water: ACN	1:2:1 DMF:water: ACN	1:2:1 DMF:water: ACN	1:2:1 DMF:water: ACN	1:2:1 DMF:water: ACN	1:2:1 MeOH:water :ACN	1:2:1 MeOH:water: ACN	1:2:1 MeOH:water: ACN	1:2:1 MeOH:water :ACN	1:2:1 MeOH:water :ACN	1:2:1 MeOH:water: ACN
E	DEF	DEF	DEF	DEF	DEF	DEF	EtOH	EtOH	EtOH	EtOH	EtOH	EtOH
F	2:1 DEF:water	2:1 DEF:water	2:1 DEF:water	2:1 DEF:water	2:1 DEF:water	2:1 DEF:water	1:1 water:EtOH	1:1 water:EtOH	1:1 water:EtOH	1:1 water:EtOH	1:1 water:EtOH	1:1 water:EtOH
G	1:1 water:DEF	1:1 water:DEF	1:1 water:DEF	1:1 water:DEF	1:1 water:DEF	1:1 water:DEF	IPA	IPA	IPA	IPA	IPA	IPA
H	1:2 DEF:water	1:2 DEF:water	1:2 DEF:water	1:2 DEF:water	1:2 DEF:water	1:2 DEF:water	1:1 water:IPA	1:1 water:IPA	1:1 water:IPA	1:1 water:IPA	1:1 water:IPA	1:1 water:IPA

Table S2: Aqueous conditions to screen for crystal formation using a 96 well plate.

	5mM $Zn(NO_3)_2$						5mM $Fe(NO_3)_3$					
	25mM MES pH 5.5	25 mM MES pH 6	25 mM MES pH 6.5	25 mM HEPES pH 7	25 mM HEPES pH 7.5	25 mM HEPES pH 8	25mM MES pH 5.5	25 mM MES pH 6	25 mM MES pH 6.5	25 mM HEPES pH 7	25 mM HEPES pH 7.5	25 mM HEPES pH 8
	1	2	3	4	5	6	7	8	9	10	11	12
A	-	-	-	-	-	-	-	-	-	-	-	-
B	PEG300 2%	PEG300 2%	PEG300 2%	PEG300 2%	PEG300 2%	PEG300 2%	PEG300 2%	PEG300 2%	PEG300 2%	PEG300 2%	PEG300 2%	PEG300 2%
C	PEG1000 2%	PEG1000 2%	PEG1000 2%	PEG1000 2%	PEG1000 2%	PEG1000 2%	PEG1000 2%	PEG1000 2%	PEG1000 2%	PEG1000 2%	PEG1000 2%	PEG1000 2%
D	PEG2000 2%	PEG2000 2%	PEG2000 2%	PEG2000 2%	PEG2000 2%	PEG2000 2%	PEG2000 2%	PEG2000 2%	PEG2000 2%	PEG2000 2%	PEG2000 2%	PEG2000 2%
E	PEG4000 2%	PEG4000 2%	PEG4000 2%	PEG4000 2%	PEG4000 2%	PEG4000 2%	PEG4000 2%	PEG4000 2%	PEG4000 2%	PEG4000 2%	PEG4000 2%	PEG4000 2%
F	PEP 2%	PEP 2%	PEP 2%	PEP 2%	PEP 2%	PEP 2%	PEP 2%	PEP 2%	PEP 2%	PEP 2%	PEP 2%	PEP 2%
G	EtOH 25%	EtOH 25%	EtOH 25%	EtOH 25%	EtOH 25%	EtOH 25%	EtOH 25%	EtOH 25%	EtOH 25%	EtOH 25%	EtOH 25%	EtOH 25%
H	MeOH 25%	MeOH 25%	MeOH 25%	MeOH 25%	MeOH 25%	MeOH 25%	MeOH 25%	MeOH 25%	MeOH 25%	MeOH 25%	MeOH 25%	MeOH 25%

Table S3: Observational notes for screened peptides.

peptide	sequence	precipitate	particles	crystalline
6.3_123_cell029	CYS DHIS PRO CYS DHIS PRO CYS DHIS PRO	DMF ($Zn(NO_3)_2$), MeOH ($Zn(NO_3)_2$), 1:4 water:DMF (($Zn(NO_3)_2$)),		
11.4_1240_cell028	GLU DHIS SER GLU DHIS SER GLU DHIS SER	DMF ($Zn(NO_3)_2$), MeOH ($Zn(NO_3)_2$), 1:4 water:DMF ($Zn(NO_3)_2$), H2O ($Zn(OAc)_2$, 100mM HEPES ($Cu(OAc)_2$)		
6.10_1223_cell028	CYS DHIS MET CYS DHIS MET CYS DHIS MET	1:1 water:MeOH ($Zn(NO_3)_2$, DMF ($Zn(OAc)_2$, $ZnCl_2$), 1:1 water:DMF ($Cu(OAc)_2$), DEF ($Zn(NO_3)_2$, $ZnCl_2$), 2:1 DEF:water ($Zn(NO_3)_2$), 1:1 water:DEF ($Cu(NO_3)_2$, $Cu(OAc)_2$), 1:2 DEF:water ($Cu(NO_3)_2$, $Cu(OAc)_2$)		
7.1_192_cell026	DCYS MET HIS DCYS MET HIS DCYS MET HIS	DMF ($Zn(NO_3)_2$, $Cu(NO_3)_2$), 1:4 water:DMF ($Zn(NO_3)_2$, $Cu(NO_3)_2$), 1:1 water:MeOH ($Zn(NO_3)_2$, $Cu(NO_3)_2$) DMF ($ZnCl_2$, $CuCl_2$), MeOH ($Zn(NO_3)_2$, $Cu(NO_3)_2$), 1:1 water:DMF ($ZnCl_2$, $CuCl_2$), DEF ($Zn(NO_3)_2$, $CuCl_2$), 2:1 DEF:water ($CuCl_2$), 1:1 water:DEF ($Cu(NO_3)_2$, $CuCl_2$), 1:2 DEF:water ($CuCl_2$)		
7.10_194_cell034	CYS PRO HIS CYS PRO HIS CYS PRO HIS	DMF ($Zn(NO_3)_2$, $Cu(NO_3)_2$), 1:4 water:DMF ($Zn(NO_3)_2$, $Cu(NO_3)_2$), MeOH ($Zn(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:1 water:MeOH ($ZnCl_2$), 2:1:1 DMF:water:ACN ($ZnCl_2$), 2:1:1 MeOH:water:ACN ($Cu(NO_3)_2$, $CuCl_2$), 1:2:1 DMF:water:ACN ($ZnCl_2$), 1:2:1 MeOH:water:ACN ($ZnCl_2$), EtOH ($Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:1 water:etOH ($ZnCl_2$, $CuCl_2$), 1:1 water:DEF ($Cu(OAc)_2$), IPA ($Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$), 1:2 DEF:water ($Cu(OAc)_2$), 1:1 water:IPA ($Cu(NO_3)_2$, $Zn(OAc)_2$)		
1.3_176_cell037	ASP DHIS DMET ASP DHIS DMET ASP DHIS DMET	DMF ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $ZnCl_2$), MeOH ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$), 1:1 water:DMF ($ZnCl_2$), 1:1 water:MeOH ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$), 2:1:1 DMF:water:ACN ($Cu(NO_3)_2$, $Zn(OAc)_2$, $ZnCl_2$, $CuCl_2$), 2:1:1 MeOH:water:ACN ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$), 1:2:1 DMF:water:ACN ($Cu(NO_3)_2$, $ZnCl_2$, $CuCl_2$), 1:2:1 MeOH:water:ACN ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$), DEF	1:1 water:DEF ($Cu(NO3)_2$, $Cu(OAc)_2$, $ZnCl_2$), IPA ($CuCl_2$), 1:2 DEF:water ($Cu(NO3)_2$, $Cu(OAc)_2$, $ZnCl_2$)	DEF ($Zn(NO_3)_2$), 10mM MES pH 6 ($Zn(NO_3)_2$), 20mM HEPES pH 7 ($Zn(NO_3)_2$)

		($Zn(OAc)_2$, $CuCl_2$), EtOH ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $CuCl_2$), 2:1 DEF:water ($Zn(OAc)_2$, $ZnCl_2$), 1:1 water:EtOH ($Zn(OAC)_2$, $Cu(NO_3)_2$, $Zn(NO_3)_2$), IPA ($Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:2 DEF:water ($CuCl_2$), 1:1 water:IPA ($Zn(NO_3)_2$, $Cu(NO_3)_2$)		
1.3_70_cell037	MET DHIS DASP MET DHIS DASP MET DHIS DASP	DMF($Zn(NO_3)_2$), 1:4 water:DMF ($Zn(NO_3)_2$), 1:1 water:MeOH ($Zn(NO_3)_2$), DMF ($ZnCl_2$), MeOH ($Zn(OAc)_2$), 1:1 water:DMF ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$), 2:1:1 DMF:water:ACN ($Zn(NO_3)_2$), DEF ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$), 2:1 DEF:water ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$), 1:1 water:DEF ($CuCl_2$), IPA ($Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$)		
12.6_637_cell047	GLU DHIS PRO GLU DHIS PRO GLU DHIS PRO	1:1 water:DMF ($Zn(OAc)_2$, $Cu(OAc)_2$, $Cu(NO_3)_2$), DMF ($Zn(OAc)_2$,		DMF ($Zn(NO_3)_2$), 20mM HEPES pH 7 ($Cu(NO_3)_2$, $Co(NO_3)_2$)), 20mM HEPES pH 8.2 ($Co(NO_3)_2$)
34.8_0_cell028	dHIS ALA ASP SER dHIS ALA ASP SER dHIS ALA ASP SER	DMF ($Zn(NO_3)_2$, $Cu(OAc)_2$), 1:1 water:DMF ($Zn(NO_3)_2$, $Cu(OAc)_2$), DEF ($CU(OAc)_2$, $CuCl_2$), EtOH ($Cu(NO_3)_2$, $Cu(OAc)_2$), 2:1 DEF:water ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, IPA ($Cu(OAc)_2$)	1:1 water:DEF ($Zn(OAc)_2$)	1:1 water:DMF ($Zn(OAc)_2$)
25.8_19_cell031	MET DHIS PRO DASP MET DHIS PRO DASP MET DHIS PRO DASP	DMF($Zn(OAc)_2$, $Cu(OAc)_2$), 1:1 water:DMF ($Zn(OAc)_2$, $Cu(OAc)_2$), 1:1 water:MeOH ($Zn(OAc)_2$, $Cu(OAc)_2$, 20mM HEPES pH 7 ($ZN(OAc)_2$, $Zn(NO_3)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$), 20mM HEPES pH 8 ($Zn(OAc)_2$, $Zn(NO_3)_2$, $Cu(OAc)_2$)		
7.7_120_cell034	DASP HIS MET DASP HIS MET DASP HIS MET	DMF ($Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$), 1:1 water:DMF ($Zn(OAc)_2$), 1:1 water:MeOH ($Zn(NO_3)_2$), 2:1:1 DMF:water:ACN ($Zn(OAc)_2$), 2:1:1 MeOH:water:ACN ($Zn(NO_3)_2$), 1:2:1 DMF:water:ACN ($Zn(OAc)_2$), 1:2:1 MeOH:water:ACN ($Zn(NO_3)_2$), DEF ($Zn(NO_3)_2$, $ZnCl_2$), 2:1 DEF:water ($Zn(NO_3)_2$, $Zn(OAc)_2$), 1:1 water:EtOH ($Zn(NO_3)_2$, $Cu(NO_3)_2$), 1:1 water:DEF ($Zn(NO_3)_2$), IPA ($Cu(NO_3)_2$, $Cu(OAc)_2$, $CuCl_2$), 1:2 DEF:water ($Zn(OAc)_2$), 1:1 water:IPA ($Zn(NO_3)_2$, $Zn(OAc)_2$, $CuCl_2$)	2:1 DEF:water ($ZnCl_2$), DEF($Cu(OAc)_2$), 1:1 water:DEF ($Cu(OAc)_2$, $ZnCl_2$), 1:2 DEF:water ($Cu(OAc)_2$, $ZnCl_2$)	DMF ($Zn(NO_3)_2$, DEF ($Cu(NO_3)_2$)
3.4_196_cell030	DASP DGLN HIS DASP DGLN HIS DASP DGLN HIS	DMF ($Zn(NO_3)_2$, $CU(NO_3)_2$), MeOH ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:1 water:DMF ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $ZnCl_2$), 1:1 water:MeOH ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 2:1:1 DMF:water:ACN ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 2:1:1 MeOH:water:ACN ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:2:1 DMF:water:ACN ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$), 1:2:1 MeOH:water:ACN ($Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), DEF ($Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$), DMF ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $CuCl_2$), EtOH ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 2:1 DEF:water ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$), 1:1 water:etOH ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$), 1:1 water:DEF ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$), IPA ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:2 DEF:water ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$), 20mM HEPES pH7 ($Zn(OAc)_2$, $Zn(NO_3)_2$, $Cu(OAc)_2$, $Cu(NO_3)_2$), 20mM HEPES pH 8 ($Zn(OAc)_2$, $Zn(NO_3)_2$)	DMF ($Cu(NO_3)_2$), 1:1 MeOH:water ($Zn(NO_3)_2$)	
3.5_187_cell031	DASP DTRP HIS DASP DTRP HIS DASP DTRP HIS	DMF ($Zn(NO_3)_2$, $Zn(NO_3)_2$), MeOH ($Cu(NO_3)_2$), 1:1 water:MeOH ($Cu(NO_3)_2$), 2:1:1 DMF:water:ACN ($CuCl_2$), 2:1:1 MeOH:water:ACN ($Cu(NO_3)_2$, $ZnCl_2$), 1:2:1 DMF:water:ACN ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $CuCl_2$), 1:2:1 MeOH:water:ACN ($Zn(NO_3)_2$,	MeOH ($Zn(NO_3)_2$, 1:1 water:MeOH ($CuCl_2$)	

		$Cu(NO_3)_2$, $Zn(OAc)_2$), DEF ($Zn(NO_3)_2$, $Zn(OAc)_2$), EtOH ($Cu(NO_3)_2$, $Cu(OAc)_2$, $CuCl_2$), 2:1 DEF:water ($Zn(NO_3)_2$), 1:1 water:EtOH ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:1 water:DEF ($Cu(NO_3)_2$, $Zn(OAc)_2$), IPA ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:2 DEF:water ($Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:1 water:IPA ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$), 20mM HEPES pH 7 ($Zn(OAc)_2$, $Zn(NO_3)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $Co(NO_3)_2$), 20mM HEPES pH 8 ($Zn(OAc)_2$, $Zn(NO_3)_2$, $Cu(OAc)_2$, $Cu(NO_3)_2$, $FeCl_3$, $Co(NO_3)_2$)		
28.6_2_cell032_peak 2	THR MET DCYS DHIS THR MET DCYS DHIS THR MET DCYS DHIS	MeOH ($Zn(OAc)_2$, 1:1 water:DMF ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $ZnCl_2$), 2:1:1 DMF:water:ACN ($Cu(OAc)_2$, $ZnCl_2$), 2:1:1 DMF:water:ACN ($Zn(OAc)_2$), 1:2:1 DMF:water:ACN ($Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$), 1:2:1 MeOH:water:ACN ($Zn(NO_3)_2$, $CuCl_2$), DEF ($Zn(NO_3)_2$), EtOH ($Zn(NO_3)_2$, $Zn(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:1 water:EtOH ($Zn(NO_3)_2$, $ZnCl_2$), 1:1 water:DEF ($Zn(OAc)_2$), IPA ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:2 DEF:water ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$), 1:1 water:IPA ($Zn(NO_3)_2$, $Zn(OAc)_2$, $CuCl_2$), 20mM HEPES pH 7 ($Zn(OAc)_2$, $Cu(OAc)_2$, $Cu(NO_3)_2$, $FeCl_3$, $Zn(NO_3)_2$,$CaCl_2$, $Co(NO_3)_2$), 20mM HEPES pH 8 ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $Cu(NO_3)_2$, $FeCl_3$, $Co(NO_3)_2$, $CaCl_2$)	DMF ($CuCl_2$)	
34.3_0_cell032	MET DHIS DCYS PRO MET DHIS DCYS PRO MET DHIS DCYS PRO	MeOH ($CuNO_3)_2$), 1:! water:DMF ($Cu(NO_3)_2$, $CuCl_2$), 1:1 water:MeOH ($CuCl_2$), DEF ($Cu(OAc)_2$), 2:1 DEF:water ($Cu(OAc)_2$), 1:1 water:DEF ($Cu(NO_3)_2$, $Zn(OAc)_2$), IPA ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:2 DEF:water ($Zn(OAc)_2$, $Cu(OAc)_2$), 10mM MES pH 6 ($Zn(NO_3)_2$, $Cu(NO_3)_2$), 20mM HEPES pH 7 ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $FeCl_3$), 20mM HEPES pH 8 ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $FeCl_3$, $Co(NO_3)_2$)		
8.2_26_cell029	DCYS ALA PRO DASP DCYS ALA PRO DASP DCYS ALA PRO DASP	MeOH($Cu(OAc)_2$, $ZnCl_2$), 1:1 water:MeOH ($Cu(OAc)_2$), 2:1:1 MeOH:water:ACN ($ZnCl_2$), 1:2:1 DMF: water:ACN ($CuCl_2$), DEF ($Cu(NO_3)_2$), EtOH ($Cu(NO_3)_2$), 2:1 DEF:water ($Zn(NO_3)_2$), 1:1 water:EtOH ($Cu(OAc)_2$), IPA ($CuCl_2$), 1:1 water:IPA ($CuCl_2$)	2:1:1 DMF:water:ACN ($Cu(NO_3)_2$), 1:2:1 DMF:water:ACN ($Cu(NO_3)_2$), 1:1 water:DEF ($Cu(OAc)_2$), 1:2 DEF:water ($Cu(NO_3)_2$, $Cu(OAc)_2$), 1:1 water:IPA ($Cu(NO_3)_2$)	1:1 water:DMF ($Cu(OAc)_2$), 1:2:1 DMF:water:ACN ($Cu(OAc)_2$), 1:1 water:DEF ($Cu(NO_3)_2$)
1.3_470_cell032	HIS DASP DMET HIS DASP DMET HIS DASP DMET	DMF($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$), DEF ($Zn(NO_3)_2$), MeOH ($Zn(OAc)_2$, $Cu(OAc)_2$, $CuCl_2$), 1:1 water:DMF ($Zn(OAc)_2$), 1:1 water:MeOH ($Zn(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$), 2:1:1 DMF:water:ACN ($Zn(OAc)_2$), 2:1:1 MeOH:water:ACN ($Zn(OAc)_2$, $Cu(OAc)_2$), 1:2:1 MeOH:water:ACN ($Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$), EtOH ($Zn(OAc)_2$, $Cu(OAc)_2$, $CuCl_2$), 2:1 DEF:water ($Zn(OAc)_2$, $Cu(OAc)_2$), 1:1 water:EtOH ($Zn(OAc)_2$, $Cu(OAc)_2$, $CuCl_2$), 1:1 water:DEF ($Cu(OAc)_2$, $CuCl_2$), IPA($Zn(OAc)_2$, $Cu(OAc)_2$), 1:2 DEF:water ($Cu(NO_3)_2$, $Zn(OAc)_2$, $CuCl_2$), 1:1 water:IPA ($Zn(OAc)_2$, $Cu(OAc)_2$)		
1.5_72_cell034	MET DASP CYS MET DASP CYS MET DASP CYS	DMF ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), MeOH ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:1 water:DMF ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:1 water:MeOH ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 2:1:1 DMF:water:ACN ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$,		

		$Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 2:1:1 MeOH:water:ACN ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:2:1 DMF:water:ACN ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:2:1 MeOH:water:ACN ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), DEF ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), EtOH ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 2:1 DEF:water ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:1 water:EtOH ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:1 water:DEF ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, Cu(OAc)2, $ZnCl_2$, $CuCl_2$), IPA ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:2 DEF:water ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:1 water:IPA ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$)		
10.1_32_cell030	HIS DVAL DASP HIS DVAL DASP HIS DVAL DASP	DMF ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$), MeOH ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Cu(OAc)_2$, $ZnCl_2$, $CuCl_2$), 1:1 water:DMF ($Zn(NO_3)_2$, $Cu(NO_3)_2$), 1:1 water:MeOH ($Zn(NO_3)_2$, $ZnCl_2$), 2:1:1 DMF:water:ACN ($Zn(NO_3)_2$, $Cu(NO_3)_2$), 2:1:1 MeOH:water:ACN ($Zn(NO_3)_2$, $CuCl_2$), 1:2:1 DMF:water:ACN ($CuCl_2$), 1:2:1 MeOH:water:ACN ($Zn(NO_3)_2$), DEF ($Cu(NO_3)_2$, $Zn(OAc)_2$), EtOH($Zn(NO_3)_2$, $Cu(NO_3)_2$), 2:1 DEF:water ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$), 1:1 water:DEF ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$), 1:2 DEF:water ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $Zn(OAc)_2$), 1:1 water:IPA ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $ZnCl_2$)		1:2:1 DMF:water:ACN ($Zn(NO_3)_2$, $Cu(OAc)_2$), DEF ($Zn(NO_3)_2$), 20mM HEPES pH 8 ($Co(NO_3)_2$)
10.5_9_I213_199	DASP PRO DASP DASP PRO DASP DASP PRO DASP	1:2 water:DEF ($Zn(OAc)_2$)	MeOH ($Cu(OAc)_2$, $CuCl_2$), 1:1 Water:DMF ($Cu(OAc)_2$, $ZnCl_2$),	
11.10_9_I213_16	ASP DPRO ASP ASP DPRO ASP ASP DPRO ASP	1:1 water:DMF ($Cu(NO_3)_2$), 1:1 water:MeOH ($CuOAc)_2$, 2:1 DEF:water ($Cu(OAc)_2$, 1:1 water:EtOH ($Zn(OAc)_2$), 29mM HEPES pH 7 ($FeCl_3$)	MeOH($Cu(OAc)_2$, $CuCl_2$)	2:1:1 DMF:water:ACN ($Cu(OAc)_2$)
2.2_9_I213_166	DASP DASP TYR DASP DASP TYR DASP DASP TYR	DMF ($Zn(OAc)_2$, $Cu(OAc)_2$), 20mM HEPES pH 8 ($FeCl_3$)	MeOH ($Cu(OAc)_2$, $Cu(NO_3)_2$), EtOH ($Cu(NO_3)_2$, $Cu(OAc)_2$)	DMF($CuCl_2$), MeOH ($ZnCl_2$), 2:1:1 DMF:water:ACN ($Zn(OAc)_2$, 1:2:1 DMF:water:ACN ($Zn(NO_3)_2$),
3.9_9_I213_260	VAL DASP DASP VAL DASP DASP VAL DASP DASP		MeOH($Cu(NO_3)_2$, $Cu(OAc)_2$), 1:1 water:MeOH ($Cu(NO_3)_2$),	
1.4_9_I213_51	ASP ASP DTYR ASP ASP DTYR ASP ASP DTYR	DMF ($ZnCl_2$)	DMF ($Cu(NO_3)_2$, $ZnCl_2$), MeOH ($Cu(NO_3)_2$, $Cu(OAc)_2$), 1:1 water:DMF ($CuNO_3)_2$), 1:1 water:MeOH ($Cu(NO_3)_2$, $Cu(OAc)_2$), 2:1:1 MeOH:water:ACN ($Zn(OAc)_2$), 1:2:1 DMF:water:ACN ($Zn(OAc)_2$), 1:2:1 MeOH:water:ACN ($Cu(OAc)_2$), DEF($Zn(OAc)_2$, EtOH ($Cu(OAc)_2$)	2:1:1 MeOH:water:ACN ($Zn(NO_3)_2$), 1:2:1 MeOH:water:ACN ($Cu(NO_3)_2$), EtOH ($Cu(NO_3)_2$)
3.9_9_I213_134	SER DASP DASP SER DASP DASP SER DASP DASP		MeOH ($Cu(NO_3)_2$, $Cu(OAc)_2$), 1:1 Water:MeOH ($Cu(NO_3)_2$),	
17.2_12_P4132_1	GLU DPRO GLU	1:1 water:DMF ($Zn(NO_3)_2$, $Cu(NO_3)_2$, $ZnCl_2$, $CuCl_2$), 1:2:1	2:1 DEF:water ($CuCl_2$), 1:1	DMF ($Cu(OAc)_2$), MeOH

34	DASN GLU DPRO GLU DASN GLU DPRO GLU DASN	DMF:water:ACN ($Cu(NO_3)_2$), MeOH ($Cu(NO_3)_2$, $Cu(OAc)_2$), 1:1 water:MeOH ($Cu(OAc)_2$)	water:DEF ($Cu(OAc)_2$), DMF ($CaCl_2$), 1:1 water:MeOH ($Cu(NO_3)_2$)	($Cu(OAc)_2$), MeOH ($Cu(NO_3)_2$), 1:1 water:MeOH ($Cu(NO_3)_2$)
64.10_12_P4132_2659	SER GLU PRO GLU SER GLU PRO GLU SER GLU PRO GLU		DMF ($Cu(NO_3)_2$), 1:1 water:DMF ($Cu(NO_3)_2$), 2:1:1 DMF:water:ACN ($Cu(NO_3)_2$), 1:2:1 DMF:water:ACN ($Cu(NO_3)_2$), DEF ($Cu(NO_3)_2$), 1:1 water:DEF ($Cu(NO_3)_2$), 1:2 DEF:water ($Cu(NO_3)_2$, $Cu(OAc)_2$)	
28.1_12_P4132_164	ASP ASP DMET DPRO ASP ASP DMET DPRO ASP ASP DMET DPRO		MeOH ($Cu(OAc)_2$), 1:2 DEF:H2O ($Cu(OAc)_2$)	
90.1_12_I213_1099	DASP PRO DASP SER DASP PRO DASP SER DASP PRO DASP SER		DMF ($Zn(NO_3)_2$, $Cu(NO_3)_2$), 1:1 water:DMF ($Cu(NO_3)_2$), 1:2:1 DMF:water:ACN ($Cu(NO_3)_2$, $Cu(OAc)_2$), 1:2:1 MeOH:water:ACN ($Cu(NO_3)_2$, $Cu(OAc)_2$), DEF ($Cu(OAc)_2$), EtOH ($Cu(OAc)_2$), 2:1 DEF:water ($Cu(OAc)_2$), 1:1 water:DEF ($Cu(OAc)_2$), IPA ($Cu(NO_3)_2$), 1:2 DEF:water ($Cu(NO_3)_2$),	
34.4_12_I213_499	ASP DTYR ASP DPRO ASP DTYR ASP DPRO ASP DTYR ASP DPRO		MeOH ($Cu(NO_3)_2$), MeOH ($Cu(OAc)_2$)	
11.3_20_3_1in	dASP MET GLN dASP MET GLN dASP MET GLN	20mM HEPES pH 7 and 8 ($Cu(OAc)_2$), 20mM HEPES pH 7 and 8 ($Cu(NO_3)_2$), 20mM HEPES pH 7 and 8 ($FeCl_3$), 20mM HEPES pH 8 ($Zn(NO_3)_2$)		
7.7_22_13	SER GLU dSER SER GLU dSER SER GLU dSER	20mM HEPES pH 8 ($Cu(OAc)_2$), 20mM HEPES pH 7 and 8 ($Cu(NO_3)_2$), 20mM HEPES pH 7 and 8 ($FeCl_3$), 20mM HEPES pH 8 (($Zn(NO_3)_2$))		
11.9_20_0	ASP dMET THR ASP dMET THR ASP dMET THR	20mM HEPES pH 8 ($Cu(OAc)_2$), 20mM HEPES pH 8 ($Cu(NO_3)_2$), 20mM HEPES pH 7 and 8 ($FeCl_3$)		
33.5_12_P4132_3404	GLU DMET LYS GLU GLU DMET LYS GLU GLU DMET LYS GLU			
10.10_12_I213_193	THR MET DASP DASP THR MET DASP DASP THR MET DASP DASP			
12.4_12_I213_261	ASP DMET DSER ASP ASP DMET DSER ASP ASP DMET DSER ASP			
38.10_12_I213_155	DASP THR DASP DVAL DASP THR DASP DVAL DASP			

	THR DASP DVAL			
57.1_12_I213_234	GLU GLU DTHR DMET GLU GLU DTHR DMET GLU GLU DTHR DMET			
80.1_12_I213_270	DASP MET DSER DASP DASP MET DSER DASP DASP MET DSER DASP			
3.9_18_0	dASP dALA SER dASP dALA SER dASP dALA SER			
11.3_20_3	dASP MET THR dASP MET THR dASP MET THR			
2.1_20_39	MET dSER GLU MET dSER GLU MET dSER GLU			
10.7_21_58	MET dSER dGLU MET dSER dGLU MET dSER dGLU			
10.7_20_2	ASP dVAL dSER ASP dVAL dSER ASP dVAL dSER			
1.2_18_18	GLU MET dALA GLU MET dALA GLU MET dALA			

2.6.3 Computational Methods

All scripts for metal crystal lattice design are available in the following repository: https://github.com/willsheffler/mof/tree/master/mof and shown in the following listings.

Listing S1: The following script samples rotamers of metal chelating residues such as aspartate, histidine, cystine, and glutamate around different metal geometries.

```
import numpy as np, rpxdock as rp, copy, os, mof
from mof import util
from mof.pyrosetta_init import make_1res_pose, get_sfxn, rVec, xform_pose, Pose
from abc import ABC, abstractmethod
"""
CONCERNS:
how to handle multiple metal binding sides not covered by rotamers, as in GLU
how to handle CYS chi2, which is based on HG being free-ish to rotate
"""

def get_rotclouds(**kw):
   kw = rp.Bunch(kw)
```

```
   chiresl_asp1 = kw.chiresl_asp1 / kw.scale_number_of_rotamers
   chiresl_asp2 = kw.chiresl_asp2 / kw.scale_number_of_rotamers
   chiresl_cys1 = kw.chiresl_cys1 / kw.scale_number_of_rotamers
   chiresl_cys2 = kw.chiresl_cys2 / kw.scale_number_of_rotamers
   chiresl_his1 = kw.chiresl_his1 / kw.scale_number_of_rotamers
   chiresl_his2 = kw.chiresl_his2 / kw.scale_number_of_rotamers
   chiresl_glu1 = kw.chiresl_glu1 / kw.scale_number_of_rotamers
   chiresl_glu2 = kw.chiresl_glu2 / kw.scale_number_of_rotamers
   chiresl_glu3 = kw.chiresl_glu3 / kw.scale_number_of_rotamers

   os.makedirs(kw.rotcloud_cache, exist_ok=True)

   params = (kw.chiresl_his1, kw.chiresl_his2, kw.chiresl_cys1, kw.chiresl_cys2,
kw.chiresl_asp1,
             kw.chiresl_asp2, kw.chiresl_glu1, kw.chiresl_glu2, kw.chiresl_glu3,
kw.maxdun_cys,
             kw.maxdun_asp, kw.maxdun_glu, kw.maxdun_his,
kw.scale_number_of_rotamers)
   ident = mof.util.hash_str_to_int(str(params))

   cache_file = kw.rotcloud_cache + '/%i.pickle' % ident
   if os.path.exists(cache_file):
      lC, lD, lE, lH, lJ, dC, dD, dE, dH, dJ, lB = rp.util.load(cache_file)
   else:
      print('building rotamer clouds')
      chi_range = lambda resl: np.arange(-180, 180, resl)
      chi_asp = [chi_range(x) for x in (chiresl_asp1, chiresl_asp2)]
      chi_cys = [chi_range(x) for x in (chiresl_cys1, chiresl_cys2)]
      chi_his = [chi_range(x) for x in (chiresl_his1, chiresl_his2)]
      chi_glu = [chi_range(x) for x in (chiresl_glu1, chiresl_glu2, chiresl_glu3)]


      lC = mof.rotamer_cloud.RotCloudCysZN(grid=chi_cys, max_dun_score=4.0 * 1.5)
      lD = mof.rotamer_cloud.RotCloudAspZN(grid=chi_asp, max_dun_score=5.0 * 1.5)
      lE = mof.rotamer_cloud.RotCloudGluZN(grid=chi_glu, max_dun_score=5.0 * 1.5)
      lH = mof.rotamer_cloud.RotCloudHisZN(grid=chi_his, max_dun_score=5.0 * 1.5)
      lJ = mof.rotamer_cloud.RotCloudHisdZN(grid=chi_his, max_dun_score=5.0 * 1.5)
      dC = mof.rotamer_cloud.RotCloudDCysZN(grid=chi_cys, max_dun_score=4.0 * 1.5)
      dD = mof.rotamer_cloud.RotCloudDAspZN(grid=chi_asp, max_dun_score=5.0 * 1.5)
      dE = mof.rotamer_cloud.RotCloudDGluZN(grid=chi_glu, max_dun_score=5.0 * 1.5)
      dH = mof.rotamer_cloud.RotCloudDHisZN(grid=chi_his, max_dun_score=5.0 * 1.5)
      dJ = mof.rotamer_cloud.RotCloudDHisdZN(grid=chi_his, max_dun_score=5.0 * 1.5)

      lB = mof.rotamer_cloud.RotCloudBPY(grid=chi_his, max_dun_score=3.0)

      rp.util.dump([lC, lD, lE, lH, lJ, dC, dD, dE, dH, dJ, lB], cache_file)

   return dict(lC=lC, lD=lD, lE=lE, lH=lH, lJ=lJ, dC=dC, dD=dD, dE=dE, dH=dH, dJ=dJ,
lB=lB)

class RotamerCloud(ABC):
   """holds transforms for a set of rotamers positioned at the origin"""
   def __init__(
      self,
      amino_acid,
      rotchi=None,
      max_dun_score=4.0,
      grid=None,
   ):
      super(RotamerCloud, self).__init__()
      self.amino_acid = amino_acid
      sfxn_rotamer = get_sfxn('rotamer')
```

```
      pose = make_1res_pose(amino_acid)
      if rotchi is None:
         if grid is None:
            rotchi = util.get_rotamers(pose.residue(1))
            rotchi = np.array([list(x) for x in rotchi])
         else:
            mesh = np.meshgrid(*grid, indexing='ij')
            rotchi = np.stack(mesh, axis=len(mesh))
            rotchi = rotchi.reshape(-1, len(mesh))

      assert len(rotchi), 'no chi angles specified'
      self.original_rotchi = rotchi
      self.original_origin = _get_stub_1res(pose)
      xform_pose(pose, np.linalg.inv(self.original_origin))
      self.rotchi = list()
      self.rotbin = list()
      self.rotscore = list()
      self.frameidx = list()
      self.rotframes = list()
      for irot, chis in enumerate(rotchi):
         for ichi, chi in enumerate(chis):
            pose.set_chi(ichi + 1, 1, chi)
         dun = sfxn_rotamer(pose)
         if dun > max_dun_score: continue

         for iframe, frame in enumerate(self.get_effector_frame(pose.residue(1))):
            print(irot, iframe, chis)
            self.rotbin.append(irot)
            self.rotchi.append(chis)
            self.rotscore.append(dun)
            self.frameidx.append(iframe)
            self.rotframes.append(frame)

      assert self.rotbin, 'no chi angles pass dun cut'

      self.rotbin = np.array(self.rotbin)
      self.rotscore = np.array(self.rotscore)
      self.rotchi = np.stack(self.rotchi)
      self.frameidx = np.stack(self.frameidx)
      self.rotframes = np.stack(self.rotframes)

      print(
         f'created RotamerCloud {self.amino_acid} nrots:
{len(np.unique(self.rotbin))} nframes: {len(self.rotbin)}'
      )

   def make_pose1res(self):
      pose = make_1res_pose(self.amino_acid)
      xform_pose(pose, np.linalg.inv(self.original_origin))
      return pose

   def subset(self, which):
      new_one = copy.copy(self)
      new_one.rotchi = self.rotchi[which]
      new_one.rotbin = self.rotbin[which]
      new_one.rotscore = self.rotscore[which]
      new_one.rotframes = self.rotframes[which]
      return new_one

   @abstractmethod
   def get_effector_frame(self, residue):
      pass
```

```
   def dump_pdb(self, path=None, position=np.eye(4), which=None, append=False):
      if path is None: path = self.amino_acid + '.pdb'
      res = self.make_pose1res().residue(1)
      natm = res.natoms()
      F = rp.io.pdb_format_atom
      with open(path, 'a' if append else 'w') as out:
         loopey_doodle = enumerate(self.rotchi)
         if which is not None:
            loopey_doodle = ((which, self.rotchi[which]), )
         for irot, chis in loopey_doodle:
            out.write('MODEL %i\n' % irot)
            for ichi, chi in enumerate(chis):
               res.set_chi(ichi + 1, chi)
            for ia in range(1, natm + 1):
               xyz = res.xyz(ia)
               xyz = position @ np.array([xyz[0], xyz[1], xyz[2], 1])
               line = F(ia=ia, ir=1, an=res.atom_name(ia), rn=res.name3(), c='A',
xyz=xyz)
               out.write(line)
            orig = self.rotframes[irot, :, 3]
            x = orig + 0.5 * self.rotframes[irot, :, 0]
            y = orig + 0.5 * self.rotframes[irot, :, 1]
            z = orig + 0.5 * self.rotframes[irot, :, 2]
            orig = position @ orig
            x = position @ x
            y = position @ y
            z = position @ z
            out.write(F(ia=natm + 1, ir=1, an='ORIG', rn='END', c='B', xyz=orig))
            out.write(F(ia=natm + 2, ir=1, an='XDIR', rn='END', c='B', xyz=x,
elem='O'))
            out.write(F(ia=natm + 3, ir=1, an='YDIR', rn='END', c='B', xyz=y,
elem='CL'))
            out.write(F(ia=natm + 4, ir=1, an='ZDIR', rn='END', c='B', xyz=z,
elem='N'))
            out.write('ENDMDL\n')

   def __len__(self):
      return len(self.rotchi)



class RotCloudHisZN(RotamerCloud):
   def __init__(self, *args, **kw):
      super().__init__('HIS', *args, **kw)

   def get_effector_frame(self, residue):
      cd = residue.xyz('CD2')
      ce = residue.xyz('CE1')
      ne = residue.xyz('NE2')

      zn = rVec(0, 0, 0)
      for i in range(3):
         zn[i] = ne[i] - (cd[i] + ce[i]) / 2
      zn.normalize()
      for i in range(3):
         zn[i] = ne[i] + 2.2 * zn[i]

      return [rp.motif.frames.stub_from_points(zn, ne, ce).squeeze()]

class RotCloudDHisZN(RotCloudHisZN):
   def __init__(self, *args, **kw):
```

```
      RotamerCloud.__init__(self, 'DHIS', *args, **kw)

class RotCloudHisdZN(RotamerCloud):
   def __init__(self, *args, **kw):
      super(RotCloudHisdZN, self).__init__('HIS_D', *args, **kw)

   def get_effector_frame(self, residue):
      cg = residue.xyz('CG')
      ce = residue.xyz('CE1')
      nd = residue.xyz('ND1')

      zn = rVec(0, 0, 0)
      for i in range(3):
         zn[i] = nd[i] - (cg[i] + ce[i]) / 2
      zn /= np.linalg.norm(zn)
      for i in range(3):
         zn[i] = nd[i] + 2.2 * zn[i]

      return [rp.motif.frames.stub_from_points(zn, nd, ce).squeeze()]

class RotCloudDHisdZN(RotCloudHisdZN):
   def __init__(self, *args, **kw):
      RotamerCloud.__init__(self, 'DHIS_D', *args, **kw)

class RotCloudCysZN(RotamerCloud):
   def __init__(self, *args, **kw):
      super(RotCloudCysZN, self).__init__('CYS', *args, **kw)

   def get_effector_frame(self, residue):
      hg = residue.xyz('HG')
      sg = residue.xyz('SG')
      cb = residue.xyz('CB')
      orig = (hg - sg).normalized()
      for i in range(3):
         orig[i] = orig[i] * 2.32 + sg[i]
      frame = rp.motif.frames.stub_from_points(orig, sg, cb).squeeze()
      return [frame]

class RotCloudBPY(RotamerCloud):
   def __init__(self, *args, **kw):
      RotamerCloud.__init__(self, 'BPY', *args, **kw)

   def get_effector_frame(self, residue):
      ne1 = residue.xyz('NE1')
      nn1 = residue.xyz('NN1')
      fe = residue.xyz('FE')

      x = rp.homog.align_vectors([1, 0, 0], [0, 1, 0], ne1 - fe, nn1 - fe)
      x[:3, 3] = fe.x, fe.y, fe.z
      x1 = x @ rp.homog.align_vector([1, -1, 1], [0, 0, 1])
      x2 = x @ rp.homog.align_vector([1, -1, -1], [0, 0, 1])


      return [x1]  #, x2]

class RotCloudDCysZN(RotCloudCysZN):
   def __init__(self, *args, **kw):
      RotamerCloud.__init__(self, 'DCYS', *args, **kw)

class RotCloudAspZN(RotamerCloud):
   def __init__(self, *args, **kw):
      super(RotCloudAspZN, self).__init__('ASP', *args, **kw)
```

```
   def get_effector_frame(self, residue):
      return _asp_glu_effectors(residue)

class RotCloudDAspZN(RotCloudAspZN):
   def __init__(self, *args, **kw):
      RotamerCloud.__init__(self, 'DASP', *args, **kw)

class RotCloudGluZN(RotamerCloud):
   def __init__(self, *args, **kw):
      super(RotCloudGluZN, self).__init__('GLU', *args, **kw)

   def get_effector_frame(self, residue):
      return _asp_glu_effectors(residue)

class RotCloudDGluZN(RotCloudGluZN):
   def __init__(self, *args, **kw):
      RotamerCloud.__init__(self, 'DGLU', *args, **kw)

def _asp_glu_effectors(residue):
   if residue.name() in ('ASP', 'DASP'):
      names = 'CG', 'OD1', 'OD2'
   elif residue.name() in ('GLU', 'DGLU'):
      names = 'CD', 'OE1', 'OE2'
   else:
      raise NotImplementedError

   c = np.array(residue.xyz(names[0])).reshape(1, 3)
   o1 = np.array(residue.xyz(names[1])).reshape(1, 3)
   o2 = np.array(residue.xyz(names[2])).reshape(1, 3)



   orig = (c - o2) / np.linalg.norm(c - o2)
   orig = o1 + orig * 2.1

   c = rp.homog.hpoint(c)
   o1 = rp.homog.hpoint(o1)
   o2 = rp.homog.hpoint(o2)
   orig = rp.homog.hpoint(orig)
   rotaxis = rp.homog.hcross(o1 - c, o2 - c)
   rot = rp.homog.hrot(rotaxis, 10, o1, degrees=True).squeeze()
   frames = list()
   for irot in range(13):
      frame = rp.motif.frames.stub_from_points(orig, o1, c).squeeze()
      frames.append(frame)
      orig = (rot @ orig.squeeze()).reshape(1, 4)
   return frames

def _get_stub_1res(pose):
   res = pose.residue(1)
   n = res.xyz('N')
   ca = res.xyz('CA')
   c = res.xyz('C')
   return rp.motif.frames.bb_stubs(
      np.array([[n[0], n[1], n[2]]]),
      np.array([[ca[0], ca[1], ca[2]]]),
      np.array([[c[0], c[1], c[2]]]),
   ).squeeze()

# for pickle test file compatibility... this is very lazy... should regen
RotamerCloudAspZN = RotCloudAspZN
```

```
RotamerCloudCysZN = RotCloudCysZN
RotamerCloudGluZN = RotCloudGluZN
RotamerCloudHisZN = RotCloudHisZN
```

Listing S2: We define geometric parameters for each space group such as the axes for the symmetry components.

```
import numpy as np, rpxdock as rp
from rpxdock import homog as hm
from mof.data import data_dir

class XtalSpec:
   pass

class XtalSpecCC(XtalSpec):
   def __init__(
      self,
      spacegroup,
      nfold1,
      axis1,
      orig1,
      nfold2,
      axis2,
      orig2,
      nsubs,
   ):

      self.spacegroup = spacegroup
      self.nfold1 = int(nfold1)
      self.sym1 = 'C%i' % nfold1
      self.axis1 = hm.hnormalized(hm.hvec(axis1))
      self.orig1 = hm.hpoint(orig1)
      self.nfold2 = int(nfold2)
      self.sym2 = 'C%i' % nfold2
      self.axis2 = hm.hnormalized(hm.hvec(axis2))
      if hm.angle(self.axis1, self.axis2) > np.pi / 2:
         self.axis2[:3] = -self.axis2[:3]
      assert 90 > hm.angle_degrees(self.axis1, self.axis2)
      self.orig2 = hm.hpoint(orig2)
      self.nsubs = nsubs
      self.dihedral = np.degrees(hm.angle(axis1, axis2))
      self.axis1d = None
      self.axis2d = None
      try:
         self.frames = rp.load(_frames_files[spacegroup])
      except:
         self.frames = None

class XtalSpecCD(XtalSpec):
   def __init__(
      self,
      spacegroup,
      nfold1,
      axis1,
      orig1,
      nfold2,
      axis2,
```

```
      orig2,
      axis2d,
      nsubs,
   ):
      self.spacegroup = spacegroup
      self.nfold1 = int(nfold1)
      self.sym1 = 'C%i' % nfold1
      self.axis1 = hm.hnormalized(hm.hvec(axis1))
      self.orig1 = hm.hpoint(orig1)
      self.nfold2 = int(nfold2)
      self.sym2 = 'C%i' % nfold2
      self.axis2 = hm.hnormalized(hm.hvec(axis2))
      if hm.angle(self.axis1, self.axis2) > np.pi / 2:
         self.axis2[:3] = -self.axis2[:3]
      assert 90 > hm.angle_degrees(self.axis1, self.axis2)

      self.orig2 = hm.hpoint(orig2)
      self.nsubs = nsubs
      self.dihedral = np.degrees(hm.angle(axis1, axis2))
      self.axis1d = None
      self.axis2d = hm.hnormalized(hm.hvec(axis2d))
      try:
         self.frames = rp.load(_frames_files[spacegroup])
      except:
         self.frames = None

_frames_files = {
   'I 21 3': data_dir + '/i213_redundant111_n16_maxrad2.pickle',
   'P 41 3 2': data_dir + '/p4132_trionly_n12_maxrad3.pickle',
   'P 43 3 2': data_dir + '/p4132_trionly_n12_maxrad3.pickle',
}

def get_xtal_spec(name):
   try:
      return _xspec[name.lower()]
   except KeyError:
      print(f'spacegroup {name} not implemented')


_xspec = dict(
   f432=XtalSpecCC(
      'F 4 3 2',
      3,
      [1, 1, 1],
      [0.5, 0.5, 0],
      4,
      [1, 0, 0],
      [0, 0, 0],
      24,
   ),
   p213=XtalSpecCC(
      'P 21 3',
      3,
      [+1, +1, +1],
      [0, 0, 0],
      3,
      [-1, +1, +1],
      [+0., +0, +0.5],
      12,
   ),
   i213=XtalSpecCC(
      'I 21 3',
```

```
      3,
      [1, 1, 1],
      [0, 0, 0],
      2,
      [0, 0, 1],
      [0, -0.25, 0],
      12,
   ),

   p4132=XtalSpecCC(
      'P 41 3 2',
      3,
      [-1, -1, 1],
      [-0.5, 0, -0.5],
      2,
      [0, -1, 1],
      [-0.125, -0.125, -0.125],
      24,
   ),
   p4332=XtalSpecCC(
      'P 43 3 2',
      3,
      [-1, -1, 1],
      [-0.5, 0, -0.5],
      2,
      [0, -1, 1],
      [-0.375, -0.125, -0.125],
      24,
   ),
   p23=XtalSpecCD(
      'P 2 3',
      3,
      [1, -1, 1],
      [-2 / 3, -1 / 3, 1 / 3],
      2,
      [0, 1, 0],
      [-1 / 2, -1 / 2, 0],
      [1, 0, 0],
      12,
   ))
```

Listing S3: We dock C3 peptides and C3 octahedral metal sites into space group $P2_13$.

```
import mof, os, numpy as np, rpxdock as rp, rpxdock.homog as hm
from mof.pyrosetta_init import (rosetta, makelattice, get_sfxn, xform_pose,
make_residue)
from mof.util import align_cx_pose_to_z, variant_remove
from pyrosetta.rosetta.numeric import xyzVector_double_t as xyzVec
from pyrosetta import AtomID, get_score_function

def main_loop():

   kw = mof.app.options_setup(get_test_kw, verbose=False)

   if kw.postprocess:
      return mof.app.postprocess(kw)

   pept_axis = np.array([0, 0, 1, 0])
```

```
   pept_orig = np.array([0, 0, 0, 1])

   results = list()
   rfname = f'{kw.output_prefix}results.pickle'
   print('fname prefix:', kw.output_prefix)
   print('result fname:', rfname)

   sfxn = get_score_function()

   results = list()

   for ipdbpath, pdbpath in enumerate(kw.inputs):

      pose = rosetta.core.import_pose.pose_from_file(pdbpath)
      if not align_cx_pose_to_z(pose, pdbpath):
         print(f"WARNING failed align_cx_pose_to_z: {pdbpath}")
      variant_remove(pose)
      rpxbody = rp.Body(pose)

      rotclouds = mof.rotamer_cloud.get_rotclouds(**kw)

      minscore = 9e9

      print()
      print(f'{" %i of %i "%(ipdbpath+1 , len(kw.inputs)):#^80}')
      print(pdbpath)
      print(f'{"":#^80}')

      for spacegroup in kw.spacegroups:

         search_spec = mof.xtal_search.XtalSearchSpec(
            spacegroup=spacegroup,
            pept_orig=np.array([0, 0, 0, 1]),
            pept_axis=np.array([0, 0, 1, 0]),
            # are these necessary:
            sym_of_ligand=dict(HZ3='C3', DHZ3='C3', HZ4='C4', DHZ4='C4', HZD='D2',
DHZD='D2',
                               BPY='C3'),
            ligands=['HZ3', 'DHZ3'],
            **kw,
         )
         xspec = search_spec.xtal_spec
         target_angle = hm.line_angle_degrees(xspec.axis1, xspec.axis2)

         sym_num = xspec.nfold1
         nresasym = len(pose.residues) // sym_num

         for iaa, aa in enumerate(kw.aa_labels):

            rotcloud = rotclouds[mof.app.lblmap[aa]]

            for ires in range(nresasym):

               print(f'{f" LOOP {spacegroup} {aa} {ires} ":*^80}')
               stub = rpxbody.stub[ires]

               rotframes = stub @ rotcloud.rotframes

               for irot, rotframe in enumerate(rotframes):
                  ligsymaxis = rotframe[:, 2]  # z axis

                  if np.pi / 2 < hm.angle(pept_axis, ligsymaxis):
```

```
                    ligsymaxis *= -1

                angle = hm.angle_degrees(pept_axis, ligsymaxis)
                if abs(angle - target_angle) > kw.angle_err_tolerance:
                    continue


                orig_metal_pos = rotframe[:, 3]


                xalign, delta = hm.align_lines_isect_axis2(
                    pept_orig, pept_axis, orig_metal_pos, ligsymaxis, xspec.axis1,
xspec.orig1,
                    xspec.axis2, xspec.orig2 - xspec.orig1, strict=False)


                aligned_pept_axis = xalign @ pept_axis
                aligned_ligsym_axis = xalign @ ligsymaxis
                aligned_metal_pos = xalign @ orig_metal_pos
                assert np.allclose(aligned_pept_axis, xspec.axis1,
atol=kw.angle_err_tolerance)
                assert np.allclose(aligned_ligsym_axis, xspec.axis2,
                                   atol=kw.angle_err_tolerance)

                _, isect = hm.line_line_closest_points_pa(aligned_metal_pos,
                                                          aligned_ligsym_axis, [0,
0, 0, 1],
                                                          xspec.orig2 -
xspec.orig1)
                cell_spacing = abs(isect[2] / xspec.orig2[2])
                if cell_spacing < 10 or cell_spacing > 25:
                    continue


                outpose0 = mof.util.mutate_one_res(pose, ires + 1, aa,
rotcloud.rotchi[irot],
                                                   sym_num)
                xform_pose(outpose0, xalign)
                outasym = rosetta.protocols.grafting.return_region(outpose0, 1,
nresasym)
                ci = rosetta.core.io.CrystInfo()
                ci.A(cell_spacing)  # cell dimensions
                ci.B(cell_spacing)
                ci.C(cell_spacing)
                ci.alpha(90)  # cell angles n
                ci.beta(90)
                ci.gamma(90)
                ci.spacegroup(xspec.spacegroup)  # sace group
                pi = rosetta.core.pose.PDBInfo(outasym)
                pi.set_crystinfo(ci)
                outasym.pdb_info(pi)

                sympose = outasym.clone()
                rosetta.protocols.cryst.MakeLatticeMover().apply(sympose)
                conf = sympose.conformation().clone()
                assert rosetta.core.conformation.symmetry.is_symmetric(conf)
                syminfo = rosetta.core.pose.symmetry.symmetry_info(sympose)

                nasym = outasym.size()
                conf.declare_chemical_bond(nasym, 'C', 1, 'N')
                sympose.set_new_conformation(conf)
                sympose.set_new_energies_object(
```

```
                     rosetta.core.scoring.symmetry.SymmetricEnergies())

                  sc = sfxn.score(sympose)
                  minscore = min(minscore, sc)
                  if sc > 7000:
                     continue

                  syminfo = rosetta.core.pose.symmetry.symmetry_info(sympose)
                  surfvol = rosetta.core.scoring.packing.get_surf_vol(sympose, 1.4)
                  peptvol = 0.0
                  for ir in range(1, syminfo.get_nres_subunit() + 1):
                     res = sympose.residue(ir)
                     ir = ir + 1  # rosetta numbering
                     for ia in range(1, res.natoms() + 1):
                        v = surfvol.vol[AtomID(ia, ir)]
                        if not np.isnan(v): peptvol += v
                  peptvol *= xspec.nsubs
                  print(f'peptvol {peptvol} {peptvol / cell_spacing**3}')
                  solv_frac = max(0.0, 1.0 - peptvol / cell_spacing**3)

                  tag = ''.join([
                     f'_cell{int(cell_spacing):03}_',
                     f'_sc{int(sc):05}_',
                     f'_solv{int(solv_frac*100):02}_',
                     f'{os.path.basename(pdbpath)}',
                     f'_nres{nresasym}',
                     f'{aa}_',
                     f'{len(results):06}',
                  ])
                  fn = kw.output_prefix + tag + '.pdb'

                  print('HIT %7.3f' % sc, outasym)
                  results.append([sc, cell_spacing, sympose])
                  if True:
                     outasym.dump_pdb(fn)

      print('minscore', minscore)

   results.sort

   if not results:
      print(f'{"":!^100}')
      print('NO RESULTS!!!')
      print(f'{"":!^100}')
      print('DONE')

   return results

validated_c3 = [
   '<path_to_validated_peptides/peptide.pdb>',
]

def get_test_kw(kw):
   if not kw.inputs:
      kw.inputs = ['mof/data/peptides/c.2.6_0001.pdb']
      # kw.inputs = validated_c3
      print(f'{"":!^80}')
      print(f'{"no pdb list input, using test only_one":!^80}')
      print(f'{str(kw.inputs):!^80}')
      print(f'{"":!^80}')

   kw.spacegroups = ['p213']
```

```
   kw.aa_labels = ['BPY']
   kw.output_prefix = '_mof_test_c3c3' + '_'.join(kw.spacegroups) + '/'
   kw.angle_err_tolerance = 3
   kw.scale_number_of_rotamers = 0.25
   kw.max_bb_redundancy = 2.0
   kw.max_dun_score = 4.0
   kw.clash_dis = 3.3
   kw.contact_dis = 7.0
   kw.min_contacts = 0
   kw.max_score_minimized = 40.0
   kw.min_cell_size = 0
   kw.max_cell_size = 50
   kw.max_solv_frac = 0.80
   kw.debug = True
   # kw.continue_from_checkpoints = False
   return kw
```

Listing S4: We dock C3 peptides and D2 tetrahedral metal sites into *P*23 space group.

```
import mof, os, numpy as np, rpxdock as rp, rpxdock.homog as hm
from mof.pyrosetta_init import (rosetta, makelattice, get_sfxn, xform_pose,
make_residue)
from mof.util import align_cx_pose_to_z, variant_remove
from pyrosetta.rosetta.numeric import xyzVector_double_t as xyzVec
from pyrosetta import AtomID, get_score_function

def main_loop():

   kw = mof.app.options_setup(get_test_kw, verbose=False)

   if kw.postprocess:
      return mof.app.postprocess(kw)

   pept_axis = np.array([0, 0, 1, 0])
   pept_orig = np.array([0, 0, 0, 1])

   results = list()
   rfname = f'{kw.output_prefix}results.pickle'
   print('fname prefix:', kw.output_prefix)
   print('result fname:', rfname)

   sfxn = get_score_function()

   results = list()

   for ipdbpath, pdbpath in enumerate(kw.inputs):

      pose = rosetta.core.import_pose.pose_from_file(pdbpath)
      if not align_cx_pose_to_z(pose, pdbpath):
         print(f"WARNING failed align_cx_pose_to_z: {pdbpath}")
      variant_remove(pose)
      rpxbody = rp.Body(pose)

      rotclouds = mof.rotamer_cloud.get_rotclouds(**kw)

      minscore = 9e9

      print()
```

```
      print(f'{" %i of %i "%(ipdbpath+1 , len(kw.inputs)):#^80}')
      print(pdbpath)
      print(f'{"":#^80}')

      for spacegroup in kw.spacegroups:

         search_spec = mof.xtal_search.XtalSearchSpec(
            spacegroup=spacegroup,
            pept_orig=np.array([0, 0, 0, 1]),
            pept_axis=np.array([0, 0, 1, 0]),
            # are these necessary:
            sym_of_ligand=dict(HZ3='C3', DHZ3='C3', HZ4='C4', DHZ4='C4', HZD='D2',
DHZD='D2',
                               BPY='C3'),
            ligands=['HZ3', 'DHZ3'],
            **kw,
         )
         xspec = search_spec.xtal_spec
         target_angle = hm.line_angle_degrees(xspec.axis1, xspec.axis2)

         sym_num = xspec.nfold1
         nresasym = len(pose.residues) // sym_num

         for iaa, aa in enumerate(kw.aa_labels):

            rotcloud = rotclouds[mof.app.lblmap[aa]]

            for ires in range(nresasym):

               print(f'{f" LOOP {spacegroup} {aa} {ires} ":*^80}')
               stub = rpxbody.stub[ires]

               rotframes = stub @ rotcloud.rotframes

               for irot, rotframe in enumerate(rotframes):
                  ligsymaxis = rotframe[:, 2]  # z axis

                  if np.pi / 2 < hm.angle(pept_axis, ligsymaxis):
                     ligsymaxis *= -1

                  angle = hm.angle_degrees(pept_axis, ligsymaxis)
                  if abs(angle - target_angle) > kw.angle_err_tolerance:
                     continue


                  xalign, delta = hm.align_lines_isect_axis2(
                     pept_orig, pept_axis, orig_metal_pos, ligsymaxis, xspec.axis1,
xspec.orig1,
                     xspec.axis2, xspec.orig2 - xspec.orig1, strict=False)

                  aligned_pept_axis = xalign @ pept_axis
                  aligned_ligsym_axis = xalign @ ligsymaxis
                  aligned_metal_pos = xalign @ orig_metal_pos
                  assert np.allclose(aligned_pept_axis, xspec.axis1,
atol=kw.angle_err_tolerance)
                  assert np.allclose(aligned_ligsym_axis, xspec.axis2,
                                     atol=kw.angle_err_tolerance)

                  _, isect = hm.line_line_closest_points_pa(aligned_metal_pos,
                                                            aligned_ligsym_axis, [0,
0, 0, 1],
                                                            xspec.orig2 -
```

```
xspec.orig1)
                  cell_spacing = abs(isect[2] / xspec.orig2[2])
                  if cell_spacing < 10 or cell_spacing > 25:
                     continue

                  outpose0 = mof.util.mutate_one_res(pose, ires + 1, aa,
rotcloud.rotchi[irot],
                                                     sym_num)
                  xform_pose(outpose0, xalign)
                  outasym = rosetta.protocols.grafting.return_region(outpose0, 1,
nresasym)
                  ci = rosetta.core.io.CrystInfo()
                  ci.A(cell_spacing)  # cell dimensions
                  ci.B(cell_spacing)
                  ci.C(cell_spacing)
                  ci.alpha(90)  # cell angles n
                  ci.beta(90)
                  ci.gamma(90)
                  ci.spacegroup(xspec.spacegroup)  # sace group
                  pi = rosetta.core.pose.PDBInfo(outasym)
                  pi.set_crystinfo(ci)
                  outasym.pdb_info(pi)

                  sympose = outasym.clone()
                  rosetta.protocols.cryst.MakeLatticeMover().apply(sympose)
                  conf = sympose.conformation().clone()
                  assert rosetta.core.conformation.symmetry.is_symmetric(conf)
                  syminfo = rosetta.core.pose.symmetry.symmetry_info(sympose)

                  nasym = outasym.size()
                  conf.declare_chemical_bond(nasym, 'C', 1, 'N')
                  sympose.set_new_conformation(conf)
                  sympose.set_new_energies_object(
                     rosetta.core.scoring.symmetry.SymmetricEnergies())

                  sc = sfxn.score(sympose)
                  minscore = min(minscore, sc)
                  if sc > 7000:
                     continue

                  syminfo = rosetta.core.pose.symmetry.symmetry_info(sympose)
                  surfvol = rosetta.core.scoring.packing.get_surf_vol(sympose, 1.4)
                  peptvol = 0.0
                  for ir in range(1, syminfo.get_nres_subunit() + 1):
                     res = sympose.residue(ir)
                     ir = ir + 1  # rosetta numbering
                     for ia in range(1, res.natoms() + 1):
                        v = surfvol.vol[AtomID(ia, ir)]
                        if not np.isnan(v): peptvol += v
                  peptvol *= xspec.nsubs
                  print(f'peptvol {peptvol} {peptvol / cell_spacing**3}')
                  solv_frac = max(0.0, 1.0 - peptvol / cell_spacing**3)

                  tag = ''.join([
                     f'_cell{int(cell_spacing):03}_',
                     f'_sc{int(sc):05}_',
                     f'_solv{int(solv_frac*100):02}_',
                     f'{os.path.basename(pdbpath)}',
                     f'_nres{nresasym}',
                     f'{aa}_',
                     f'{len(results):06}',
                  ])
```

```
                  fn = kw.output_prefix + tag + '.pdb'

                  print('HIT %7.3f' % sc, outasym)
                  results.append([sc, cell_spacing, sympose])
                  if True:
                     outasym.dump_pdb(fn)

      print('minscore', minscore)

   results.sort

   if not results:
      print(f'{"":!^100}')
      print('NO RESULTS!!!')
      print(f'{"":!^100}')
      print('DONE')

   return results

validated_c3 = [
'<path_to_validated_pdbs/peptide.pdb>',
]

def get_test_kw(kw):
   if not kw.inputs:
      kw.inputs = ['mof/data/peptides/c.2.6_0001.pdb']
      print(f'{"":!^80}')
      print(f'{"no pdb list input, using test only_one":!^80}')
      print(f'{str(kw.inputs):!^80}')
      print(f'{"":!^80}')

   kw.spacegroups = ['p213']
   kw.aa_labels = ['BPY']
   kw.output_prefix = '_mof_test_c3c3' + '_'.join(kw.spacegroups) + '/'
   kw.angle_err_tolerance = 3
   kw.scale_number_of_rotamers = 0.25
   kw.max_bb_redundancy = 2.0
   kw.max_dun_score = 4.0
   kw.clash_dis = 3.3
   kw.contact_dis = 7.0
   kw.min_contacts = 0
   kw.max_score_minimized = 40.0
   kw.min_cell_size = 0
   kw.max_cell_size = 50
   kw.max_solv_frac = 0.80
   kw.debug = True
   # kw.continue_from_checkpoints = False
   return kw
```

Listing S5: We dock C3 peptides and C2 metal sites into three space groups ($I2_13$, $P4_132$ and $P4_332$).

```
import mof, rpxdock as rp, numpy as np
from rpxdock import homog as hm

from mof.pyrosetta_init import get_sfxn
from pyrosetta import rosetta as rosetta
from pyrosetta.rosetta.core.pose import Pose
from pyrosetta.rosetta.core.id import AtomID
```

```
from pyrosetta.rosetta.numeric import xyzVector_double_t as rVec
from pyrosetta import rosetta as rt, init as pyrosetta_init

class XtalSearchSpec(object):
   """stuff needed for pepdide xtal search"""
   def __init__(
         self,
         spacegroup,
         pept_axis,
         pept_orig,
         ligands,
         sym_of_ligand,
         max_dun_score,
         **kw,
   ):
      super(XtalSearchSpec, self).__init__()
      kw = rp.Bunch(kw)
      self.spacegroup = spacegroup
      self.pept_axis = pept_axis
      self.pept_orig = pept_orig
      self.ligands = ligands
      self.sym_of_ligand = sym_of_ligand
      self.max_dun_score = max_dun_score
      self.xtal_spec = mof.xtal_spec.get_xtal_spec(self.spacegroup)
      self.chm = rt.core.chemical.ChemicalManager.get_instance()
      self.rts = self.chm.residue_type_set('fa_standard')

      self.sfxn_rotamer = get_sfxn('rotamer')

      self.sfxn_sterics = get_sfxn('sterics')

      self.sfxn_minimize = get_sfxn('minimize')


def xtal_search_two_residues(
      search_spec,
      pose,
      rotcloud1base,
      rotcloud2base,
      err_tolerance,
      dist_err_tolerance,
      angle_err_tolerance,
      min_dist_to_z_axis,
      sym_axes_angle_tolerance,
      angle_to_cart_err_ratio,
      debug=False,
      **kw,
):
   kw = rp.Bunch(kw)
   if not kw.timer: kw.timer = rp.Timer().start()
   kw.timer.checkpoint()

   spec = search_spec
   xspec = spec.xtal_spec
   aa1 = rotcloud1base.amino_acid
   aa2 = rotcloud2base.amino_acid

   results = list()

   dont_replace_these_aas = [spec.rts.name_map(aa) for aa in
kw.dont_replace_these_aas]
```

```
   farep_orig = search_spec.sfxn_sterics(pose)

   p_n = pose.pdb_info().name().split('/')[-1]
   # gets rid of the ".pdb" at the end of the pdb name
   pdb_name = p_n[:-4]

   print(f'  {pdb_name} searching', aa1, aa2)

   # check the symmetry type of the pdb
   last_res = rt.core.pose.chain_end_res(pose).pop()
   total_res = int(last_res)

   sym_num = 3
   sym = 3
   if sym_num < 2:
      print('bad pdb', p_n)
      return list()
   asym_nres = int(total_res / sym)
   peptide_sym = "C%i" % sym_num

   rpxbody = rp.Body(pose)

   for ires1 in range(1, asym_nres + 1):
      if pose.residue_type(ires1) in dont_replace_these_aas: continue
      stub1 = rpxbody.stub[ires1 - 1]

      kw.timer.checkpoint('xtal_search')
      rots1ok = min_dist_to_z_axis < np.linalg.norm(
         (stub1 @ rotcloud1base.rotframes)[:, :2, 3], axis=1)
      if 0 == np.sum(rots1ok): continue
      rotcloud1 = rotcloud1base.subset(rots1ok)
      rotframes1 = stub1 @ rotcloud1.rotframes

      kw.timer.checkpoint('position rotcloud')

      range2 = range(1, int(total_res) + 1)
      if rotcloud1base is rotcloud2base: range2 = range(ires1 + 1, int(total_res) +
1)
      for ires2 in range2:
         if ires1 == ((ires2 - 1) % asym_nres + 1): continue
         if pose.residue_type(ires2) in dont_replace_these_aas:
            continue
         stub2 = rpxbody.stub[ires2 - 1]

         kw.timer.checkpoint('xtal_search')
         rots2ok = min_dist_to_z_axis < np.linalg.norm(
            (stub2 @ rotcloud2base.rotframes)[:, :2, 3], axis=1)
         if 0 == np.sum(rots2ok): continue
         rotcloud2 = rotcloud2base.subset(rots2ok)
         rotframes2 = stub2 @ rotcloud2.rotframes

         kw.timer.checkpoint('rotcloud positioning')

         dist = rotframes1[:, :, 3].reshape(-1, 1, 4) - rotframes2[:, :,
3].reshape(1, -1, 4)
         dist = np.linalg.norm(dist, axis=2)

         kw.timer.checkpoint('rotcloud dist')

         dot = np.sum(
            rotframes1[:, :, 0].reshape(-1, 1, 4) * rotframes2[:, :, 0].reshape(1,
-1, 4), axis=2)
```

```
         ang = np.degrees(np.arccos(np.clip(dot, -1, 1)))
         ang_delta = np.abs(ang - 109.4712206)

         kw.timer.checkpoint('rotcloud ang')

         err = np.sqrt((ang_delta / angle_to_cart_err_ratio)**2 + dist**2)
         rot1err2 = np.min(err, axis=1)
         bestrot2 = np.argmin(err, axis=1)
         disterr = dist[np.arange(len(bestrot2)), bestrot2]
         angerr = ang_delta[np.arange(len(bestrot2)), bestrot2]
         ok = (rot1err2 < err_tolerance)
         ok *= (angerr < angle_err_tolerance)
         ok *= (disterr < dist_err_tolerance)

         hits1 = np.argwhere(ok).reshape(-1)

         kw.timer.checkpoint('rotcloud match check')

         if len(hits1):
            hits2 = bestrot2[hits1]
            hits = np.stack([hits1, hits2], axis=1)
            for ihit, hit in enumerate(hits):
               frame1 = rotframes1[hit[0]]
               frame2 = rotframes2[hit[1]]

               kw.timer.checkpoint('xtal_search')

               parl = (frame1[:, 0] + frame2[:, 0]) / 2.0
               perp = rp.homog.hcross(frame1[:, 0], frame2[:, 0])
               metalaxis1 = rp.homog.hrot(parl, +45) @ perp
               metalaxis2 = rp.homog.hrot(parl, -45) @ perp
               symang1 = rp.homog.line_angle(metalaxis1, spec.pept_axis)
               symang2 = rp.homog.line_angle(metalaxis2, spec.pept_axis)
               match1 = np.abs(np.degrees(symang1) - xspec.dihedral) <
sym_axes_angle_tolerance
               match2 = np.abs(np.degrees(symang2) - xspec.dihedral) <
sym_axes_angle_tolerance
               if not (match1 or match2): continue
               matchsymang = symang1 if match1 else symang2
               metal_axis = metalaxis1 if match1 else metalaxis2
               if rp.homog.angle(metal_axis, spec.pept_axis) > np.pi / 2:
                  metal_axis[:3] = -metal_axis[:3]
               metal_pos = (rotframes1[hit[0], :, 3] + rotframes2[hit[1], :, 3]) /
2.0

               correction_axis = rp.homog.hcross(metal_axis, spec.pept_axis)
               correction_angle = np.abs(matchsymang - np.radians(xspec.dihedral))


               for why_do_i_need_this in (-correction_angle, correction_angle):
                  metal_axis_try = rp.homog.hrot(correction_axis,
why_do_i_need_this) @ metal_axis
                  if np.allclose(rp.homog.angle_degrees(metal_axis_try,
spec.pept_axis),
                                 xspec.dihedral, atol=0.001):
                     metal_axis = metal_axis_try
                     break

               assert np.allclose(rp.homog.angle_degrees(metal_axis,
spec.pept_axis),
                                  xspec.dihedral, atol=0.001)
```

```
                kw.timer.checkpoint('axes geom checks')


                pose2mut = mof.util.mutate_two_res(pose, ires1, aa1,
rotcloud1.rotchi[hit[0]],
                                                   ires2, aa2,
rotcloud2.rotchi[hit[1]], sym_num)

                search_spec.sfxn_sterics(pose2mut)
                sc_2res = (pose2mut.energies().residue_total_energy(ires1) +
                           pose2mut.energies().residue_total_energy(ires2))
                sc_2res_orig = (pose.energies().residue_total_energy(ires1) +
                                pose.energies().residue_total_energy(ires2))

                kw.timer.checkpoint('mut_two_res')

                if sc_2res - sc_2res_orig > kw.max_2res_score: continue

                tag = ('hit_%s_%s_%i_%i_%i' % (aa1, aa2, ires1, ires2, ihit))

                kw.timer.checkpoint('xtal_search')

                xtal_poses = mof.xtal_build.xtal_build(
                   pdb_name,
                   xspec,
                   aa1,
                   aa2,
                   pose2mut,
                   peptide_sym,
                   spec.pept_orig,
                   spec.pept_axis,
                   'C2',
                   metal_pos,
                   metal_axis,
                   rpxbody,
                   tag,
                   **kw,
                )
                if not xtal_poses: continue


                kw.timer.checkpoint('xtal_search')

                for ixtal, (xalign, xtal_pose, body_pdb, ncontact, enonbonded,
                            solv_frac) in enumerate(xtal_poses):

                   xtal_pose_min, mininfo = mof.minimize.minimize_mof_xtal(
                      spec.sfxn_minimize,
                      xspec,
                      xtal_pose,
                      **kw,
                   )
                   if not xtal_pose_min:
                      continue
                   if kw.max_score_minimized < mininfo.score:
                      continue
                      print('     ', xspec.spacegroup, pdb_name, aa1, aa2, 'a on
minimzied score',
                            mininfo.score)
                      continue
                   celldim = xtal_pose.pdb_info().crystinfo().A()
                   label = f"{pdb_name}_{xspec.spacegroup.replace('
```

```
','_')}_{tag}_cell{int(celldim):03}_ncontact{ncontact:02}_score{int(enonbonded):03}"

                  info = mininfo.sub(  # adding to mininfo
                     label=label,
                     xalign=xalign,
                     ncontact=ncontact,
                     enonbonded=enonbonded,
                     sequence=','.join(r.name() for r in xtal_pose.residues),
                     solv_frac=solv_frac,
                     celldim=celldim,
                     spacegroup=xspec.spacegroup,
                     nsubunits=xspec.nsubs,
                     nres=xtal_pose_min.size() - 1,
                  )
                  bbcoords = np.array(
                     [(v[0], v[1], v[2]) for v in [[r.xyz(n)
                                                    for n in ('N', 'CA', 'C')]
                                                   for r in
xtal_pose_min.residues[:-1]]])
                  bbpad = np.zeros(shape=(kw.max_pept_size - xtal_pose_min.size() +
1, 3, 3))
                  info['bbcoords'] = np.concatenate([bbcoords, bbpad])

                  results.append(
                     rp.Bunch(
                        xspec=xspec,
                        asym_pose_min=xtal_pose_min,
                        info=info,
                     ))
                  ### debug crap
                  if results:
                     print('  * HIT %10s %7s %7s %3i %3i %9s %-7.3f %5.3f %s' % (
                        xspec.spacegroup.replace(' ', '_'),
                        aa1,
                        aa2,
                        ires1,
                        ires2,
                        xtal_pose_min.sequence(),
                        info.score,
                        info.solv_frac,
                        pdb_name,
                     ))
                  ### end debug crap

               kw.timer.checkpoint('build_result')

   kw.timer.checkpoint('xtal_search')

   return results

def xtal_search_single_residue(search_spec, pose, **kw):
   raise NotImplemntedError('xtal_search_single_residue needs updating')
   kw = rp.Bunch(kw)

   spec = search_spec
   xspec = spec.xtal_spec

   results = list()

   p_n = pose.pdb_info().name().split('/')[-1]
   pdb_name = p_n[:-4]
```

```
   print(f'{pdb_name} searching')

   # check the symmetry type of the pdb
   last_res = rt.core.pose.chain_end_res(pose).pop()
   total_res = int(last_res)
   sym_num = pose.chain(last_res)
   if sym_num < 2:
      print('bad pdb', p_n)
      return list()
   sym = int(sym_num)
   peptide_sym = "C%i" % sym_num

   for ires in range(1, int(total_res / sym) + 1):
      if pose.residue_type(ires) not in (spec.rts.name_map('GLY'),
spec.rts.name_map('ALA'),
                                         spec.rts.name_map('DALA')):
         continue
      lig_poses = util.mut_to_ligand(pose, ires, spec.ligands, spec.sym_of_ligand)
      bad_rots = 0
      for ilig, lig_pose in enumerate(lig_poses):
         mut_res_name, lig_sym = lig_poses[lig_pose]

         rotamers = lig_pose.residue(ires).get_rotamers()
         rotamers = util.extra_rotamers(rotamers, lb=-20, ub=21, bs=20)

         pose_num = 1
         for irot, rotamer in enumerate(rotamers):
            for i in range(len(rotamer)):
               lig_pose.residue(ires).set_chi(i + 1, rotamer[i])
            rot_pose = rt.protocols.grafting.return_region(lig_pose, 1,
lig_pose.size())

            if kw.debug:
               rot_pose.set_xyz(AtomID(rot_pose.residue(ires).atom_index('1HB'),
ires),
                                rVec(0, 0, -2))
               rot_pose.set_xyz(AtomID(rot_pose.residue(ires).atom_index('CB'),
ires),
                                rVec(0, 0, +0.0))
               rot_pose.set_xyz(AtomID(rot_pose.residue(ires).atom_index('2HB'),
ires),
                                rVec(0, 0, +2))

            spec.sfxn_rotamer(rot_pose)
            dun_score = rot_pose.energies().residue_total_energy(ires)
            if dun_score >= spec.max_dun_score:
               bad_rots += 1
               continue
            rpxbody = rp.Body(rot_pose)


            metal_orig = hm.hpoint(util.coord_find(rot_pose, ires, 'VZN'))
            hz = hm.hpoint(util.coord_find(rot_pose, ires, 'HZ'))
            ne = hm.hpoint(util.coord_find(rot_pose, ires, 'VNE'))
            metal_his_bond = hm.hnormalized(metal_orig - ne)
            metal_sym_axis0 = hm.hnormalized(hz - metal_orig)
            dihedral = xspec.dihedral


            rots_around_nezn = hm.xform_around_dof_for_vector_target_angle(
               fix=spec.pept_axis, mov=metal_sym_axis0, dof=metal_his_bond,
               target_angle=np.radians(dihedral))
```

```
            for idof, rot_around_nezn in enumerate(rots_around_nezn):
               metal_sym_axis = rot_around_nezn @ metal_sym_axis0
               assert np.allclose(hm.line_angle(metal_sym_axis, spec.pept_axis),
                                  np.radians(dihedral))

               newhz = util.coord_find(rot_pose, ires, 'VZN') + 2 *
metal_sym_axis[:3]

               aid = rt.core.id.AtomID(rot_pose.residue(ires).atom_index('HZ'),
ires)
               xyz = rVec(newhz[0], newhz[1], newhz[2])
               rot_pose.set_xyz(aid, xyz)

               tag = f'{pdb_name}_{ires}_{lig_poses[lig_pose][0]}_{idof}_{pose_num}'
               xtal_poses = mof.xtal_build.xtal_build(
                  pdb_name,
                  xspec,
                  aa1,
                  aa2,
                  rot_pose,
                  peptide_sym,
                  spec.pept_orig,
                  spec.pept_axis,
                  lig_sym,
                  metal_orig,
                  metal_sym_axis,
                  rpxbody,
                  tag,
               )

               if False and xtal_poses:
                  print('hoaktolfhtoia')
                  print(rot_pose)
                  print(pdb_name)
                  print(xspec)
                  rot_pose.dump_pdb('test_xtal_build_p213.pdb')
                  print(ires)
                  print(peptide_sym)
                  print(spec.pept_orig)
                  print(spec.pept_axis)
                  print(lig_sym)
                  print(metal_orig)
                  print(metal_sym_axis)
                  print('rp.Body(pose)')

                  xalign, xpose, bodypdb = xtal_poses[0]
                  print(xalign)
                  xpose.dump_pdb('xtal_pose.pdb')

                  assert 0

               for ixtal, (xalign, xtal_pose, body_pdb) in enumerate(xtal_poses):
                  celldim = xtal_pose.pdb_info().crystinfo().A()
                  fname = f"{xspec.spacegroup.replace('
','_')}_cell{int(celldim):03}_{tag}"
                  results.append(
                     mof.result.Result(
                        xspec,
                        fname,
                        xalign,
                        rpxbody,
```

```
                        xtal_pose,
                        body_pdb,
                    ))

            pose_num += 1

   return results

def hokey_position_atoms(pose, ires1, ires2, metal_pos, metalaxispos):
   znres1, znres2 = None, None
   znres1 = ires1
   znres2 = ires2
   znatom1 = '1HB'
   znatom2 = '1HB'
   axisatom1 = '2HB'
   axisatom2 = '2HB'
   for znres, znatom, axisatom in [(znres1, znatom1, axisatom1), (znres2, znatom2,
axisatom2)]:
      pose.set_xyz(AtomID(pose.residue(znres).atom_index(znatom), znres),
                   rVec(metal_pos[0], metal_pos[1], metal_pos[2]))
      pose.set_xyz(AtomID(pose.residue(znres).atom_index(axisatom), znres),
                   rVec(metalaxispos[0], metalaxispos[1], metalaxispos[2]))
```

Listing S6: Crystal parameters such as resolution of amino acid rotamer sampling and deviation from ideal metal coordination geometry are specified.

```
import sys, argparse, rpxdock as rp

def default_cli_parser(parent=None, **kw):
  parser = parent if parent else argparse.ArgumentParser(allow_abbrev=False)
  addarg = rp.app.options.add_argument_unless_exists(parser)


  addarg("inputs", nargs="*", type=str, default=[], help='input structures')

  addarg('--aa_labels', type=str, nargs='*',                    default='CYS DCYS ASP
DASP GLU DGLU HIS DHIS HISD DHISD'.split(),
         help='choices: CYS DCYS ASP DASP GLU DGLU HIS DHIS HISD DHISD',)
  addarg('--aa_pair_labels', type=str, nargs='*',               default=['ALL'],
help='give in pairs (--aa_pair_labels A B C D yields A-B and C-B pairs) choices: CYS
DCYS ASP DASP GLU DGLU HIS DHIS HISD DHISD')
  addarg('--angle_err_tolerance', type=float,                   default=15,
help='max allowed angular deviation from ideal metal binding. applied early, so ok
to be generous')
  addarg('--angle_to_cart_err_ratio', type=float,               default=20.0,
help='lever distance to equate angular and cartesian errors. probably no reason to
change, unless you know why')
  addarg('--chiresl_asp1', type=float,                          default=8.0,
help='resolution of scanning for asp chi1')
  addarg('--chiresl_asp2', type=float,                          default=5.0,
help='resolution of scanning for asp chi2')
  addarg('--chiresl_cys1', type=float,                          default=6.0,
help='resolution of scanning for cys chi1')
  addarg('--chiresl_cys2', type=float,                          default=8.0,
help='resolution of scanning for cys chi2')
  addarg('--chiresl_glu1', type=float,                          default=6.0,
help='resolution of scanning for glu chi1')
```

```
  addarg('--chiresl_glu2', type=float,                          default=12.0,
help='resolution of scanning for glu chi2')
  addarg('--chiresl_glu3', type=float,                          default=6.0,
help='resolution of scanning for glu chi3')
  addarg('--chiresl_his1', type=float,                          default=3.0,
help='resolution of scanning for his chi1')
  addarg('--chiresl_his2', type=float,                          default=8.0,
help='resolution of scanning for his chi2')
  addarg('--clash_dis', type=float,                             default=3.3,
help='distance below which atoms "clash"')
  addarg('--cluster', action="store_true",                    default=False,
help='')
  addarg('--contact_dis', type=float,                           default=7.0,
help='max CB-CB distance between residue "neighbors"')
  addarg('--cst_ang_metal', type=float,                         default=109.47,
help='desired angle between metal liganding atoms')
  addarg('--cst_dis_metal', type=float,                         default=2.2,
help='desired distance from metal to liganding atoms')

  addarg('--cst_sd_cut_ang', type=float,                        default=0.01,
help='std dev of angular cutpoint constraint (lower is stronger constraint)')
  addarg('--cst_sd_cut_dih', type=float,                        default=0.1,
help='std dev of dihedral cutpoint constraint (lower is stronger constraint)')
  addarg('--cst_sd_cut_dis', type=float,                        default=0.01,
help='std dev of distance cutpoint constraint (lower is stronger constraint)')

  addarg('--cst_sd_metal_coo', type=float,                      default=0.5,
help='std dev of metal-O-C-O dihedral constraint (lower is stronger constraint)')
  addarg('--cst_sd_metal_dir', type=float,                      default=0.4,
help='std dev of ligand "orbital points at metal" angle constraint (lower is
stronger constraint)')
  addarg('--cst_sd_metal_lig_ang', type=float,                  default=0.4,
help='std dev of lig-metal-lig angle constraint (lower is stronger constraint)')
  addarg('--cst_sd_metal_lig_dist', type=float,                 default=0.2,
help='std dev of metal-lig distance constraint (lower is stronger constraint)')
  addarg('--cst_sd_metal_olap', type=float,                     default=0.03,
help='std dev of distance between symmetric copies of metal constraint (lower is
stronger constraint)')

  addarg('--test_run', action="store_true",                        default=False,
help='ignores most flags and inputs, set to test values')
  addarg('--debug', action="store_true",                        default=False,
help='extra output and maybe modified behavior')
  addarg('--dist_err_tolerance', type=float,                    default=1.0,
help='max allowed deviation of symmetric metal overlap from 0. applied early, so ok
to be generous')
  addarg("--dont_replace_these_aas", nargs="*", type=str,      default=['PRO'],
help='AAs which should not be changed')
  addarg('--err_tolerance', type=float,                         default=2.0,
help='max allowed combination of dist_err and angle_err. applied early, so ok to be
generous')
  addarg('--max_2res_score', type=float,                        default=10.0,
help='max score delta upon placing rotamers')
  addarg('--max_bb_redundancy', type=float,                     default=0.1,
help='max non-aligned rms distance between any outputs')
  addarg('--max_cell_size', type=float,                         default=50,
help='maximum cell size (will be correlated to peptide size, maybe use
--max_solv_frac if you have mixed size peptides')
  addarg('--max_dun_score', type=float,                         default=6.0,
help='overall maximum DUN score ever allowed for consideration')
  addarg('--max_pept_size', type=int,                           default=10,
help='reserve output space for this many residues (keep same to match run results
```

```
together)')
  addarg('--max_score_minimized', type=float,                  default=50.0,
help='maximum score after minimization (including cst and all)')
  addarg('--max_solv_frac', type=float,                        default=0.8,
help='maximum solvent fraction accepted. this is highly approximate... recommend
spot-checking and adjusting as needed')
  addarg('--max_sym_score', type=float,                        default=100.0,
help='max nonbonded energy across xtal contacts (induding clash)')
  addarg('--maxdun_asp', type=float,                           default=5.0,
help='max dunbrak score for asp')
  addarg('--maxdun_cys', type=float,                           default=4.0,
help='max dunbrak score for cys')
  addarg('--maxdun_glu', type=float,                           default=5.0,
help='max dunbrak score for glu')
  addarg('--maxdun_his', type=float,                           default=5.0,
help='max dunbrak score for his')
  addarg('--min_cell_size', type=float,                        default=0,
help='minimum cell size')
  addarg('--min_contacts', type=float,                         default=0,
help='minimun number of res-res contacts across symmetric units')
  addarg('--min_dist_to_z_axis', type=float,                   default=5.0,
help='maybe dont change this')
  addarg('--output_prefix', type=str,
default='results/mofdock_',  help='prefix to output filenames')
  addarg('--overwrite', action="store_true",                   default=False,
help='')
  addarg('--rotcloud_cache', type=str,
default='.rotcloud_cache',   help='cache file location')
  addarg('--scale_number_of_rotamers', type=float,             default=1.0,
help='modify resolution of rotamers')
  addarg('--sfxn_minimize_weights', type=str,
default='minimize.wts',      help='minimization score func wts file')
  addarg('--sfxn_rotamer_weights', type=str,
default='rotamer.wts',       help='rotamer scanning score func wts file')
  addarg('--sfxn_sterics_weights', type=str,
default='sterics.wts',       help='clash checking score func wts file')
  addarg('--spacegroups', nargs='*', type=str,                 default=[],
help='list of spacegroups')
  addarg('--sym_axes_angle_tolerance', type=float,             default=5.0,
help='max deviation from crystal "magic angle"')
  addarg('--bb_break_dist', type=float,             default=3.0)

  addarg('--postprocess', action="store_true",                 default=False,
help='')
  addarg('--strip_rosetta_content_from_results', action="store_true",
default=False,               help='')


  parser.has_mof_args = True
  return parser

def get_cli_args(argv=None, parent=None, **kw):
  parser = default_cli_parser(parent, **kw)
  argv = sys.argv[1:] if argv is None else argv
  argv = rp.app.options.make_argv_with_atfiles(argv, **kw)
  options = parser.parse_args(argv)
  return rp.Bunch(options)

def defaults():
  return get_cli_args([])
```

Listing S7: Rosetta minimizes the resulting modeled lattices.

```
import os, mof, numpy as np, rpxdock as rp
from mof.pyrosetta_init import (rosetta as r, rts, makelattice, addcst_dis,
addcst_ang,
                                addcst_dih, name2aid, printscores)
from pyrosetta import AtomID
from pyrosetta.rosetta.numeric import xyzVector_double_t as xyzVec

def print_nonzero_energies(sfxn, pose):
   for st in sfxn.get_nonzero_weighted_scoretypes():
      print(st)

def minimize_mof_xtal(sfxn, xspec, pose, debug=False, **kw):
   kw = rp.Bunch(kw)

   nresasym = pose.size()
   beg = 1
   end = nresasym - 1
   metalres = rts.name_map('ZN')
   metalname = 'ZN'
   metalresnos = [nresasym, 2 * nresasym]  # TODO.. make not stupid
   metalnbonds = 4

   metalaid = AtomID(1, metalresnos[0])

   pose = pose.clone()
   r.core.pose.remove_lower_terminus_type_from_pose_residue(pose, beg)
   r.core.pose.remove_upper_terminus_type_from_pose_residue(pose, end)
   for ir in range(1, pose.size() + 1):
      if 'HIS' in pose.residue(ir).name():
         newname = pose.residue(ir).name().replace('HIS', 'HIS_D')
         newname = newname.replace('_D_D', '')
         r.core.pose.replace_pose_residue_copying_existing_coordinates(
            pose, ir, rts.name_map(newname))

   if False:
      tmp = pose.clone()
      r.core.pose.replace_pose_residue_copying_existing_coordinates(tmp,
metalresnos[0], metalres)
      makelattice(tmp)
      tmp.dump_pdb('before.pdb')
   makelattice(pose)
   if debug: print(f'minimize.py score initial...................
{sfxn(pose):10.3f}')

   syminfo = r.core.pose.symmetry.symmetry_info(pose)
   symdofs = syminfo.get_dofs()
   allowed_jumps = list()

   nxyz = pose.residue(beg).xyz('N')
   cxyz = pose.residue(end).xyz('C')
   nac, cac = None, None  # (N/C)-(a)djacent (c)hain
   for isub in range(1, syminfo.subunits()):
      othern = pose.residue((isub + 0) * nresasym + 1).xyz('N')
      otherc = pose.residue((isub + 1) * nresasym - 1).xyz('C')
      if nxyz.distance(otherc) < 2.0: cac = (isub + 1) * nresasym - 1
      if cxyz.distance(othern) < 2.0: nac = (isub + 0) * nresasym + 1
```

```
   assert nac and cac, 'backbone is weird?'
   if debug: print('peptide connection 1:', cac, beg)
   if debug: print('peptide_connection 2:', end, nac)


   f_metal_lig_dist = r.core.scoring.func.HarmonicFunc(kw.cst_dis_metal,
kw.cst_sd_metal_lig_dist)
   f_metal_lig_ang = r.core.scoring.func.HarmonicFunc(np.radians(kw.cst_ang_metal),
                                                      kw.cst_sd_metal_lig_ang)
   f_metal_olap = r.core.scoring.func.HarmonicFunc(0.0, kw.cst_sd_metal_olap)
   f_point_at_metal = r.core.scoring.func.HarmonicFunc(0.0, kw.cst_sd_metal_dir)
   f_metal_coo = r.core.scoring.func.CircularHarmonicFunc(0.0, kw.cst_sd_metal_coo)
   f_cut_dis = r.core.scoring.func.HarmonicFunc(1.328685, kw.cst_sd_cut_dis)
   f_cut_ang_cacn = r.core.scoring.func.HarmonicFunc(2.028, kw.cst_sd_cut_ang)
   f_cut_ang_cnca = r.core.scoring.func.HarmonicFunc(2.124, kw.cst_sd_cut_ang)
   f_cut_dih = r.core.scoring.func.CircularHarmonicFunc(np.pi, kw.cst_sd_cut_dih)
   f_cut_dih0 = r.core.scoring.func.CircularHarmonicFunc(0.00, kw.cst_sd_cut_dih)

   ################### check cutpoint ##################

   conf = pose.conformation().clone()
   assert r.core.conformation.symmetry.is_symmetric(conf)
   pi = pose.pdb_info()
   conf.declare_chemical_bond(cac, 'C', beg, 'N')
   pose.set_new_conformation(conf)
   pose.set_new_energies_object(r.core.scoring.symmetry.SymmetricEnergies())
   pose.pdb_info(pi)
   if debug: print(f'minimize.py: score after chem bonds.....
{sfxn(pose):10.3f}')

   ##############################################

   cst_cut, cst_lig_dis, cst_lig_ang, cst_lig_ori = list(), list(), list(), list()

   ############### chainbreaks ################

   cst_cut.append(addcst_dis(pose, cac, 'C ', beg, 'N', f_cut_dis))
   cst_cut.append(addcst_dis(pose, end, 'C ', nac, 'N', f_cut_dis))
   if debug: print(f'minimize.py: score after chainbreak dis......
{sfxn(pose):10.3f}')
   cst_cut.append(addcst_ang(pose, cac, 'CA', cac, 'C', beg, 'N ', f_cut_ang_cacn))
   cst_cut.append(addcst_ang(pose, cac, 'C ', beg, 'N', beg, 'CA', f_cut_ang_cnca))
   cst_cut.append(addcst_ang(pose, end, 'CA', end, 'C', nac, 'N ', f_cut_ang_cacn))
   cst_cut.append(addcst_ang(pose, end, 'C ', nac, 'N', nac, 'CA', f_cut_ang_cnca))
   if debug: print(f'minimize.py: score after chainbreak ang......
{sfxn(pose):10.3f}')

   cst_cut.append(addcst_dih(pose, cac, 'CA', cac, 'C', beg, 'N ', beg, 'CA',
f_cut_dih))
   cst_cut.append(addcst_dih(pose, end, 'CA', end, 'C', nac, 'N ', nac, 'CA',
f_cut_dih))
   cst_cut.append(addcst_dih(pose, cac, 'O ', cac, 'C', beg, 'N ', beg, 'CA',
f_cut_dih0))
   cst_cut.append(addcst_dih(pose, end, 'O ', end, 'C', nac, 'N ', nac, 'CA',
f_cut_dih0))
   if debug: print(f'minimize.py: score after chainbreak dihedral.
{sfxn(pose):10.3f}')

   ############## metal constraints ################

   for i, j in [(i, j) for i in metalresnos for j in metalresnos if i < j]:
      addcst_dis(pose, i, metalname, j, metalname, f_metal_olap)
```

```
   if debug:
      print(f'minimize.py: score after metal olap ......... {sfxn(pose):10.3f}')

   allowed_elems = 'NOS'
   znpos = pose.residue(metalresnos[0]).xyz(1)
   znbonded = list()
   for ir in range(1, len(pose.residues) + 1):
      res = pose.residue(ir)
      if not res.is_protein(): continue
      for ia in range(5, res.nheavyatoms() + 1):
         aid = AtomID(ia, ir)
         elem = res.atom_name(ia).strip()[0]
         if elem in allowed_elems:
            xyz = pose.xyz(aid)
            dist = xyz.distance(znpos)
            if dist < 3.5:
               if res.atom_name(ia) in (' OD2', ' OE2'):  # other COO O sometimes
closeish
                  continue
               znbonded.append(aid)
   if len(znbonded) != metalnbonds:
      print('WRONG NO OF LIGANDING ATOMS', len(znbonded))
      if debug:
         for aid in znbonded:
            print(pose.residue(aid.rsd()).name(),
pose.residue(aid.rsd()).atom_name(aid.atomno()))
            pose.dump_pdb('WRONG_NO_OF_LIGANDING_ATOMS.pdb')
            return None, None

   for i, aid in enumerate(znbonded):
      cst = r.core.scoring.constraints.AtomPairConstraint(metalaid, aid,
f_metal_lig_dist)
      cst_lig_dis.append(cst)
      pose.add_constraint(cst)

   if debug: print(f'minimize.py: score after metal dist .........
{sfxn(pose):10.3f}')

   for i, aid in enumerate(znbonded):
      ir, res = aid.rsd(), pose.residue(aid.rsd())
      if all(_ not in res.name() for _ in 'ASP CYS HIS GLU'.split()):
         assert 0, f'unrecognized res {res.name()}'
      if any(_ in res.name() for _ in 'ASP GLU'.split()):
         # metal comes off of OD1/OE1
         ir, coo = aid.rsd(), ('OD1 CG OD2' if 'ASP' in res.name() else 'OE1 CD
OE2').split()
         cst_lig_ori.append(
            addcst_dih(pose, ir, coo[0], ir, coo[1], ir, coo[2], metalaid.rsd(),
metalname,
                       f_metal_coo))
      else:
         if 'HIS' in res.name(): aname = 'HD1' if res.has('HD1') else 'HE2'
         if 'CYS' in res.name(): aname = 'HG'
         cst_lig_ori.append(
            addcst_ang(pose, ir, res.atom_name(aid.atomno()), metalaid.rsd(),
metalname, ir,
                       aname, f_point_at_metal))

   if debug: print(f'minimize.py: score after metal dir...........
{sfxn(pose):10.3f}')

   for i, iaid in enumerate(znbonded):
```

```
      for j, jaid in enumerate(znbonded[:i]):
         cst = r.core.scoring.constraints.AngleConstraint(iaid, metalaid, jaid,
f_metal_lig_ang)
         cst_lig_ang.append(cst)
         pose.add_constraint(cst)

   if debug: print(f'minimize.py: score after lig angle added.....
{sfxn(pose):10.3f}')

   ################ minimization ##########################

   movemap = r.core.kinematics.MoveMap()
   movemap.set_bb(True)
   movemap.set_chi(True)
   movemap.set_jump(False)
   for i in allowed_jumps:
      movemap.set_jump(True, i)
   minimizer = r.protocols.minimization_packing.symmetry.SymMinMover(
      movemap, sfxn, 'lbfgs_armijo_nonmonotone', 0.01, True)  # tol, nblist
   if sfxn.has_nonzero_weight(r.core.scoring.ScoreType.cart_bonded):
      minimizer.cartesian(True)
   minimizer.apply(pose)
   if debug: print(f'minimize.py: score after min no scale........
{sfxn(pose):10.3f}')


   kw.timer.checkpoint(f'min scale 1.0')

   asym = r.core.pose.Pose()
   r.core.pose.symmetry.extract_asymmetric_unit(pose, asym, False)
   r.core.pose.replace_pose_residue_copying_existing_coordinates(asym,
metalresnos[0], metalres)

   if debug: print(kw.timer)

   info = rp.Bunch()
   info.score = sfxn(pose)

   ############### score component stuff ################
   st = r.core.scoring.ScoreType
   etot = pose.energies().total_energies()
   info.score_fa_atr = (etot[st.fa_atr])
   info.score_fa_rep = (etot[st.fa_rep])
   info.score_fa_sol = (etot[st.fa_sol])
   info.score_lk_ball = (etot[st.lk_ball] + etot[st.lk_ball_iso] +
etot[st.lk_ball_bridge] +
                         etot[st.lk_ball_bridge_uncpl])
   info.score_fa_elec = (etot[st.fa_elec] + etot[st.fa_intra_elec])
   info.score_hbond_sr_bb = (etot[st.hbond_sr_bb] + etot[st.hbond_lr_bb] +
etot[st.hbond_bb_sc] +
                             etot[st.hbond_sc])
   info.score_dslf_fa13 = (etot[st.dslf_fa13])
   info.score_atom_pair_constraint = (etot[st.atom_pair_constraint])
   info.score_angle_constraint = (etot[st.angle_constraint])
   info.score_dihedral_constraint = (etot[st.dihedral_constraint])
   info.score_omega = (etot[st.omega])
   info.score_rotamer = (etot[st.fa_dun] + etot[st.fa_dun_dev] + etot[st.fa_dun_rot]
+
                         etot[st.fa_dun_semi] + etot[st.fa_intra_elec] +
etot[st.fa_intra_rep] +
                         etot[st.fa_intra_atr_xover4] + etot[st.fa_intra_rep_xover4]
+
```

```
                        etot[st.fa_intra_sol_xover4])
   info.score_ref = (etot[st.ref])
   info.score_rama_prepro = (etot[st.rama_prepro])
   info.score_cart_bonded = (etot[st.cart_bonded])
   info.score_gen_bonded = (etot[st.gen_bonded])

   pose.remove_constraints()
   info.score_wo_cst = sfxn(pose)

   [pose.add_constraint(cst) for cst in cst_cut]
   info.score_cst_cut = sfxn(pose) - info.score_wo_cst
   pose.remove_constraints()

   [pose.add_constraint(cst) for cst in cst_lig_dis]
   [pose.add_constraint(cst) for cst in cst_lig_ang]
   [pose.add_constraint(cst) for cst in cst_lig_ori]
   info.score_cst_lig_ori = sfxn(pose) - info.score_wo_cst
   pose.remove_constraints()

   [pose.add_constraint(cst) for cst in cst_lig_dis]
   info.score_cst_lig_dis = sfxn(pose) - info.score_wo_cst
   pose.remove_constraints()

   [pose.add_constraint(cst) for cst in cst_lig_ang]
   info.score_cst_lig_ang = sfxn(pose) - info.score_wo_cst
   pose.remove_constraints()

   [pose.add_constraint(cst) for cst in cst_lig_ori]
   info.score_cst_lig_ori = sfxn(pose) - info.score_wo_cst
   pose.remove_constraints()

   return asym, info
```

Designing non-coordinating residues: Once the crystals are modeled and filtered for clashing, amino acids not involved in the metal coordination are designed using Rosetta. This optimizes the sequence for the best packing in the context of the crystal lattice while constraining the metal coordinating residues.

Listing S8: The following xml protocol is used to design macrocycles in the context of the 3D lattices.

```
<ROSETTASCRIPTS>
    <SCOREFXNS>
        <ScoreFunction name="beta" weights="ref2015" symmetric="1" />

        <ScoreFunction name="beta_ncs" weights="ref2015" symmetric="1" >
            <Reweight scoretype="dihedral_constraint" weight="1.0" />
        </ScoreFunction>

        <ScoreFunction name="beta_soft" weights="ref2015" symmetric="1" >
            <Reweight scoretype="fa_rep" weight="0.05" />
        </ScoreFunction>

        <ScoreFunction name="beta_cst_highhbond" weights="ref2015_cst.wts"
symmetric="true" >
           <Reweight scoretype="chainbreak" weight="25.0" />
           <Reweight scoretype="hbond_sr_bb" weight="10.0" />
```

```
            <Reweight scoretype="hbond_lr_bb" weight="10.0" />
        </ScoreFunction>

        <ScoreFunction name="beta_cst_highhbond_comp" weights="ref2015_cst.wts"
symmetric="true" >
            <Reweight scoretype="chainbreak" weight="25.0" />
            <Reweight scoretype="hbond_sr_bb" weight="10.0" />
            <Reweight scoretype="hbond_lr_bb" weight="10.0" />
            <Reweight scoretype="aa_composition" weight="1.0" />
            <Reweight scoretype="aspartimide_penalty" weight="1.0" />
        </ScoreFunction>
    </SCOREFXNS>

    <RESIDUE_SELECTORS>
        <ResidueName name="metal_binding"
residue_name3="ASP,GLU,HIS,CYS,DAS,DGU,DHI,DCS" />
        <Not name="designable" selector="metal_binding" />
        <Phi name="pos_phi" select_positive_phi="true"
ignore_unconnected_upper="false"/>
        <Not name="neg_phi" selector="pos_phi" />
        <And name="designable_posphi" selectors="designable,pos_phi" />
        <And name="designable_negphi" selectors="designable,neg_phi" />
        <Index name="asym_unit" resnums="1-3" />
        <Index name="select_first_res" resnums="1" />
        <Index name="select_last_res" resnums="3" />
        <And name="designable_asym_unit" selectors="designable,asym_unit" />
    </RESIDUE_SELECTORS>

    <PACKER_PALETTES>
        <CustomBaseTypePackerPalette name="palette"
additional_residue_types="AIB,DALA,DPHE,DILE,DLYS,DLEU,DMET,DASN,DPRO,DGLN,DARG,DSER
,DTHR,DVAL,DTRP,DTYR"/>
    </PACKER_PALETTES>

    <TASKOPERATIONS>

        <ExtraRotamersGeneric name="extrarot" ex1="true" ex2="true" ex3="false"
ex4="false" extrachi_cutoff="3" />
        <ReadResfile name="laa" filename="inputs/laa.resfile"
selector="designable_negphi" />
        <ReadResfile name="daa" filename="inputs/daa.resfile"
selector="designable_posphi" />
        <OperateOnResidueSubset name="no_design_metal_binding_positions"
selector="metal_binding" >
            <PreventRepackingRLT/>
        </OperateOnResidueSubset>
    </TASKOPERATIONS>

    <FILTERS>
        <ShapeComplementarity name="sc_filt" jump="1" verbose="1" min_sc="0.2"
write_int_area="1" confidence="0"/>
        <ScoreType name="fa_rep" score_type="fa_rep" threshold="4000"
scorefxn="beta" confidence="1" />
        <ScoreType name="fa_rep_reporter" score_type="fa_rep" threshold="2000"
scorefxn="beta" confidence="0" />
        <ScoreType name="total_score_reporter" score_type="total_score"
threshold="2000" scorefxn="beta" confidence="0" />

        <OversaturatedHbondAcceptorFilter name="oversat"
max_allowed_oversaturated="0" hbond_energy_cutoff="-0.1"
consider_mainchain_only="true" scorefxn="beta" />
    </FILTERS>
```

```
    <MOVERS>
        <MakeLatticeMover name="make_lattice" contact_dist="20" />
        TaskAwareSymMinMover name="min" scorefxn="beta" bb="0" chi="1" rb="1"
symdofs="A,B" task_operations="design_task" />

        <ModifyVariantType name="upper_cutpoints" add_type="CUTPOINT_UPPER"
residue_selector="select_first_res" />
       <ModifyVariantType name="lower_cutpoints" add_type="CUTPOINT_LOWER"
residue_selector="select_last_res" />

        <DeclareBond name="bond1" res1="9" res2="1" atom1="C" atom2="N"
add_termini="false" />

       <Small name="small_perturbation" angle_max="5.0" scorefxn="beta" />
       <FastRelax name="frlx1" repeats="1" scorefxn="beta_cst_highhbond"
min_type="dfpmin">
           <MoveMap name="frlx_mm1" >
              <Span begin="1" end="999" bb="false" chi="false" />
           </MoveMap>
       </FastRelax>
       <FastRelax name="frlx2" repeats="1" scorefxn="beta_cst_highhbond"
min_type="dfpmin">
           <MoveMap name="frlx_mm2" >
              <Span begin="1" end="999" bb="true" chi="true" />
           </MoveMap>
       </FastRelax>
       <AddCompositionConstraintMover name="aacomp_all" filename="inputs/all.comp"
selector="designable_asym_unit" />

       <PackRotamersMover name="fdes1" scorefxn="beta_cst_highhbond_comp"
task_operations="laa,daa,no_design_metal_binding_positions" packer_palette="palette"
/>
        <FastRelax name="frlx3" repeats="1" scorefxn="beta" min_type="dfpmin"
task_operations="extrarot" >
           <MoveMap name="frlx_mm3" >
              <Span begin="1" end="999" bb="true" chi="true" />
           </MoveMap>
        </FastRelax>


    </MOVERS>
    <APPLY_TO_POSE>
    </APPLY_TO_POSE>
    <PROTOCOLS>
        <Add mover="make_lattice" />
        <Add filter="sc_filt" />
        <Add filter="fa_rep" />
        <Add filter="fa_rep_reporter" />
        <Add filter="total_score_reporter" />
        <Add mover="small_perturbation" />
       <Add mover="aacomp_all" />
       <Add mover="fdes1" />
        <Add filter="sc_filt" />
        <Add filter="fa_rep" />
    </PROTOCOLS>
</ROSETTASCRIPTS>
```

Listing S9: The following flags file specifies the options used while designing the lattices.

```
-l inputs/all.list
-nstruct 1
-in:file:extra_res_fa HZD.params
-parser:protocol inputs/working_peptide_xtal_design.xml
-crystal_refine
-ex1
-ex2
-out:file:scorefile score.sc
-matdes::num_subs_building_block 2
-no_his_his_pairE
-per_chain_renumbering
-out:path:all output/
-detect_disulf false
-symmetry_definition CRYST1
-overwrite
-out:path:pdb outputs
```

Filtering modeled lattices: Modeled crystal lattices were filtered based on density and energy calculations. The density of each crystal model was calculated using the following script. Here the peptides are split into asymmetric units (i.e. a C3 peptide has three repeating asymmetric units). The mass of the asymmetric unit is calculated and multiplied by the number of asymmetric units in a unit cell. Then, the volume of the unit cell is also calculated and density is determined by dividing the mass by the volume and saved in a list.

Listing S10: Density calculations for designed crystal lattices.

```
## This script takes a list of pdb names without a path or a '.pdb' at the end.
## The mass of an asymmetric unit (aka one chain) is calculated.
## The unit cell volume is calculated based on CRYST1 line


from pyrosetta import *
import re
import argparse
import glob
import sys
import math
from pathlib import Path
from astropy.table import Table
from astropy.io import ascii

#takes a list of pdb names
def parse_arguments(argv):
    parser = argparse.ArgumentParser(description="takes list of peptide names")
    parser.add_argument('PEP_LIST', nargs = 1, type = str, help = 'List of pdb
names')
    args = parser.parse_args()
    return args


#calculates the cubic volume of the unit cell. Takes in the file path to a pdb
def unit_cell_volume (pdb_file_path):
    file_path = Path(pdb_file_path)
    with open (file_path) as pdb:
        #read Cryst1 line
```

```
        cryst1 = []
        for line in pdb:
            if line.startswith("CRYST1"):
                cryst1 = (line).split()
        print("CRYST! LINE IS: ", cryst1)
        #extract x,y,z, and angles from cryst1 based on digit patterns
        x_y_z = [float(i) for i in cryst1[1:7]]

        #determine crystal system
        if x_y_z[0] == x_y_z[1] == x_y_z[2] and x_y_z[3] == x_y_z[4] == x_y_z[5] ==
90.00:
            crystal_system = "cubic"
        elif x_y_z[0] == x_y_z[1] != x_y_z[2] and x_y_z[3] == x_y_z[4] == 90.00 and
x_y_z[5] == 120.00:
            crystal_system = "hexagonal"
        elif x_y_z[0] == x_y_z[1] != x_y_z[2] and x_y_z[3] == x_y_z[4] == x_y_z[5]
== 90.00:
            crystal_system = "tetragonal"
        elif x_y_z[0] == x_y_z[1] == x_y_z[2] and x_y_z[3] == x_y_z[4] == x_y_z[5]
!= 90.00:
            crystal_system = "rhombohedral"
        elif x_y_z[0] != x_y_z[1] != x_y_z[2] and x_y_z[3] == x_y_z[4] == x_y_z[5]
== 90.00:
            crystal_system = "orthorhombic"
        elif x_y_z[0] != x_y_z[2] and x_y_z[3] == x_y_z[4] == 90.00 and x_y_z[5] !=
90.00:
            crystal_system = "monoclinic"
        elif x_y_z[0] != x_y_z[1] != x_y_z[2] and x_y_z[3] != x_y_z[4] != x_y_z[5]:
            crystal_system = "triclinic"

        #determine lattice centering and number of asu in a unit cell
        if cryst1[7] == "P":
            lattice_centering = "primitive"
            N_asu = "1"
        elif cryst1[7] == "I":
            lattice_centering = "body centered"
            N_asu = "2"
        elif cryst1[7] == "F":
            lattice_centering = "face centered"
            N_asu = "4"
        elif cryst1[7] == "A" or cryst1[7] == "B" or cryst1[7]== "C":
            lattice_centering = "base centered"
            N_asu = "2"
        elif cryst1[7] == "R":
            lattice_centering = "D centered"
            N_asu = "1"

        #calculate cubic volume of unit cell
        if crystal_system == "cubic":
            volume = x_y_z[0] * x_y_z[1] * x_y_z[2]
            return volume
        elif crystal_system == "hexagonal":
            volume = 0.866 * x_y_z[0] * x_y_z[0] * x_y_z[2]
            return volume
        elif crystal_system == "tetragonal":
            volume = x_y_z[0] * x_y_z[0] * x_y_z[2]
            return volume
        elif crystal_system == "rhombohedral":
            volume = [sqrt(1- 3 * (math.cos(x_y_z[3]))**2 + 2 *
(math.cos(x_y_z[3]))**3)] * (x_y_z[0] ** 3)
            return volume
        elif crystal_system == "orthorhombic":
```

```
            volume = x_y_z[0] * x_y_z[1] * x_y_z[2]
            return volume
        elif crystal_system == "monoclinic":
            volume = x_y_z[0] * x_y_z[1] * x_y_z[2] * math.sin(x_y_z[4])
            return volume
        elif crystal_system == "triclinic":
            volume = x_y_z[0] * x_y_z[1] * x_y_z[2]


        return volume

#calculates the mass of one chain. Takes in the file path to a pdb and the length of
the asymmetric unit.
def get_mass_asym_unit (pdb_file_path):
    file_path = Path(pdb_file_path)

    with open (file_path) as pdb:

        #read Cryst1 line
        cryst1=[]
        for line in pdb:
            if line.startswith("CRYST1"):
                cryst1 = (line).split()

        #extract x,y,z, and angles from cryst1 based on digit patterns
        x_y_z = cryst1[1]

    #use ref 2015 score function
    sfxn = get_score_function()

    #creates pose from one chain as the asymmetric unit
    pose = pose_from_pdb(pdb_file_path)


    #make a list of atom types in a pose and count the number of atoms in that list.
    Atoms = []
    for res in pose:
        for i in range(1, len(res.atoms())+1):
            Atoms.append(res.atom_name(i))


    #calculate mass of asym unit pose in grams/pose not moles
    mass = pyrosetta.rosetta.core.pose.mass(1,3,pose) / (6.022*10**23)

    #returns the mass and energy of the unit cell
    return [mass, sfxn(pose)]




if __name__ == '__main__':
    pyrosetta.init('-crystal_refine -in:file:extra_res_fa
/projects/peptides/mofs/params_files/HDZ.params' )

    args = parse_arguments(sys.argv)

    #takes in a list of pdb names. ex tmp.list
    input_file = args.PEP_LIST[0]

    data_rows = []

    with open (input_file) as f:
```

```
        for lines in f:

            designed_name = 'input/' + lines.rstrip()
            print('THIS IS THE PATH!!!!!' + designed_name)

            vol = (unit_cell_volume(designed_name))


 ###################REMEMBER TO CHANGE BASED ON SPACE GROUP#######################
 ####density of P23 space group using multi = 12  ################################
 ################################################################################
            density = round((get_mass_asym_unit(designed_name)[0]*12)/(vol*10**-24),
6)

            data_rows.append((lines.rstrip(), vol,  density) )

    #saves data in a table format
    data = Table(rows=data_rows, names= ('PDB', 'UNIT_CELL_VOLUME',
'Density_g/cm^3'))

    #export data as energy_data.list so it can be sorted and filtered later.
    ascii.write(data, 'energy_data_g_cm.list', overwrite=True, format='fixed_width'
)
```

Listing S11: The lattice energy landscape is estimated by draping the peptide sequence on all modeled lattices, then scoring the results using Rosetta's energy function.

```
import pyrosetta
from pyrosetta import rosetta

import numpy as np

pyrosetta.init('-extra_res_fa /home/csykang/scripts/Params/HZD.params
-in:file:fullatom -symmetry_definition CRYST1')
path_to_scaffolds =
str("/projects/peptides/mofs/mof_design/xtal_matches_WS/P4332_c3_9res_20200608/")

# This script will grab the sequences from the designed pdbs and determine the
extent to which a given sequence is favored by a given peptide xtal lattice
# This will be done by draping each sequence and circular permutations of the
sequence over a series of lattices, then scoring each draped sequence to see which
peptide + scaffold combinations work the best.

_DEBUG = False


#takes a list of pdb names
def read_in_pdbs():
    # Reads a pdbs.list file to get the paths to all of the pdbs I want to test,
    # then opens the pdbs with import_pose.pose_from_file
    # Returns a list, raw_pose_list, that contains all of the poses
    raw_pose_path_list = []
    raw_pose_list = []
    pdbslist = open("pdbs.list", "r")
    lines = pdbslist.read().splitlines()
```

```
    for line in lines:
        raw_pose_path_list.append(line)
    for path in raw_pose_path_list:
        raw_pose = rosetta.core.import_pose.pose_from_file(path)
        raw_pose_list.append(raw_pose)
    return raw_pose_list

def cyc_align(pose):
    # Takes in a pose, cyclizes it, then aligns it to the z-axis
    # Returns a pose
    pcm = rosetta.protocols.cyclic_peptide.PeptideCyclizeMover()
    ata=rosetta.protocols.cyclic_peptide.SymmetricCycpepAlign()
    pcm.apply(pose)
    ata.apply(pose)
    return pose

def mutate_residues(t,r,p):
    mut = rosetta.protocols.simple_moves.MutateResidue()
    mut.set_target(t)
    mut.set_res_name(r)
    mut.set_preserve_atom_coords(False)
    mut.apply(p)
    return p

def fix_HZD_termini(s):
    substring = ":"
    if s == "HZD":
        return "HIS"
    elif substring in s:
        return s.split(':')[0]
    else:
        return s



def output_data(seq_pdb, scaff_score_dictionary):
    seq_name = seq_pdb.split(".pdb")[0]

    file_name = 'drape_seq_summaries/sequence_{}_scores.txt'.format(seq_name)
    o = open(file_name, "w")
    o.write("DESIGNED SEQ PDB NAME: %s\n" % seq_pdb)

    for scaff in scaff_score_dictionary:
        o.write("SCAFFOLD: %s\t\t" % scaff)
        o.write("SEQ_PERMUTATION, SCORE: %s\n" % scaff_score_dictionary[scaff])

def minimize(p,sfxn):
    movemap = rosetta.core.kinematics.MoveMap()
    movemap.set_chi(True)
    movemap.set_bb(False)
    movemap.set_jump(False)
    minmover = pyrosetta.rosetta.protocols.minimization_packing.MinMover(movemap,
sfxn, 'linmin', 0.001, True)
    for i in range(5):
        minmover.apply(p)

def cyclic_perm(list_of_seq):
    n = len(list_of_seq)
    result = []
    for j in range(n):
        def f(l, k=j):
            return list(map(lambda i: l[i - k], range(n)))
```

```
        result.append(f)
    return result


def check_and_remove_variant(input_pose):
    ''' util function to remove the terminal variant
        in order to correctly calculate the distances between fragment pairs
        and find the correct atom (for example, "N", "C") to declare bond and
determine connecting direction
    '''
    for in_idx in range(1, input_pose.size()+1):
        upper =
input_pose.residue(in_idx).has_variant_type(pyrosetta.rosetta.core.chemical.VariantT
ype.UPPER_TERMINUS_VARIANT)
        if upper:
            pyrosetta.rosetta.core.pose.remove_variant_type_from_pose_residue(
input_pose, pyrosetta.rosetta.core.chemical.UPPER_TERMINUS_VARIANT, in_idx)

        lower =
input_pose.residue(in_idx).has_variant_type(pyrosetta.rosetta.core.chemical.VariantT
ype.LOWER_TERMINUS_VARIANT)
        if lower:
            pyrosetta.rosetta.core.pose.remove_variant_type_from_pose_residue(
input_pose, pyrosetta.rosetta.core.chemical.LOWER_TERMINUS_VARIANT, in_idx)

def create_xyz_array(input_pose):
    ''' go through all the heavyatoms of the pose
        extract the xyz coordinate, and put it into a hugh xyz numpy array
        and use a dictionary to keep track the residue and atom index for each numpy
array entry
    '''
    xyz_list = []
    idx_info_dict = {}
    count = 0
    for res_i in range(1, input_pose.size()+1):
        this_res = input_pose.residue(res_i)
        if this_res.is_virtual_residue(): continue
        this_aa3  = this_res.name3().strip()
        if this_aa3 == "ZN":continue
        for atom_i in range(1, 5):
            this_atom_name = this_res.atom_name(atom_i).strip()
            this_xyz = this_res.atom(this_atom_name).xyz()
            this_array = [this_xyz.x, this_xyz.y, this_xyz.z]
            xyz_list.append(this_array)
            idx_info_dict[count] = [res_i, atom_i, this_atom_name]
            count += 1
    return np.array(xyz_list), idx_info_dict

def connect_downstream_frag(input_lattice, ref_frag_key = 1, dist_cutoff = 2.0):
    from scipy.spatial.distance import cdist
    ''' This function will find the downstream fragment of a specified fragment in a
lattice
        And declare a bond between the fragment and the found downstream one.
        The purpose of the bond declaration is for correctly energy calculation
    '''
    ## ------- step1 extract all the fragments, and clean up by removing all the
terminal variants --------- ##
    fragments = input_lattice.split_by_chain()
    for frag_key, each_frag in enumerate(fragments, start=1):
        check_and_remove_variant(each_frag)

    ## go through fragments, create a dictionary storing the fragment information
```

```
    ## fragment key is actually also the chain index of the lattice, 1-based index
    frag_dict = {}
    for frag_key, each_frag in enumerate(fragments, start=1):
        xyz_array, idx_info_dict = create_xyz_array(each_frag)
        frag_dict[frag_key] = (each_frag, xyz_array, idx_info_dict)

    ## go through fragment,
    ## and find the fragment corresponding to the specified fragment (reference
fragment)
    ## and find its neighbor fragments
    ## it only need to connect the downstream one, due to the symmetry
    connections = {}
    for frag_key, frag_value in frag_dict.items():
        ## skip the specified fragment
        if frag_key == ref_frag_key: continue

        ## collect the query and ref information
        query_xyz       = frag_value[1]
        query_info_dict = frag_value[2]

        ref_xyz       = frag_dict[ref_frag_key][1]
        ref_info_dict = frag_dict[ref_frag_key][2]

        if query_xyz.size > 0:
            pair_dist = cdist(query_xyz, ref_xyz)
            query_indices, ref_indices = np.where(pair_dist < dist_cutoff)
            for query_idx, ref_idx in zip(query_indices, ref_indices):
                query_atom = query_info_dict[query_idx][2]
                ref_atom   =   ref_info_dict[  ref_idx][2]

                query_resnum = query_info_dict[query_idx][0]
                ref_resnum   =   ref_info_dict[  ref_idx][0]


                if f"{ref_atom}{query_atom}" == "CN":
                    if ref_frag_key == 1:
                        res1 = ref_resnum
                    else:
                        res1 = input_lattice.chain_end(ref_frag_key-1) + ref_resnum
                    atom1 = ref_atom
                    if frag_key == 1:
                        res2 = query_resnum
                    else:
                        res2 = input_lattice.chain_end(frag_key-1) + query_resnum
                    atom2 = query_atom
                    connections["down"] = ((ref_frag_key, frag_key), (res1, atom1,
res2, atom2))


    declare_bond = pyrosetta.rosetta.protocols.cyclic_peptide.DeclareBond()
    print(f'Declaring bond of the specified fragment {connections["down"][0][0]} to
the downstream fragment: {connections["down"][0][1]}')
    declare_bond.set(connections["down"][1][0], connections["down"][1][1],
connections["down"][1][2], connections["down"][1][3], add_termini = False)
    declare_bond.apply(input_lattice)
    return None

def connect_downstream_frag_simplified(input_lattice, ref_frag_key = 1, dist_cutoff
= 2.0):
    from scipy.spatial.distance import cdist
    ''' This function will find the downstream fragment of a specified fragment in a
lattice
```

```
        And declare a bond between this fragment and the found downstream one.
        The purpose of the bond declaration is for correctly energy calculation
    '''
    ## ------- step1 extract all the fragments --------- ##
    fragments = input_lattice.split_by_chain()

    ref_frag  = fragments[ref_frag_key]
    ref_indices = [x for x in range(1, ref_frag.size()+1) if
ref_frag.residue(x).is_protein()]
    if not ref_indices:
        print("please choose another ref fragment")
        return None
    ref_frag.sequence()
    ref_C_res = ref_frag.residue(ref_indices[-1])
    ref_C_xyz = ref_C_res.atom("C").xyz()
    ## go through fragments, create a list storing the connections between N and C
    connections = []
    for frag_key, each_frag in enumerate(fragments, start=1):
        if frag_key == ref_frag_key: continue
        frag_indices = [x for x in range(1, each_frag.size()+1) if
each_frag.residue(x).is_protein()]
        if not frag_indices: continue
        N_res = each_frag.residue(frag_indices[0])
        N_xyz = N_res.atom("N").xyz()
        CN_dist = np.linalg.norm(ref_C_xyz - N_xyz)
        if CN_dist < dist_cutoff:
            if ref_frag_key == 1:
                res1 = ref_indices[-1]
            else:
                res1 = input_lattice.chain_end(ref_frag_key-1) + ref_indices[-1]
            atom1 = "C"
            res2 = input_lattice.chain_end(frag_key-1) + frag_indices[0]
            atom2 = "N"
            connections.append( ((ref_frag_key, frag_key), (res1, atom1, res2,
atom2)) )

    declare_bond = pyrosetta.rosetta.protocols.cyclic_peptide.DeclareBond()
    for keys, connection in connections:
        print(f'Declaring bond of the specified fragment {keys[0]} to the downstream
fragment: {keys[1]}')
        declare_bond.set(connection[0], connection[1], connection[2], connection[3],
add_termini = False)
        declare_bond.apply(input_lattice)
    return None
##-------------------------------------------------------------------------------
---------##

# Read in designed pdbs
designed_pdbs = read_in_pdbs() # imports poses - requires a "pdbs.list" file
containing paths
sfxn = rosetta.core.scoring.ScoreFunctionFactory.create_score_function('ref2015')

# Make xml objs for specific Movers
xml = rosetta.protocols.rosetta_scripts.XmlObjects.create_from_string("""
<RESIDUE_SELECTORS>
    <Index name="peptide" resnums="1-9" />
</RESIDUE_SELECTORS>
<MOVERS>
    <MakeLatticeMover name="mlm" contact_dist="20" />
</MOVERS>
""")
mlm = xml.get_mover("mlm")
```

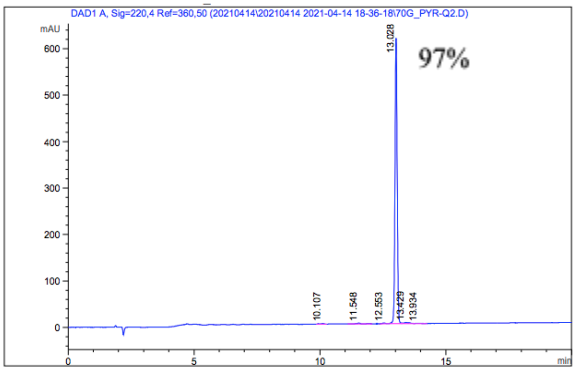
```
# Put designed sequences into a dictionary
dictionary_designed = {}
for pdb in designed_pdbs:
    pdb_name = pdb.pdb_info().name().split('/')[-1] # gets the pdb name for outputs
later
    res_index = list(range(1,rosetta.core.pose.chain_end_res(pdb,1)+1))

    seq_list = []
    for res in range(1, rosetta.core.pose.chain_end_res(pdb, 1)+1):
        seq_list.append(fix_HZD_termini(pdb.residue(res).name()))

    print("SEQ_LIST:", seq_list)

    seq_nozn_list = [x for x in seq_list if x != "ZN"]

    print("SEQ_LIST_NOZN:", seq_nozn_list)

    cp_seq_list = []
    for seq in cyclic_perm(seq_nozn_list): # cyclic permutations
        cp_seq_list.append(seq(seq_nozn_list))
        print(seq)

    dictionary_designed.update(dict({pdb_name : cp_seq_list}))

# depending on what you want, this could be either all of the scaffolds, or just the
scaffolds that are comparable (same metal binding ligands)
ext = str(".pdb")
dictionary_scaff = {}

for key in dictionary_designed.keys():
    list_of_tuples = []
    k = key.split('_0001.pdb')[0] # Needs to be checked to make sure the names still
work
    k_n = "".join((k,ext))
    s_l = "".join((path_to_scaffolds,k_n))
    scaff = rosetta.core.import_pose.pose_from_file(str(s_l))

    for resnum in range(1, int(rosetta.core.pose.chain_end_res(scaff,1))):
        print(resnum, scaff)
        mutate_residues(resnum,'GLY',scaff)

    mlm.apply(scaff) # apply MakeLatticeMover
    # This would be the best place to declare bonds on the poly gly lattices so that
they're fixed before being put into my dictionary of scaffolds, dictionary_scaff.

    connect_downstream_frag_simplified(scaff, ref_frag_key = 1, dist_cutoff = 2.0)

    polygly_score = round(sfxn(scaff), 2)

    list_of_tuples.append((scaff, polygly_score))
    dictionary_scaff.update(dict({k_n : list_of_tuples})) # key = name of the
scaffold, value = list of tuples containing (poly gly lattice pose (called "scaff")
and the score of "scaff")

    # Mutate each scaffold to the sequence saved in the dictionary
dictionary_scores = {}
for pep_seq in dictionary_designed:
    dictionary_seq_scaff = {}
    for scaff, info in dictionary_scaff.items():
        for scaff_pdb in info:
            permutation_counter = 1
```

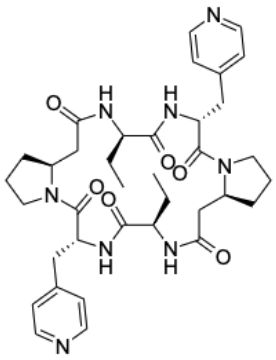
```
            list_of_tuples = []
            for seq_permutation in dictionary_designed[pep_seq]:
                for resnum in range(1,
int(rosetta.core.pose.chain_end_res(scaff_pdb[0],1))):
                    mutated_pose =
mutate_residues(resnum,seq_permutation[resnum-1],scaff_pdb[0])

                #dbm.apply(mutated_pose)
                minimize(mutated_pose,sfxn) # to fix the wonky side chains

                mut_score = round(sfxn(mutated_pose), 2) # score the peptide, round
makes it output to 2 decimal places

                # Subtracting the poly_gly score from the "total_score" value so
that we're only looking at the effects of the designed residues.
                total_score = mut_score - scaff_pdb[1]

                if _DEBUG:
                    total_energies = mutated_pose.energies().total_energies()
                    fa_rep =
mutated_pose.energies().total_energies()[pyrosetta.rosetta.core.scoring.ScoreType.fa
_rep]
                    print("POLY_GLY_only_score: ", scaff_pdb[1])
                    print("TOTAL:", total_score)

mutated_pose.dump_pdb('sequence_{}_permutation_number_{}_in_scaffold_{}'.format(pep_
seq.split(".pdb")[0].split("mofdock_asym_")[1],permutation_counter, scaff))

                list_of_tuples.append((permutation_counter, total_score))
                dictionary_seq_scaff.update(dict({scaff : list_of_tuples}))

                permutation_counter +=1

    output_data(pep_seq, dictionary_seq_scaff) # output data to a .txt file.
```

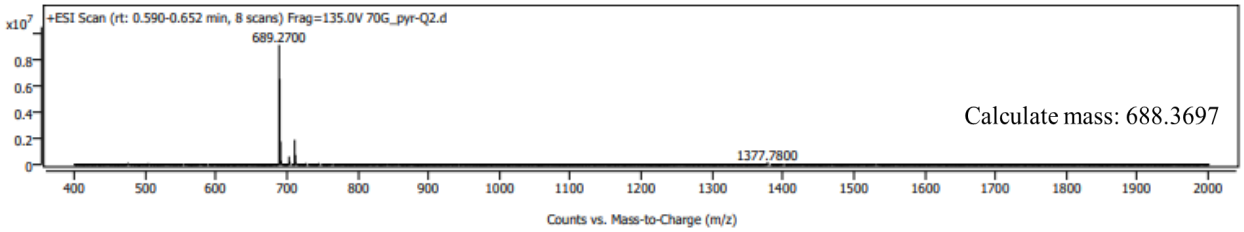
2.6.4 Supplementary Data

Analytical UPLC and LCMS spectra for each peptide
Percent purity is calculated based on area integration of the analytical plot.

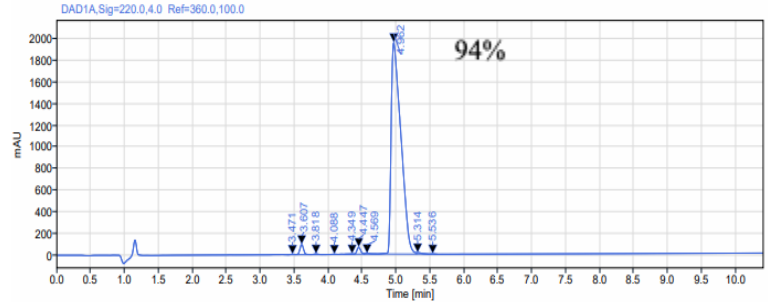
Peptide C2-1: (3-(4-Pyridyl)-alanine - β-Homoproline - α-Aminobutyric acid - 3-(4-Pyridyl)-alanine - β-Homoproline - α-Aminobutyric acid)

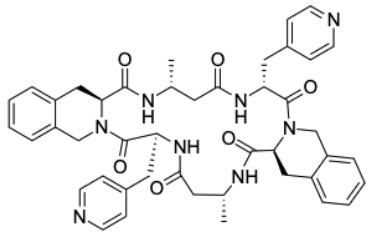


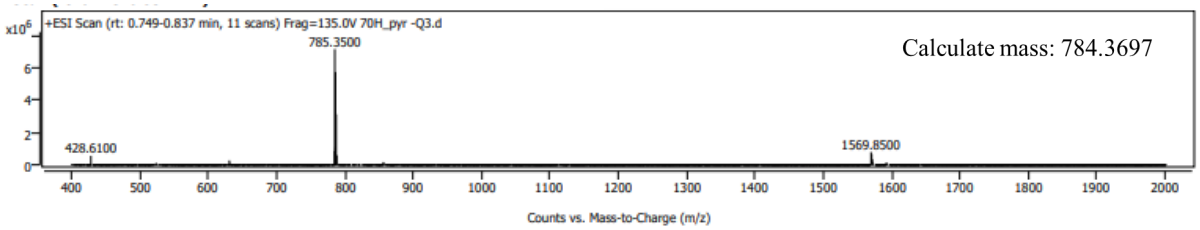




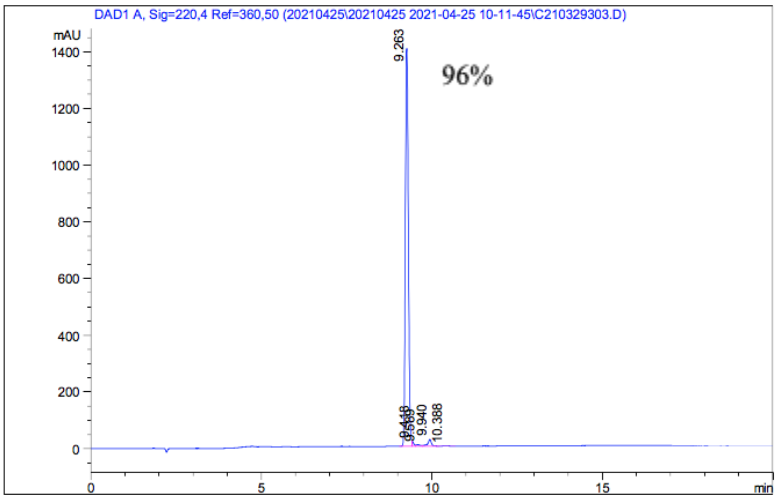
Peptide C2-2: (3-(4-Pyridyl)-alanine -1,2,3,4-tetrahydroisoquinoline-3-carboxylic acid - 3-Aminobutanoic acid -3-(4-Pyridyl)-alanine - 1,2,3,4-tetrahydroisoquinoline-3-carboxylic acid - 3-Aminobutanoic acid)

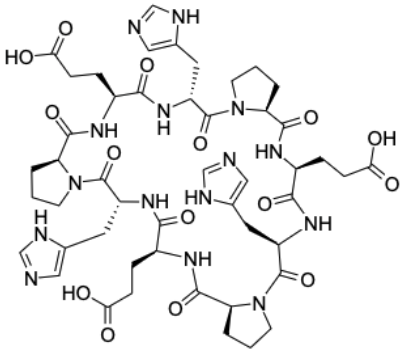


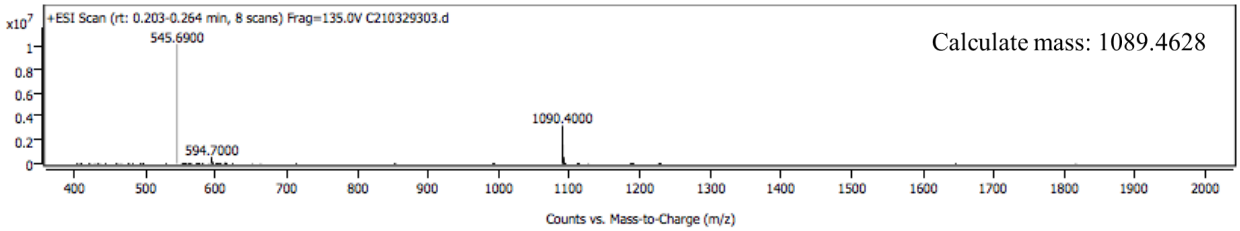




Peptide C3-1: (EhPEhPEhP)

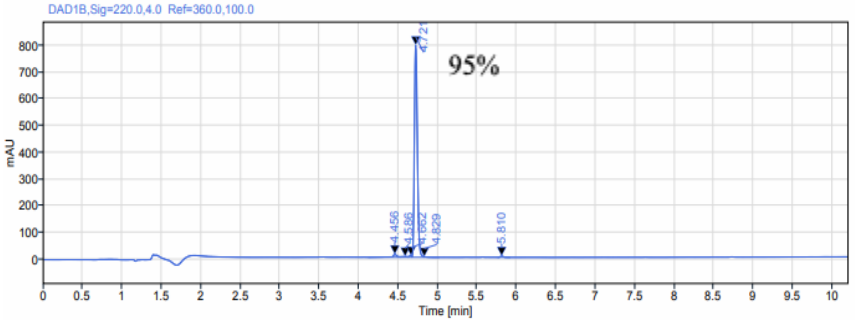


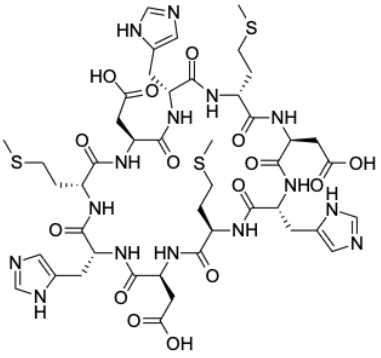


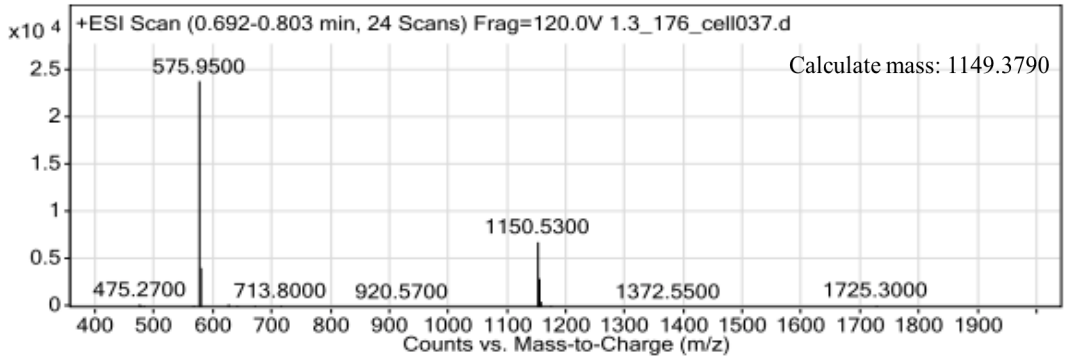


Calculate mass: 1089.4628

Peptide C3-2: (DhmDhmDhm)

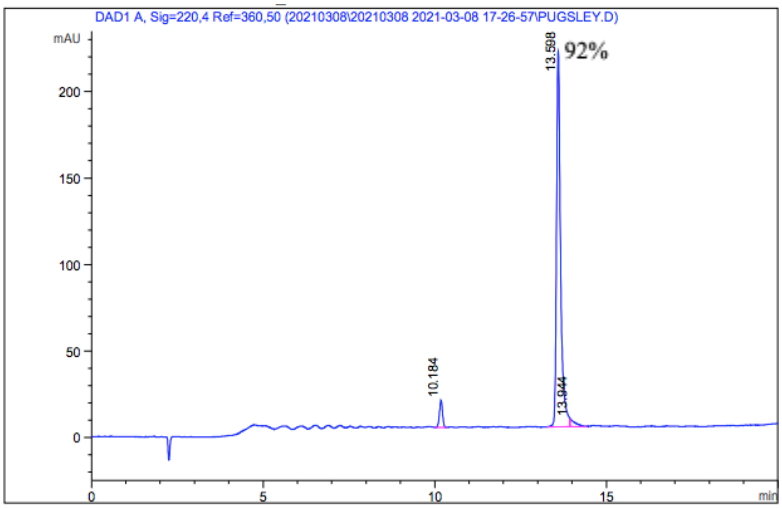


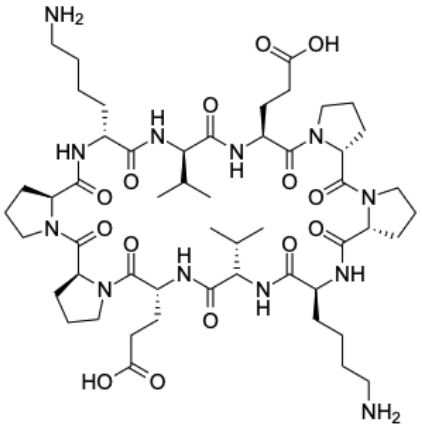


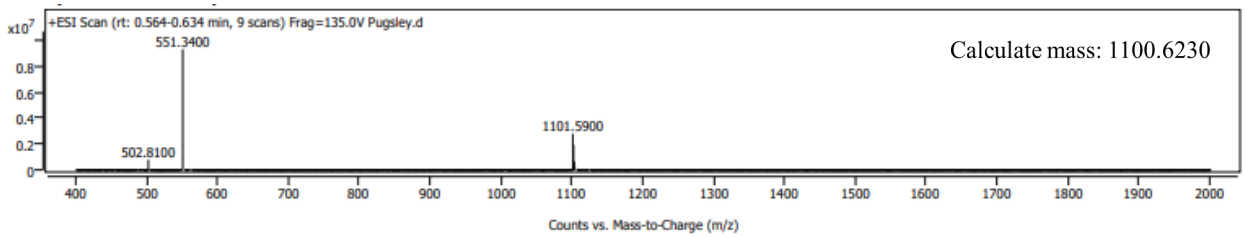


Calculate mass: 1149.3790

Peptide S2-1: (ppKvEPPkVe)

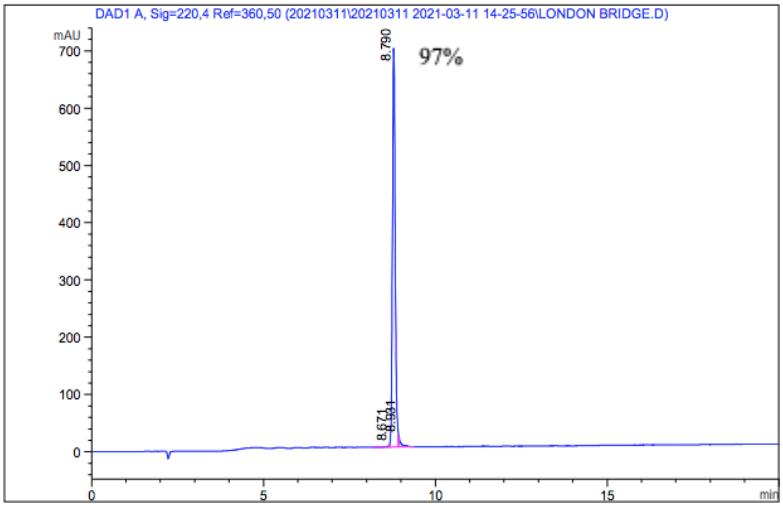


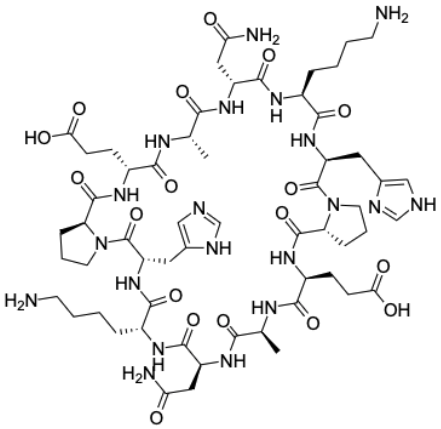


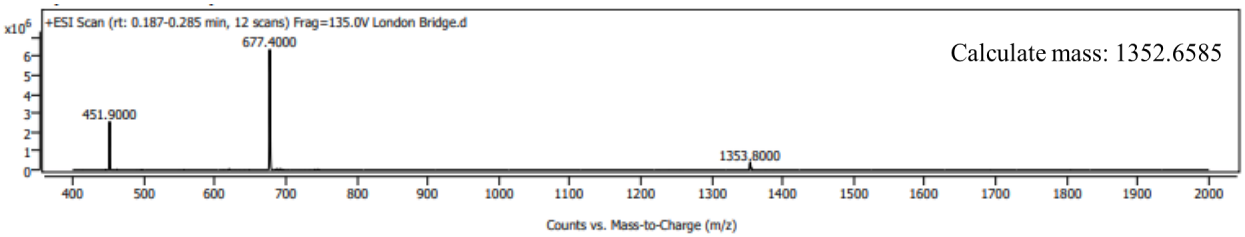


Calculate mass: 1100.6230

Peptide S2-2: (aNkhPeAnKHpE)







Calculate mass: 1352.6585

Chapter 3: Design of peptide inducible dimerization of proteins

3.1 Introduction

Structured cyclic peptides can be used to induce the oligomerization of proteins. Protein oligomerization is critically important for proper cellular functions.[57] It is involved in controlling most cellular processes such as gene expression, signal transduction, and protein transport.[58] Over the past two decades, synthetic biologists have been mimicking and interrogating these interactions using chemically inducible dimers (CIDs).[58–60] Through the modification of natural proteins, scientist have gained spatial and temporal control over cell signals in order to examine complicated cellular pathways in the lab.[61–63] Additionally, CIDs are used to control therapeutic drug activity such as making a toggle switch for chimeric antigen receptors (CARs) t-cell therapy.[64] CARs are engineered receptors clinically approved to induce an immune response against cancer. However, one limitation of this therapy is that it is constitutively turned on after administration. This leads to toxicities due to autoimmune response.[65]

The majority of existing CID systems are rapamycin based.[58,59] Rapamycin is a natural macrocyclic compound that is clinically approved as an immunosuppressant drug. Although the binding of rapamycin to its binding proteins FKBP and FRB makes a robust chemically induced dimer, its many off-target effects make it difficult to use for therapeutic and experimental purposes.[66] In addition to rapamycin based CIDs, multiple CID systems have recently been developed using clinically approved drugs as the chemical inducer. [59,63,67–69] Although successful in controlling protein dimerization, the use of natural proteins and small molecule drugs as chemically inducible dimers introduces undesired off-target effects and toxicities obscuring the results.

We aimed to design completely *de novo* chemically inducible dimers using *de novo* designed helical proteins and cyclic peptides. *De novo* chemically inducible dimeric proteins provide the advantages of thermal stability and orthogonality to biological systems. Additionally, our cyclic peptides are designed to be membrane permeable in order to target intracellular proteins.[11] C2 symmetric peptide induced homodimerization of proteins can be used to control natural protein homodimers, such as tyrosine kinases, caspases, and CAR T-cells for the exploration and regulation of cell signals.

Members of the lab have generated C2 symmetric homodimeric proteins and C2 symmetric peptide using Rosetta molecular modeling. These proteins and peptides have been validated using x-ray crystallography with some matching the designed models to below 2Å Cα RMSD.[39,70] I used Rosetta to dock cyclic peptides into protein pockets and redesign the protein-peptide interface to optimize binding energy. Finally, the designed models were filtered using Rosetta and AlphaFold2. The best designed scaffolds were experimentally tested using equilibrium dialysis and isothermal calorimetry, leading to the validation of multiple low micromolar and nanomolar peptide binders with structural validation using X-ray crystallography. These validated binders can now be optimized for chemical inducibility using a split luciferase assay.

3.2 Experimental and computational methods

3.2.1 Computational protocol

We used the Rifgen/Rifdock protocol described in Cao *et al*. to design peptide binding proteins.[71] Taking crystal structure validated cyclic peptides and AlphaFold2 validated proteins,[15] we use Rifgen to save the inverse rotamers of amino acids with positive interactions with target

peptides into a searchable hash table. Then, we use Rifdock to place the peptide on the C2 symmetry axis of the homodimeric proteins and sample translational and rotational perturbations in the docking model. Docks with high contact surface area between the peptide and the protein and at least six Rifgen matching residues are saved for further interface design to improve the predicted binding affinity.

Chosen docks are designed using Rosetta's FastDesign protocol.[71] The peptide's backbone positions are constrained in the protein pocket. Then, two rounds of FastDesign FastRelax are used to sample various amino acid rotamers and identities on the protein-peptide interface. During this process, the peptide sidechains are allowed to repack but not redesign. Rosetta interface filters such as ddG, ContactMolecularSurface, and SASA are used to extract the top designs.[71] These protein designs are further filtered using AlphaFold2 for structure prediction. If AlphaFold2 predicts a structure that matches the design model with low RMSD and high pLDDT, that design is considered to pass the filter and can be visually scrutinized. Once we have 20-40 filtered designs for each design run, genes encoding the filtered proteins are ordered from IDT for experimental validation. The exact computational protocol is available upon request and will be published with the associated manuscript.

3.2.2 Experimental protocol

The protein expression and purification protocol follows the protocol described in Hicks *et al*.[70] Genes are ordered as eblocks from IDT and are expressed in BL21 E. coli strain from NEB. The golden gate reaction is used to assemble genes into a vector containing a histidine tag as described in Wicky *et al*.[72] Proteins are purified using nickel column affinity chromatography, then size exclusion chromatography. Proteins are concentrated in 25mM Tris

HCL 100mM NaCl buffer pH 8.0 and protein purity and identity is validated using SDS PAGE gel and electrospray ionization mass spectrometry.

Peptide binding is validated using equilibrium dialysis and isothermal titration calorimetry (ITC). [73,74] For equilibrium dialysis, 6 uM peptide is dissolved in 5% DMSO in 25mM Tris HCL 100mM NaCl buffer pH 8.0. The peptide is incubated on one side of an equilibrium dialysis membrane with 100uM protein in the same buffer incubated on the other side of the membrane. The equilibrium dialysis plate is left shaking at 250 RPM at room temperature overnight. Then 20 uL from each side of the membrane is added to 140 uL acetonitrile 0.1% formic acid. This mixture is spun down at 10,000 RPM for 10 minutes to crash down the protein. Peptide concentration in the remaining supernatant is quantified using ESI mass spectrometry and a peptide calibration curve. The ratio of peptide concentration detected on the protein side of the membrane to the peptide concentration detected on the buffer side of the membrane indicates binding affinity of each protein. Proteins that show binding using equilibrium dialysis are further validated using isothermal titration calorimetry (ITC). ITC estimates the binding affinity using 100 uM protein and 250-1000 uM peptide which are each dissolved in 5% DMSO in 25mM Tris HCL 100mM NaCl buffer pH 8.0 each. Finally peptide binding proteins are screened for crystal formation using Hampton Research crystal screening plates. Crystals are sent for X-ray diffraction and data is collected on ALS beamline. Crystal structures are solved as described in Hicks *et al*. [70]

3.3 Results

Rosetta based computational methods have previously been shown to successfully design symmetric cyclic peptides and C2 homo-oligomeric proteins.[10,39,70] Taking these scaffolds as original starting points, We dock and design peptides to bind homo-oligomeric proteins. Then,

we weaken the protein-protein interface to achieve peptide induced oligomerization (Figure 1). In order to redesign the protein pores to bind cyclic peptides, we use Rifgen and Rifdock to generate favorable interactions with the peptide and dock it into thousands of AlphaFold2 validated protein scaffolds (Figure 2a-b and section 3.2.1). Docked protein-peptide complexes are redesigned using Rosetta's FastDesign protocol to optimize peptide-protein interaction based on Rosetta's energy function (Figure 2c). Finally, designs are filtered using Rosetta interface filters and AlphaFold2 filters (Figure 2d-e). We use Rosetta's ddG filter to predict binding energy between the dimeric protein and the peptide, ContactMolecularSurface to predict the interaction surface area between the peptide and the protein, and ShapeComplementarity to predict the interface shape complementarity. These filters are adjusted for each peptide target to filter out the bottom 90% of designs. The best designs are further filtered using AlphaFold2 to select proteins that are predicted to fold within 1.6 Å RMSD to the design model with above 80 pLDDT.

This pipeline was applied to three C2 symmetric peptides. We expressed and purified over a hundred proteins to bind these peptides which were soluble and dimeric by SEC. We used equilibrium dialysis to test for peptide binding. Approximately 10% of tested proteins show some peptide binding by equilibrium dialysis (example for one set of tested proteins is shown in Figure 3). Proteins that show peptide binding using equilibrium dialysis are further validated using ITC. This led to the validation of binders that range from 30 nM Kd to 200 uM Kd. Finally, we use protein crystallization screens to validate the structure of peptide binding proteins. The structure of one peptide binding protein was solved to 2.75 Å resolution (Figure 4). This crystal matches the design model of the protein-peptide complex to 0.9 Å Cα RMSD showing the impressive improvements in protein structure prediction and design.

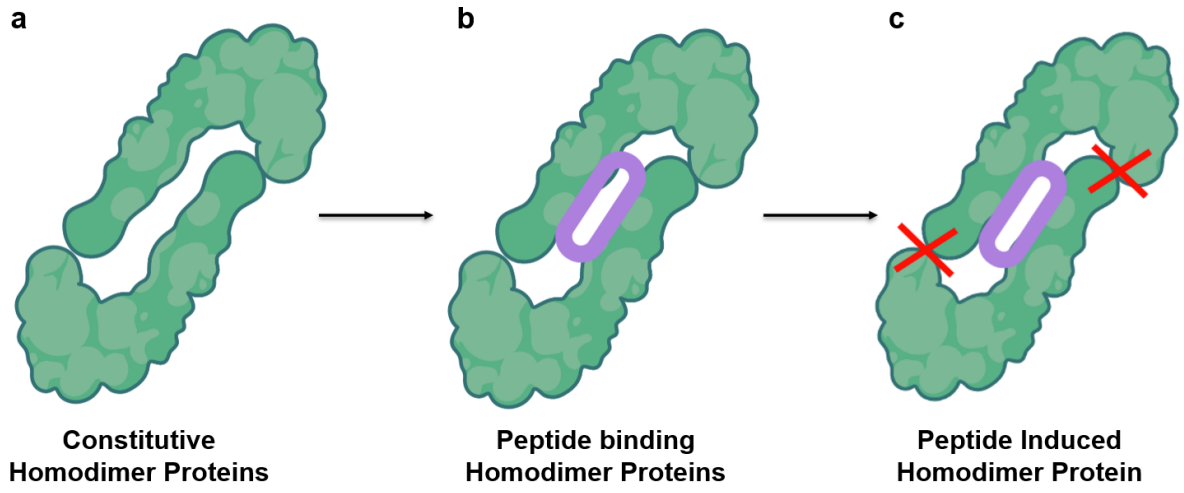


Figure 1. Schematic of the design approach to generate chemically inducible dimers. (a) We use *de novo* designed constitutive homodimer proteins. (b) *De novo* designed C2 symmetric cyclic peptides are docked into the protein pockets and the protein is designed using Rosetta to form favorable interactions with the peptide. (c) The protein-protein interface is weakened through redesign to achieve peptide dependent oligomerization.

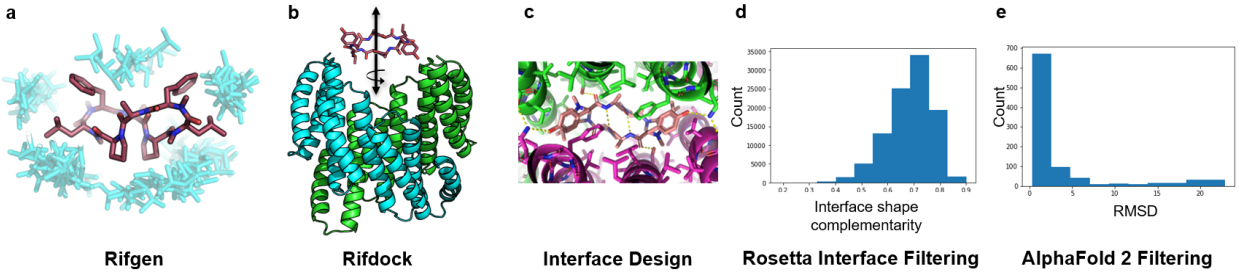


Figure 2. Computational pipeline for designing peptide binding proteins. (a) First, we use Rifgen to generate inverse rotamer clouds for amino acids that form favorable interactions with the cyclic peptide. (b) The peptide is docked into the protein pockets with rotation and translation perturbation around the axis of symmetry using Rifdock. (c) Rosetta is used to design the protein while maintaining C2 symmetry. (d) Designs are filtered using Rosetta interface filters such as interface shape complementarity, ddG, and contact molecular surface. (e) Designs are filtered using AlphaFold2 structure prediction based on RMSD, pLDDT, and pTM filters.

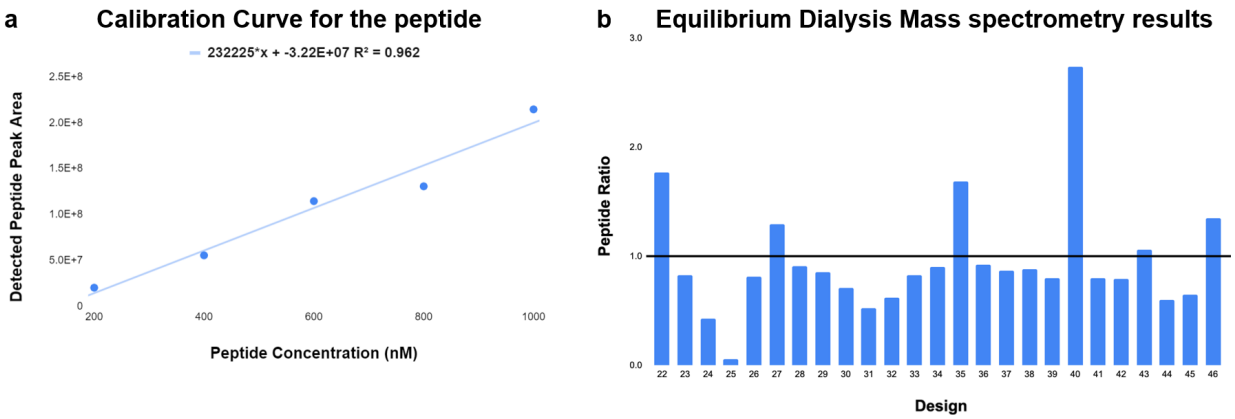


Figure 3. Equilibrium dialysis results for one set of proteins designed to bind an eight residue cyclic peptide. (a) Calibration curve for the peptide as detected by mass spectrometry. (b) Equilibrium dialysis results for 25 designed proteins. 6uM peptide was incubated against 100 uM protein on either side of an equilibrium dialysis membrane. The x-axis shows the results for different protein scaffolds while the y-axis shows the ratio of detected peptide on the protein side of the membrane to the detected peptide on the buffer side of the membrane. The black line at peptide ratio is the control with no protein.

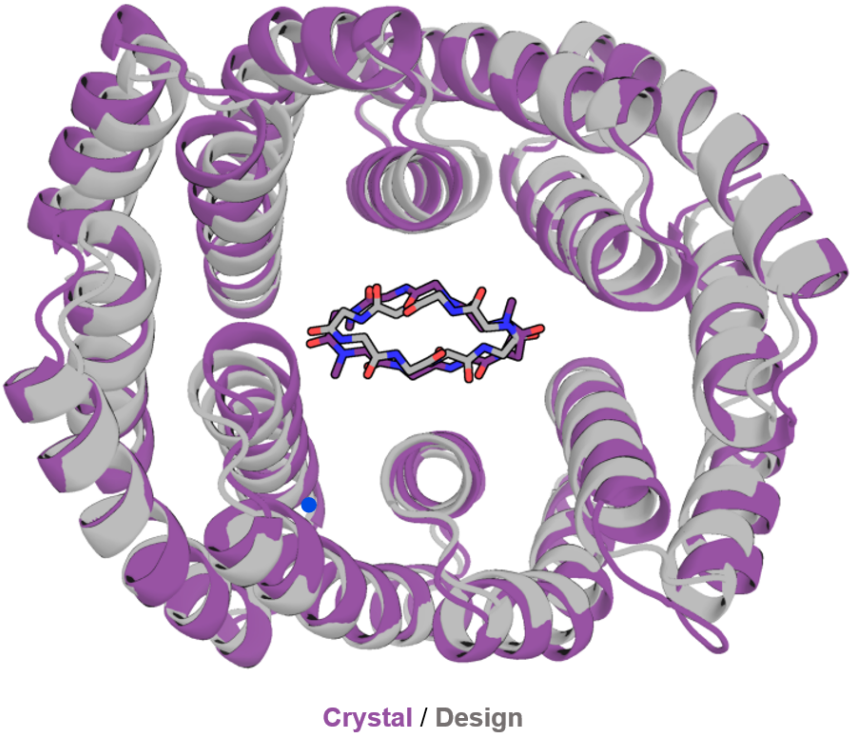


Figure 4. Super position of the Crystal structure (purple) of one peptide binding protein to the designed model (gray). The crystal structure matches the designed model to 0.9 Å Cα RMSD.

3.4 Discussion

Protein structure prediction has advanced significantly in recent years due to the development of deep learning methods such as AlphaFold2 and RoseTTAFold.[15,75] This improvement in structure prediction accuracy allows us to design thousands of homo-oligomeric proteins that match the crystal structure with close agreement and can be used for many applications.[72] These advances aided our efforts to design peptide binding proteins. We successfully designed cyclic peptide binding proteins with micromolar and nanomolar binding affinities. We also crystallized protein homodimers that are within 1Å RMSD to the design model.

Although these advances have improved protein design, accurate peptide binding affinity prediction is still a limitation. It is difficult to model cyclic peptides' dynamics, flexibility, and solvent effects using our current physics base methods on a large scale. Furthermore, due to the cyclic nature of the peptide and the incorporation of non-canonical amino acids, current deep learning protein structure prediction tools are unable to predict or score peptide binding complexes. A variety of new approaches have been developed to solve the ligand docking and affinity prediction problems. Although these approaches show exciting potential to solve the ligand docking problem, they still have some limitations.[76,77] First, there is a tradeoff between prediction accuracy and computational cost for the different methods. Additionally, while deep learning approaches have advanced ligand docking and scoring greatly in the past year, they are still not very generalizable and have limited accuracy.[77]

Although nanomolar peptide binding is very exciting, generating more CID systems will require more high affinity peptide binding proteins. In order to achieve higher affinity peptide binding, we are currently working on designing high throughput yeast display libraries.

We predict that the use of both advanced computational tools and high throughput screening will lead to more tight peptide binding. We could also use this information to develop deep learning algorithms to accurately predict cyclic peptide binding affinity.

Additionally, members of the lab are working on weakening the protein-protein interface (PPI) of current peptide binding homodimeric proteins to establish chemical inducibility. ProteinMPNN is used to redesign the interface with added alanines and polar residues to weaken the interaction energy while maintaining backbone structure. [78] PPI redesigns are then tested using a nanobit assay to validate inducibility. [79] Detailed methods and results for this chapter will be available with the associated manuscript upon publication.

3.5 Conclusion

Although there are multiple chemically inducible dimeric proteins available in the literature, they have limitations due to off target effects and toxicity. We set out to develop the first completely *de novo* CID system using computationally designed *de novo* proteins and cyclic peptides. We use *in silico* validated homodimeric proteins to bind C2 symmetric cyclic peptides. Using a Rifgen/Rifdock and Roestta design approach, we generate hundreds of thousands of peptide binding protein models, we test over a hundred proteins experimentally leading to multiple micromolar and nanomolar peptide binding homodimeric proteins. We also have a crystal structure of one such complex that matches the design model to 0.9 Å Cα RMSD suggesting that our computational protein modeling and peptide docking are accurate to sub angstrom RMSD. While the results are exciting and show the great advances in protein structure prediction and design, more work is needed to design multiple *de novo* CIDs. We are currently working on using high throughput screening methods to achieve higher affinity peptide binding

to generate orthogonal CID systems. We are also reengineering proteins so that their dimerization is dependent on peptide binding to achieve inducibility.

References

(1) Shen, H.; Fallas, J. A.; Lynch, E.; Sheffler, W.; Parry, B.; Jannetty, N.; Decarreau, J.; Wagenbach, M.; Vicente, J. J.; Chen, J.; Wang, L.; Dowling, Q.; Oberdorfer, G.; Stewart, L.; Wordeman, L.; De Yoreo, J.; Jacobs-Wagner, C.; Kollman, J.; Baker, D. De Novo Design of Self-Assembling Helical Protein Filaments. *Science* **2018**, *362* (6415), 705–709.

(2) Sahtoe, D. D.; Praetorius, F.; Courbet, A.; Hsia, Y.; Wicky, B. I. M.; Edman, N. I.; Miller, L. M.; Timmermans, B. J. R.; Decarreau, J.; Morris, H. M.; Kang, A.; Bera, A. K.; Baker, D. Reconfigurable Asymmetric Protein Assemblies through Implicit Negative Design. *Science* **2022**, *375* (6578), eabj7662.

(3) Lajoie, M. J.; Boyken, S. E.; Salter, A. I.; Bruffey, J.; Rajan, A.; Langan, R. A.; Olshefsky, A.; Muhunthan, V.; Bick, M. J.; Gewe, M.; Quijano-Rubio, A.; Johnson, J.; Lenz, G.; Nguyen, A.; Pun, S.; Correnti, C. E.; Riddell, S. R.; Baker, D. Designed Protein Logic to Target Cells with Precise Combinations of Surface Antigens. *Science* **2020**, *369* (6511), 1637–1643.

(4) Chen, Z.; Kibler, R. D.; Hunt, A.; Busch, F.; Pearl, J.; Jia, M.; VanAernum, Z. L.; Wicky, B. I. M.; Dods, G.; Liao, H.; Wilken, M. S.; Ciarlo, C.; Green, S.; El-Samad, H.; Stamatoyannopoulos, J.; Wysocki, V. H.; Jewett, M. C.; Boyken, S. E.; Baker, D. De Novo Design of Protein Logic Gates. *Science* **2020**, *368* (6486), 78–84.

(5) Siegel, J. B.; Zanghellini, A.; Lovick, H. M.; Kiss, G.; Lambert, A. R.; St Clair, J. L.; Gallaher, J. L.; Hilvert, D.; Gelb, M. H.; Stoddard, B. L.; Houk, K. N.; Michael, F. E.; Baker, D. Computational Design of an Enzyme Catalyst for a Stereoselective Bimolecular Diels-Alder Reaction. *Science* **2010**, *329* (5989), 309–313.

(6) Kipnis, Y.; Chaib, A. O.; Vorobieva, A. A.; Cai, G.; Reggiano, G.; Basanta, B.; Kumar, E.; Mittl, P. R. E.; Hilvert, D.; Baker, D. Design and Optimization of Enzymatic Activity in a de Novo β-Barrel Scaffold. *Protein Sci.* **2022**, *31* (11), e4405.

(7) Lee, J.; Schwieter, K. E.; Watkins, A. M.; Kim, D. S.; Yu, H.; Schwarz, K. J.; Lim, J.; Coronado, J.; Byrom, M.; Anslyn, E. V.; Ellington, A. D.; Moore, J. S.; Jewett, M. C. Expanding the Limits of the Second Genetic Code with Ribozymes. *Nat. Commun.* **2019**, *10* (1), 5097.

(8) Gentilucci, L.; De Marco, R.; Cerisoli, L. Chemical Modifications Designed to Improve Peptide Stability: Incorporation of Non-Natural Amino Acids, Pseudo-Peptide Bonds, and Cyclization. *Curr. Pharm. Des.* **2010**, *16* (28), 3185–3203.

(9) Almhjell, P. J.; Mills, J. H. Metal-Chelating Non-Canonical Amino Acids in Metalloprotein Engineering and Design. *Curr. Opin. Struct. Biol.* **2018**, *51*, 170–176.

(10) Hosseinzadeh, P.; Bhardwaj, G.; Mulligan, V. K.; Shortridge, M. D.; Craven, T. W.; Pardo-Avila, F.; Rettie, S. A.; Kim, D. E.; Silva, D.-A.; Ibrahim, Y. M.; Webb, I. K.; Cort, J. R.; Adkins, J. N.; Varani, G.; Baker, D. Comprehensive Computational Design of Ordered Peptide Macrocycles. *Science* **2017**, *358* (6369), 1461–1466.

(11) Bhardwaj, G.; O'Connor, J.; Rettie, S.; Huang, Y.-H.; Ramelot, T. A.; Mulligan, V. K.; Alpkilic, G. G.; Palmer, J.; Bera, A. K.; Bick, M. J.; Di Piazza, M.; Li, X.; Hosseinzadeh, P.; Craven, T. W.; Tejero, R.; Lauko, A.; Choi, R.; Glynn, C.; Dong, L.; Griffin, R.; van Voorhis, W. C.; Rodriguez, J.; Stewart, L.; Montelione, G. T.; Craik, D.; Baker, D. Accurate de Novo Design of Membrane-Traversing Macrocycles. *Cell* **2022**, *185* (19), 3520–3532.e26.

(12) Dong, J.; Liu, Y.; Cui, Y. Artificial Metal-Peptide Assemblies: Bioinspired Assembly of Peptides and Metals through Space and across Length Scales. *J. Am. Chem. Soc.* **2021**, *143* (42), 17316–17336.
(13) Fosgerau, K.; Hoffmann, T. Peptide Therapeutics: Current Status and Future Directions. *Drug Discov. Today* **2015**, *20* (1), 122–128.
(14) Said, M. Y.; Kang, C. S.; Wang, S.; Sheffler, W.; Salveson, P. J.; Bera, A. K.; Kang, A.; Nguyen, H.; Ballard, R.; Li, X.; Bai, H.; Stewart, L.; Levine, P.; Baker, D. Exploration of Structured Symmetric Cyclic Peptides as Ligands for Metal-Organic Frameworks. *Chem. Mater.* **2022**, *34* (21), 9736–9744.
(15) Jumper, J.; Evans, R.; Pritzel, A.; Green, T.; Figurnov, M.; Ronneberger, O.; Tunyasuvunakool, K.; Bates, R.; Žídek, A.; Potapenko, A.; Bridgland, A.; Meyer, C.; Kohl, S. A. A.; Ballard, A. J.; Cowie, A.; Romera-Paredes, B.; Nikolov, S.; Jain, R.; Adler, J.; Back, T.; Petersen, S.; Reiman, D.; Clancy, E.; Zielinski, M.; Steinegger, M.; Pacholska, M.; Berghammer, T.; Bodenstein, S.; Silver, D.; Vinyals, O.; Senior, A. W.; Kavukcuoglu, K.; Kohli, P.; Hassabis, D. Highly Accurate Protein Structure Prediction with AlphaFold. *Nature* **2021**, *596* (7873), 583–589.
(16) Yaghi, O. M.; O'Keeffe, M.; Ockwig, N. W.; Chae, H. K.; Eddaoudi, M.; Kim, J. Reticular Synthesis and the Design of New Materials. *Nature* **2003**, *423* (6941), 705–714.
(17) Furukawa, H.; Cordova, K. E.; O'Keeffe, M.; Yaghi, O. M. The Chemistry and Applications of Metal-Organic Frameworks. *Science* **2013**, *341* (6149), 1230444.
(18) Ji, Z.; Wang, H.; Canossa, S.; Wuttke, S.; Yaghi, O. M. Pore Chemistry of Metal–organic Frameworks. *Adv. Funct. Mater.* **2020**, *30* (41), 2000238.
(19) Zhou, H.-C. "joe"; Kitagawa, S. Metal–Organic Frameworks (MOFs). *Chem. Soc. Rev.* **2014**, *43* (16), 5415–5418.
(20) Bailey, J. B.; Tezcan, F. A. Tunable and Cooperative Thermomechanical Properties of Protein–Metal–Organic Frameworks. *J. Am. Chem. Soc.* **2020**, *142* (41), 17265–17270.
(21) Chiong, J. A.; Zhu, J.; Bailey, J. B.; Kalaj, M.; Subramanian, R. H.; Xu, W.; Cohen, S. M.; Tezcan, F. A. An Exceptionally Stable Metal–Organic Framework Constructed from Chelate-Based Metal–Organic Polyhedra. *J. Am. Chem. Soc.* **2020**, *142* (15), 6907–6912.
(22) Keskin, S.; Kızılel, S. Biomedical Applications of Metal Organic Frameworks. *Ind. Eng. Chem. Res.* **2011**, *50* (4), 1799–1812.
(23) Li, J.-R.; Sculley, J.; Zhou, H.-C. Metal-Organic Frameworks for Separations. *Chem. Rev.* **2012**, *112* (2), 869–932.
(24) Baek, J.; Rungtaweevoranit, B.; Pei, X.; Park, M.; Fakra, S. C.; Liu, Y.-S.; Matheu, R.; Alshmimri, S. A.; Alshehri, S.; Trickett, C. A.; Somorjai, G. A.; Yaghi, O. M. Bioinspired Metal-Organic Framework Catalysts for Selective Methane Oxidation to Methanol. *J. Am. Chem. Soc.* **2018**, *140* (51), 18208–18216.
(25) Kreno, L. E.; Leong, K.; Farha, O. K.; Allendorf, M.; Van Duyne, R. P.; Hupp, J. T. Metal-Organic Framework Materials as Chemical Sensors. *Chem. Rev.* **2012**, *112* (2), 1105–1125.
(26) Lee, J.; Farha, O. K.; Roberts, J.; Scheidt, K. A.; Nguyen, S. T.; Hupp, J. T. Metal-Organic Framework Materials as Catalysts. *Chem. Soc. Rev.* **2009**, *38* (5), 1450–1459.
(27) Huxford, R. C.; Della Rocca, J.; Lin, W. Metal–organic Frameworks as Potential Drug Carriers. *Curr. Opin. Chem. Biol.* **2010**, *14* (2), 262–268.
(28) Xiao, D. J.; Oktawiec, J.; Milner, P. J.; Long, J. R. Pore Environment Effects on Catalytic Cyclohexane Oxidation in Expanded Fe2(dobdc) Analogues. *J. Am. Chem. Soc.* **2016**, *138*

(43), 14371–14379.
(29) Chui, S. S.-Y.; Lo, S. M.-F.; Charmant, J. P. H.; Orpen, A. G.; Williams, I. D. A Chemically Functionalizable Nanoporous Material [Cu 3 (TMA) 2 (H 2 O) 3] N. *Science* **1999**, *283* (5405), 1148–1150.
(30) Deng, H.; Grunder, S.; Cordova, K. E.; Valente, C.; Furukawa, H.; Hmadeh, M.; Gándara, F.; Whalley, A. C.; Liu, Z.; Asahina, S.; Kazumori, H.; O'Keeffe, M.; Terasaki, O.; Stoddart, J. F.; Yaghi, O. M. Large-Pore Apertures in a Series of Metal-Organic Frameworks. *Science* **2012**, *336* (6084), 1018–1023.
(31) Schnitzer, T.; Paenurk, E.; Trapp, N.; Gershoni-Poranne, R.; Wennemers, H. Peptide-Metal Frameworks with Metal Strings Guided by Dispersion Interactions. *J. Am. Chem. Soc.* **2021**, *143* (2), 644–648.
(32) Rabone, J.; Yue, Y.-F.; Chong, S. Y.; Stylianou, K. C.; Bacsa, J.; Bradshaw, D.; Darling, G. R.; Berry, N. G.; Khimyak, Y. Z.; Ganin, A. Y.; Wiper, P.; Claridge, J. B.; Rosseinsky, M. J. An Adaptable Peptide-Based Porous Material. *Science* **2010**, *329* (5995), 1053–1057.
(33) Chino, M.; Maglio, O.; Nastri, F.; Pavone, V.; DeGrado, W. F.; Lombardi, A. Artificial Diiron Enzymes with a DE Novo Designed Four-Helix Bundle Structure. *Eur. J. Inorg. Chem.* **2015**, *2015* (21), 3371–3390.
(34) Kim, J. D.; Pike, D. H.; Tyryshkin, A. M.; Swapna, G. V. T.; Raanan, H.; Montelione, G. T.; Nanda, V.; Falkowski, P. G. Minimal Heterochiral de Novo Designed 4Fe-4S Binding Peptide Capable of Robust Electron Transfer. *J. Am. Chem. Soc.* **2018**, *140* (36), 11210–11213.
(35) Shi, J.; Li, J.; Zeng, H.; Zou, G.; Zhang, Q.; Lin, Z. Water Stable Oxalate-Based Coordination Polymers with in Situ Generated Cyclic Dipeptides Showing High Proton Conductivity. *Dalton Trans.* **2018**, *47* (43), 15288–15292.
(36) Navarro-Sánchez, J.; Argente-García, A. I.; Moliner-Martínez, Y.; Roca-Sanjuán, D.; Antypov, D.; Campíns-Falcó, P.; Rosseinsky, M. J.; Martí-Gastaldo, C. Peptide Metal-Organic Frameworks for Enantioselective Separation of Chiral Drugs. *J. Am. Chem. Soc.* **2017**, *139* (12), 4294–4297.
(37) Katsoulidis, A. P.; Antypov, D.; Whitehead, G. F. S.; Carrington, E. J.; Adams, D. J.; Berry, N. G.; Darling, G. R.; Dyer, M. S.; Rosseinsky, M. J. Chemical Control of Structure and Guest Uptake by a Conformationally Mobile Porous Material. *Nature* **2019**, *565* (7738), 213–217.
(38) Meng, W.; Kondo, S.; Ito, T.; Komatsu, K.; Pirillo, J.; Hijikata, Y.; Ikuhara, Y.; Aida, T.; Sato, H. An Elastic Metal–organic Crystal with a Densely Catenated Backbone. *Nature* **2021**, *598* (7880), 298–303.
(39) Mulligan, V. K.; Kang, C. S.; Sawaya, M. R.; Rettie, S.; Li, X.; Antselovich, I.; Craven, T. W.; Watkins, A. M.; Labonte, J. W.; DiMaio, F.; Yeates, T. O.; Baker, D. Computational Design of Mixed Chirality Peptide Macrocycles with Internal Symmetry. *Protein Sci.* **2020**, *29* (12), 2433–2445.
(40) Laniado, J.; Yeates, T. O. A Complete Rule Set for Designing Symmetry Combination Materials from Protein Molecules. *Proc. Natl. Acad. Sci. U. S. A.* **2020**, *117* (50), 31817–31823.
(41) Zubatyuk, R.; Smith, J. S.; Leszczynski, J.; Isayev, O. Accurate and Transferable Multitask Prediction of Chemical Properties with an Atoms-in-Molecules Neural Network. *Sci Adv* **2019**, *5* (8), eaav6490.
(42) King, N. P.; Bale, J. B.; Sheffler, W.; McNamara, D. E.; Gonen, S.; Gonen, T.; Yeates, T. O.;

Baker, D. Accurate Design of Co-Assembling Multi-Component Protein Nanomaterials. *Nature* **2014**, *510* (7503), 103–108.
(43) Hsia, Y.; Mout, R.; Sheffler, W.; Edman, N. I.; Vulovic, I.; Park, Y.-J.; Redler, R. L.; Bick, M. J.; Bera, A. K.; Courbet, A.; Kang, A.; Brunette, T. J.; Nattermann, U.; Tsai, E.; Saleem, A.; Chow, C. M.; Ekiert, D.; Bhabha, G.; Veesler, D.; Baker, D. Design of Multi-Scale Protein Complexes by Hierarchical Building Block Fusion. *Nat. Commun.* **2021**, *12* (1), 2294.
(44) Macrae, C. F.; Sovago, I.; Cottrell, S. J.; Galek, P. T. A.; McCabe, P.; Pidcock, E.; Platings, M.; Shields, G. P.; Stevens, J. S.; Towler, M.; Wood, P. A. Mercury 4.0: From Visualization to Analysis, Design and Prediction. *Journal of Applied Crystallography*. 2020, pp 226–235. https://doi.org/10.1107/s1600576719014092.
(45) Kabsch, W. XDS. *Acta Crystallogr. D Biol. Crystallogr.* **2010**, *66* (Pt 2), 125–132.
(46) Winn, M. D.; Ballard, C. C.; Cowtan, K. D.; Dodson, E. J.; Emsley, P.; Evans, P. R.; Keegan, R. M.; Krissinel, E. B.; Leslie, A. G. W.; McCoy, A.; McNicholas, S. J.; Murshudov, G. N.; Pannu, N. S.; Potterton, E. A.; Powell, H. R.; Read, R. J.; Vagin, A.; Wilson, K. S. Overview of the CCP4 Suite and Current Developments. *Acta Crystallogr. D Biol. Crystallogr.* **2011**, *67* (Pt 4), 235–242.
(47) Sheldrick, G. M. SHELXT– Integrated Space-Group and Crystal-Structure Determination. *Acta Crystallographica Section A Foundations and Advances*. 2015, pp 3–8. https://doi.org/10.1107/s2053273314026370.
(48) Sheldrick, G. M. Crystal Structure Refinement with SHELXL. *Acta Crystallogr. B* **2015**, *71* (Pt 1), 3–8.
(49) Emsley, P.; Cowtan, K. Coot: Model-Building Tools for Molecular Graphics. *Acta Crystallogr. D Biol. Crystallogr.* **2004**, *60* (Pt 12 Pt 1), 2126–2132.
(50) Hübschle, C. B.; Sheldrick, G. M.; Dittrich, B. ShelXle: A Qt Graphical User Interface for SHELXL. *J. Appl. Crystallogr.* **2011**, *44* (Pt 6), 1281–1284.
(51) Leman, J. K.; Weitzner, B. D.; Lewis, S. M.; Adolf-Bryfogle, J.; Alam, N.; Alford, R. F.; Aprahamian, M.; Baker, D.; Barlow, K. A.; Barth, P.; Basanta, B.; Bender, B. J.; Blacklock, K.; Bonet, J.; Boyken, S. E.; Bradley, P.; Bystroff, C.; Conway, P.; Cooper, S.; Correia, B. E.; Coventry, B.; Das, R.; De Jong, R. M.; DiMaio, F.; Dsilva, L.; Dunbrack, R.; Ford, A. S.; Frenz, B.; Fu, D. Y.; Geniesse, C.; Goldschmidt, L.; Gowthaman, R.; Gray, J. J.; Gront, D.; Guffy, S.; Horowitz, S.; Huang, P.-S.; Huber, T.; Jacobs, T. M.; Jeliazkov, J. R.; Johnson, D. K.; Kappel, K.; Karanicolas, J.; Khakzad, H.; Khar, K. R.; Khare, S. D.; Khatib, F.; Khramushin, A.; King, I. C.; Kleffner, R.; Koepnick, B.; Kortemme, T.; Kuenze, G.; Kuhlman, B.; Kuroda, D.; Labonte, J. W.; Lai, J. K.; Lapidoth, G.; Leaver-Fay, A.; Lindert, S.; Linsky, T.; London, N.; Lubin, J. H.; Lyskov, S.; Maguire, J.; Malmström, L.; Marcos, E.; Marcu, O.; Marze, N. A.; Meiler, J.; Moretti, R.; Mulligan, V. K.; Nerli, S.; Norn, C.; Ó'Conchúir, S.; Ollikainen, N.; Ovchinnikov, S.; Pacella, M. S.; Pan, X.; Park, H.; Pavlovicz, R. E.; Pethe, M.; Pierce, B. G.; Pilla, K. B.; Raveh, B.; Renfrew, P. D.; Burman, S. S. R.; Rubenstein, A.; Sauer, M. F.; Scheck, A.; Schief, W.; Schueler-Furman, O.; Sedan, Y.; Sevy, A. M.; Sgourakis, N. G.; Shi, L.; Siegel, J. B.; Silva, D.-A.; Smith, S.; Song, Y.; Stein, A.; Szegedy, M.; Teets, F. D.; Thyme, S. B.; Wang, R. Y.-R.; Watkins, A.; Zimmerman, L.; Bonneau, R. Macromolecular Modeling and Design in Rosetta: Recent Methods and Frameworks. *Nat. Methods* **2020**, *17* (7), 665–680.
(52) Alford, R. F.; Leaver-Fay, A.; Jeliazkov, J. R.; O'Meara, M. J.; DiMaio, F. P.; Park, H.; Shapovalov, M. V.; Renfrew, P. D.; Mulligan, V. K.; Kappel, K.; Labonte, J. W.; Pacella, M.

S.; Bonneau, R.; Bradley, P.; Dunbrack, R. L., Jr; Das, R.; Baker, D.; Kuhlman, B.; Kortemme, T.; Gray, J. J. The Rosetta All-Atom Energy Function for Macromolecular Modeling and Design. *J. Chem. Theory Comput.* **2017**, *13* (6), 3031–3048.

(53) Holm, R. H.; Kennepohl, P.; Solomon, E. I. Structural and Functional Aspects of Metal Sites in Biology. *Chem. Rev.* **1996**, *96* (7), 2239–2314.

(54) Párraga, G.; Horvath, S. J.; Eisen, A.; Taylor, W. E.; Hood, L.; Young, E. T.; Klevit, R. E. Zinc-Dependent Structure of a Single-Finger Domain of Yeast ADR1. *Science* **1988**, *241* (4872), 1489–1492.

(55) Yeates, T. O.; Kent, S. B. H. Racemic Protein Crystallography. *Annu. Rev. Biophys.* **2012**, *41*, 41–61.

(56) Dou, J.; Vorobieva, A. A.; Sheffler, W.; Doyle, L. A.; Park, H.; Bick, M. J.; Mao, B.; Foight, G. W.; Lee, M. Y.; Gagnon, L. A.; Carter, L.; Sankaran, B.; Ovchinnikov, S.; Marcos, E.; Huang, P.-S.; Vaughan, J. C.; Stoddard, B. L.; Baker, D. De Novo Design of a Fluorescence-Activating β-Barrel. *Nature* **2018**, *561* (7724), 485–491.

(57) Zhanhua, C.; Gan, J. G.-K.; Lei, L.; Sakharkar, M. K.; Kangueane, P. Protein Subunit Interfaces: Heterodimers versus Homodimers. *Bioinformation* **2005**, *1* (2), 28–39.

(58) Stanton, B. Z.; Chory, E. J.; Crabtree, G. R. Chemically Induced Proximity in Biology and Medicine. *Science* **2018**, *359* (6380). https://doi.org/10.1126/science.aao5902.

(59) Voß, S.; Klewer, L.; Wu, Y.-W. Chemically Induced Dimerization: Reversible and Spatiotemporal Control of Protein Function in Cells. *Curr. Opin. Chem. Biol.* **2015**, *28*, 194–201.

(60) Soini, L.; Leysen, S.; Davis, J.; Ottmann, C. Molecular Glues to Stabilise Protein–protein Interactions. *Curr. Opin. Chem. Biol.* **2022**, *69*, 102169.

(61) Spencer, D. M.; Wandless, T. J.; Schreiber, S. L.; Crabtree, G. R. Controlling Signal Transduction with Synthetic Ligands. *Science*. 1993, pp 1019–1024. https://doi.org/10.1126/science.7694365.

(62) Miyamoto, T.; DeRose, R.; Suarez, A.; Ueno, T.; Chen, M.; Sun, T.-P.; Wolfgang, M. J.; Mukherjee, C.; Meyers, D. J.; Inoue, T. Rapid and Orthogonal Logic Gating with a Gibberellin-Induced Dimerization System. *Nat. Chem. Biol.* **2012**, *8* (5), 465–470.

(63) Foight, G. W.; Wang, Z.; Wei, C. T.; Greisen, P., Jr; Warner, K. M.; Cunningham-Bryant, D.; Park, K.; Brunette, T. J.; Sheffler, W.; Baker, D.; Maly, D. J. Multi-Input Chemical Control of Protein Dimerization for Programming Graded Cellular Responses. *Nat. Biotechnol.* **2019**, *37* (10), 1209–1216.

(64) Zajc, C. U.; Dobersberger, M.; Schaffner, I.; Mlynek, G.; Pühringer, D.; Salzer, B.; Djinović-Carugo, K.; Steinberger, P.; De Sousa Linhares, A.; Yang, N. J.; Obinger, C.; Holter, W.; Traxlmayr, M. W.; Lehner, M. A Conformation-Specific ON-Switch for Controlling CAR T Cells with an Orally Available Drug. *Proc. Natl. Acad. Sci. U. S. A.* **2020**, *117* (26), 14926–14935.

(65) Neelapu, S. S.; Tummala, S.; Kebriaei, P.; Wierda, W.; Gutierrez, C.; Locke, F. L.; Komanduri, K. V.; Lin, Y.; Jain, N.; Daver, N.; Westin, J.; Gulbis, A. M.; Loghin, M. E.; de Groot, J. F.; Adkins, S.; Davis, S. E.; Rezvani, K.; Hwu, P.; Shpall, E. J. Chimeric Antigen Receptor T-Cell Therapy - Assessment and Management of Toxicities. *Nat. Rev. Clin. Oncol.* **2018**, *15* (1), 47–62.

(66) Liu, Y.; Yang, F.; Zou, S.; Qu, L. Rapamycin: A Bacteria-Derived Immunosuppressant That Has Anti-Atherosclerotic Effects and Its Clinical Application. *Front. Pharmacol.* **2018**, *9*, 1520.

(67) Guo, Z.; Smutok, O.; Johnston, W. A.; Walden, P.; Ungerer, J. P. J.; Peat, T. S.; Newman, J.; Parker, J.; Nebl, T.; Hepburn, C.; Melman, A.; Suderman, R. J.; Katz, E.; Alexandrov, K. Design of a Methotrexate-Controlled Chemical Dimerization System and Its Use in Bio-Electronic Devices. *Nat. Commun.* **2021**, *12* (1), 7137.
(68) Liang, F.-S.; Ho, W. Q.; Crabtree, G. R. Engineering the ABA Plant Stress Pathway for Regulation of Induced Proximity. *Sci. Signal.* **2011**, *4* (164), rs2.
(69) Chin, S.; Schindler, C.; Vinall, L.; Dodd, R.; Bamber, L.; Legg, S.; Sigurdardottir, A.; Rees, D.; Malcolm, T.; Spratley, S.; Tigue, N. A Novel Molecular Switch for Control of Cell and Gene Therapies Based on a Clinically Approved Small Molecule. *Research Square*, 2022. https://doi.org/10.21203/rs.3.rs-2244031/v1.
(70) Hicks, D. R.; Kennedy, M. A.; Thompson, K. A.; DeWitt, M.; Coventry, B.; Kang, A.; Bera, A. K.; Brunette, T. J.; Sankaran, B.; Stoddard, B.; Baker, D. De Novo Design of Protein Homodimers Containing Tunable Symmetric Protein Pockets. *Proc. Natl. Acad. Sci. U. S. A.* **2022**, *119* (30), e2113400119.
(71) Cao, L.; Coventry, B.; Goreshnik, I.; Huang, B.; Sheffler, W.; Park, J. S.; Jude, K. M.; Marković, I.; Kadam, R. U.; Verschueren, K. H. G.; Verstraete, K.; Walsh, S. T. R.; Bennett, N.; Phal, A.; Yang, A.; Kozodoy, L.; DeWitt, M.; Picton, L.; Miller, L.; Strauch, E.-M.; DeBouver, N. D.; Pires, A.; Bera, A. K.; Halabiya, S.; Hammerson, B.; Yang, W.; Bernard, S.; Stewart, L.; Wilson, I. A.; Ruohola-Baker, H.; Schlessinger, J.; Lee, S.; Savvides, S. N.; Garcia, K. C.; Baker, D. Design of Protein-Binding Proteins from the Target Structure Alone. *Nature* **2022**, *605* (7910), 551–560.
(72) Wicky, B. I. M.; Milles, L. F.; Courbet, A.; Ragotte, R. J.; Dauparas, J.; Kinfu, E.; Tipps, S.; Kibler, R. D.; Baek, M.; DiMaio, F.; Li, X.; Carter, L.; Kang, A.; Nguyen, H.; Bera, A. K.; Baker, D. Hallucinating Symmetric Protein Assemblies. *Science* **2022**, *378* (6615), 56–61.
(73) van Liempd, S.; Morrison, D.; Sysmans, L.; Nelis, P.; Mortishire-Smith, R. Development and Validation of a Higher-Throughput Equilibrium Dialysis Assay for Plasma Protein Binding. *J. Lab. Autom.* **2011**, *16* (1), 56–67.
(74) Duff, M. R., Jr; Grubbs, J.; Howell, E. E. Isothermal Titration Calorimetry for Measuring Macromolecule-Ligand Affinity. *J. Vis. Exp.* **2011**, No. 55. https://doi.org/10.3791/2796.
(75) Baek, M.; DiMaio, F.; Anishchenko, I.; Dauparas, J.; Ovchinnikov, S.; Lee, G. R.; Wang, J.; Cong, Q.; Kinch, L. N.; Schaeffer, R. D.; Millán, C.; Park, H.; Adams, C.; Glassman, C. R.; DeGiovanni, A.; Pereira, J. H.; Rodrigues, A. V.; van Dijk, A. A.; Ebrecht, A. C.; Opperman, D. J.; Sagmeister, T.; Buhlheller, C.; Pavkov-Keller, T.; Rathinaswamy, M. K.; Dalwadi, U.; Yip, C. K.; Burke, J. E.; Garcia, K. C.; Grishin, N. V.; Adams, P. D.; Read, R. J.; Baker, D. Accurate Prediction of Protein Structures and Interactions Using a Three-Track Neural Network. *Science* **2021**, *373* (6557), 871–876.
(76) Gorgulla, C. Recent Developments in Structure-Based Virtual Screening Approaches. *arXiv [q-bio.BM]*, 2022. http://arxiv.org/abs/2211.03208.
(77) Isert, C.; Atz, K.; Schneider, G. Structure-Based Drug Design with Geometric Deep Learning. *arXiv [physics.chem-ph]*, 2022. http://arxiv.org/abs/2210.11250.
(78) Dauparas, J.; Anishchenko, I.; Bennett, N.; Bai, H.; Ragotte, R. J.; Milles, L. F.; Wicky, B. I. M.; Courbet, A.; de Haas, R. J.; Bethel, N.; Leung, P. J. Y.; Huddy, T. F.; Pellock, S.; Tischer, D.; Chan, F.; Koepnick, B.; Nguyen, H.; Kang, A.; Sankaran, B.; Bera, A. K.; King, N. P.; Baker, D. Robust Deep Learning-Based Protein Sequence Design Using ProteinMPNN. *Science* **2022**, *378* (6615), 49–56.
(79) Dixon, A. S.; Schwinn, M. K.; Hall, M. P.; Zimmerman, K.; Otto, P.; Lubben, T. H.; Butler,

B. L.; Binkowski, B. F.; Machleidt, T.; Kirkland, T. A.; Wood, M. G.; Eggers, C. T.; Encell, L. P.; Wood, K. V. NanoLuc Complementation Reporter Optimized for Accurate Measurement of Protein Interactions in Cells. *ACS Chem. Biol.* **2016**, *11* (2), 400–408.