



Article

<https://doi.org/10.1038/s41592-025-02626-1>

Atomic context-conditioned protein sequence design using LigandMPNN

Received: 2 December 2023

Accepted: 10 February 2025

Published online: 28 March 2025

Check for updates

Justas Dauparas^{1,2}, Gyu Rie Lee^{1,2,3}, Robert Pecoraro^{1,2,4}, Linna An^{1,2}, Ivan Anishchenko^{1,2}, Cameron Glasscock^{1,2} & David Baker^{1,2,3}✉

Protein sequence design in the context of small molecules, nucleotides and metals is critical to enzyme and small-molecule binder and sensor design, but current state-of-the-art deep-learning-based sequence design methods are unable to model nonprotein atoms and molecules. Here we describe a deep-learning-based protein sequence design method called LigandMPNN that explicitly models all nonprotein components of biomolecular systems. LigandMPNN significantly outperforms Rosetta and ProteinMPNN on native backbone sequence recovery for residues interacting with small molecules (63.3% versus 50.4% and 50.5%), nucleotides (50.5% versus 35.2% and 34.0%) and metals (77.5% versus 36.0% and 40.6%). LigandMPNN generates not only sequences but also sidechain conformations to allow detailed evaluation of binding interactions. LigandMPNN has been used to design over 100 experimentally validated small-molecule and DNA-binding proteins with high affinity and high structural accuracy (as indicated by four X-ray crystal structures), and redesign of Rosetta small-molecule binder designs has increased binding affinity by as much as 100-fold. We anticipate that LigandMPNN will be widely useful for designing new binding proteins, sensors and enzymes.

De novo protein design enables the creation of novel proteins with new functions, such as catalysis¹, DNA, small-molecule and metal binding, and protein–protein interactions^{2–10}. De novo design is often carried out in three steps^{11–14}: first, the generation of protein backbones predicted to be near optimal for carrying out the new desired function^{15–19}; second, design of amino-acid sequences for each backbone to drive folding to the target structure and to make the specific interactions required for function (for example, an enzyme active site)^{20–30}; and third, sequence–structure compatibility filtering using structure prediction methods^{31–36}. In this Article, we focus on the second step, protein sequence design. Both physically based methods such as Rosetta^{37–39} and deep-learning-based models such as ProteinMPNN²⁸, IF-ESM²⁹ and others^{31–36} have been developed to solve this problem. The deep-learning-based methods outperform physically based methods in designing sequences for protein backbones, but currently available

models cannot incorporate nonprotein atoms and molecules. For example, ProteinMPNN explicitly considers only protein backbone coordinates while ignoring any other atomic context, which is critical for designing enzymes, nucleic-acid-binding proteins, sensors and all other protein functions involving interactions with nonprotein atoms.

Results

To enable the design of this wide range of protein functions, we set out to develop a deep-learning method for protein sequence design that explicitly models the full nonprotein atomic context. We sought to do this by generalizing the ProteinMPNN architecture to incorporate nonprotein atoms. As with ProteinMPNN, we treat protein residues as nodes and introduce nearest-neighbor edges based on Cα–Cα distances to define a sparse protein graph (Fig. 1); protein backbone geometry is encoded into graph edges through pairwise distances between N, Cα,

¹Department of Biochemistry, University of Washington, Seattle, WA, USA. ²Institute for Protein Design, University of Washington, Seattle, WA, USA.³Howard Hughes Medical Institute, University of Washington, Seattle, WA, USA. ⁴Department of Physics, University of Washington, Seattle, WA, USA.✉e-mail: dabaker@uw.edu

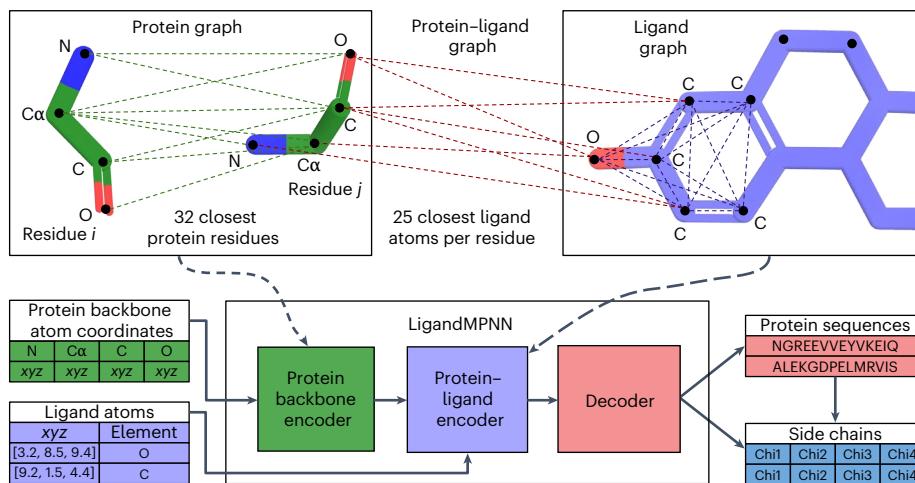


Fig. 1 | The LigandMPNN model. LigandMPNN operates on three different graphs. First, a protein-only graph with residues as nodes and 25 distances between N, C α , C, O and virtual (inferred location based on backbone coordinates to handle the glycine case) C β atoms for residues i and j . Second, an intraligand graph with atoms as nodes that encodes chemical element types and distances between atoms as edges. Third, a protein–ligand graph with residues and ligand atoms as nodes and edges encoding residue j and ligand atom geometry.

The LigandMPNN model has three neural network blocks: a protein backbone encoder, a protein–ligand encoder and a decoder. Protein sequences and sidechain torsion angles are autoregressively decoded to obtain sequence and full protein structure samples. The dotted lines show atom interactions. Metaparameter variation and ablation experiments are described in Supplementary Fig. 1a–e.

C, O and C β atoms. These input features are then processed using three encoder layers with 128 hidden dimensions to obtain intermediate node and edge representations. We experimented with introducing two additional protein–ligand encoder layers to encode protein–ligand interactions. We reasoned that, with the backbone and ligand atoms fixed in space, only ligand atoms in the immediate neighborhood (within ~10 Å) would affect amino-acid sidechain identities and conformations because the interactions (van der Waals, electrostatic, repulsive and solvation) between ligands and sidechains are relatively short range⁴⁰.

To transfer information from ligand atoms to protein residues, we construct a protein–ligand graph with protein residues and ligand atoms as nodes and edges between each protein residue and the closest ligand atoms. We also build a fully connected ligand graph for each protein residue with its nearest-neighbor ligand atoms as nodes; message passing between ligand atoms increases the richness of the information transferred to the protein through the ligand–protein edges. We obtained the best performance by selecting for the protein–ligand and individual residue intraligand graphs the 25 closest ligand atoms based on protein virtual C β and ligand atom distances (Supplementary Fig. 1a). The ligand graph nodes are initialized to one-hot-encoded chemical element types, and the ligand graph edges to the distances between the atoms (Fig. 1). The protein–ligand graph edges encode distances between N, C α , C, O and virtual C β atoms and ligand atoms (Fig. 1). The protein–ligand encoder consists of two message-passing blocks that update the ligand graph representation and then the protein–ligand graph representation. The output of the protein–ligand encoder is combined with the protein encoder node representations and passed into the decoder layers. We call this combined protein–ligand sequence design model LigandMPNN.

To facilitate the design of symmetric^{9,16} and multistate proteins¹⁰, we use a random autoregressive decoding scheme to decode the amino-acid sequence as in the case of ProteinMPNN. With the addition of the ligand atom geometry encoding and the extra two protein–ligand encoder layers, the LigandMPNN neural network has 2.62 million parameters compared with 1.66 million ProteinMPNN parameters. Both networks are high-speed and lightweight (ProteinMPNN 0.6 s and LigandMPNN 0.9 s on a single central processing unit for 100 residues), scaling linearly with respect to the protein length. We augmented the training dataset

by randomly selecting a small fraction of protein residues (2–4%) and using their sidechain atoms as context ligand atoms in addition to any small-molecule, nucleotide and metal context. Although this augmentation did not significantly increase sequence recoveries (Supplementary Fig. 1b), training in this way also enables the direct input of sidechain atom coordinates to LigandMPNN to stabilize functional sites of interest.

We also trained a sidechain packing neural network using the basic LigandMPNN architecture to predict the four sidechain torsion angles for each residue following the sequence design step. The sidechain packing model takes as input the coordinates of the protein backbone and any ligand atoms, and the amino-acid sequence, and outputs the coordinates of the protein sidechains with log-probability scores. The model predicts a mixture (three components) of circular normal distributions for the torsion angles (chi1, chi2, chi3 and chi4). For each residue, we predict three mixing coefficients, three means and three variances per chi angle. We autoregressively decompose the joint chi angle distribution by decoding all chi1 angles first, then all chi2 angles, chi3 angles and finally all chi4 angles (after the model decodes one of the chi angles, its angular value and the associated three-dimensional atom coordinates are used for further decoding).

LigandMPNN was trained on protein assemblies in the Protein Data Bank (PDB; as of 16 December 2022) determined by X-ray crystallography or cryo-electron microscopy to better than 3.5 Å resolution and with a total length of less than 6,000 residues. The train–test split was based on protein sequences clustered at a 30% sequence identity cutoff. We evaluated LigandMPNN sequence design performance on a test set of 317 protein structures containing small molecules, 74 with nucleic acids and 83 with a transition metal (Fig. 2a). For fair comparison, we retrained ProteinMPNN on the same training dataset of PDB biounits as LigandMPNN (the retrained model is referred to as ProteinMPNN in this Article), except none of the context atoms was provided during training. Protein and context atoms were noised by adding 0.1 Å standard deviation Gaussian noise to avoid protein backbone memorization²⁸. We determined the native amino-acid residue sequence recovery for positions close to the ligand (with sidechain atoms within 5.0 Å of any nonprotein atoms). The median sequence recoveries (ten designed sequences per protein) near small molecules were 50.4% for Rosetta using the genpot energy function¹⁸, 50.4% for ProteinMPNN

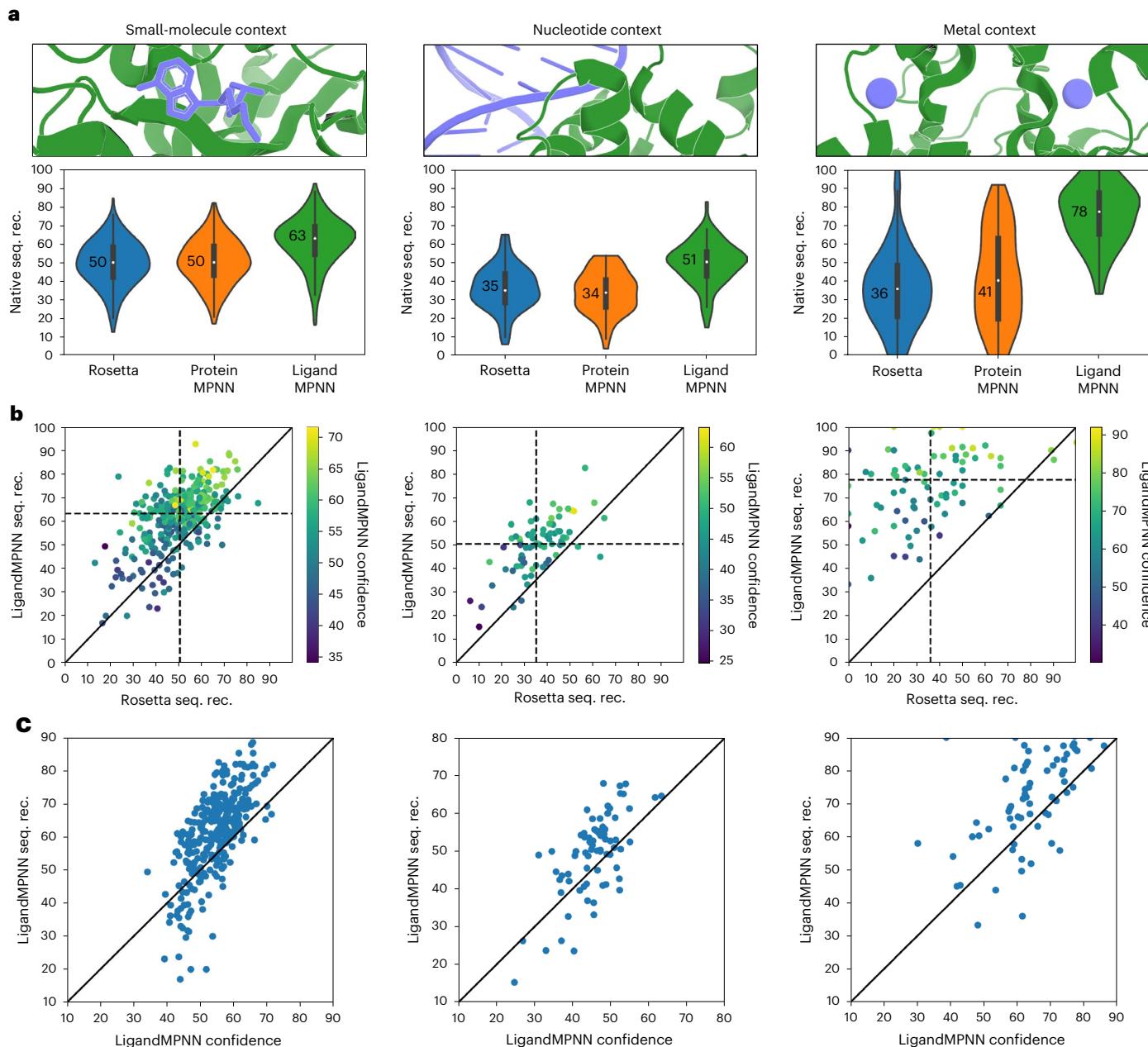


Fig. 2 | In silico evaluation of LigandMPNN sequence design. **a**, LigandMPNN has a higher recovery of native protein amino-acid identities than Rosetta and ProteinMPNN around small molecules, nucleic acids and metals. Sequence recoveries (sec. rec.) are averaged over the residues within 5.0 Å from the context atoms. **b**, LigandMPNN has higher sequence recovery around nonprotein molecules than Rosetta for most proteins. The color indicates the

LigandMPNN-predicted confidence (between 0 and 100) for a given protein. The dashed lines show the mean values. **c**, Native sequence recovery correlates with LigandMPNN predicted confidence for designed sequences. One dot represents an average sequence recovery over 10 sequences for one protein for 317 small-molecule-, 74 nucleotide- and 83 metal-containing test proteins.

and 63.3% for LigandMPNN. For residues near nucleotides, median sequence recoveries were 35.2% for Rosetta² (using an energy function optimized for protein–DNA interfaces), 34.0% for ProteinMPNN and 50.5% for LigandMPNN, and for residues near metals, 36.0% for Rosetta⁴¹, 40.6% for ProteinMPNN and 77.5% for LigandMPNN (Fig. 2a). Sequence recoveries were consistently higher for LigandMPNN over most proteins in the validation dataset (Fig. 2b; performance was correlated, probably reflecting variation in the crystal structure and the amino-acid composition of the site). LigandMPNN predicts amino-acid probability distributions and uncertainties for each residue position; the expected confidence correlates with the actual sequence recovery accuracy (Fig. 3c).

To assess the contributions to this high sustained performance, we evaluated versions in which metaparameters and features were varied or ablated (Supplementary Fig. 1a–e). Decreasing the number of context atoms per residue primarily diminished sequence recovery around nucleic acids, probably because these are larger and contain more atoms on average than small molecules and metals (Supplementary Fig. 1a). Providing sidechain atoms as additional context did not significantly affect LigandMPNN performance (Supplementary Fig. 1b). As observed for ProteinMPNN, sequence recovery is inversely proportional to the amount of Gaussian noise added to input coordinates. The baseline model was trained with 0.1 Å standard deviation noise to reduce the extent to which the native amino acid can be read out simply

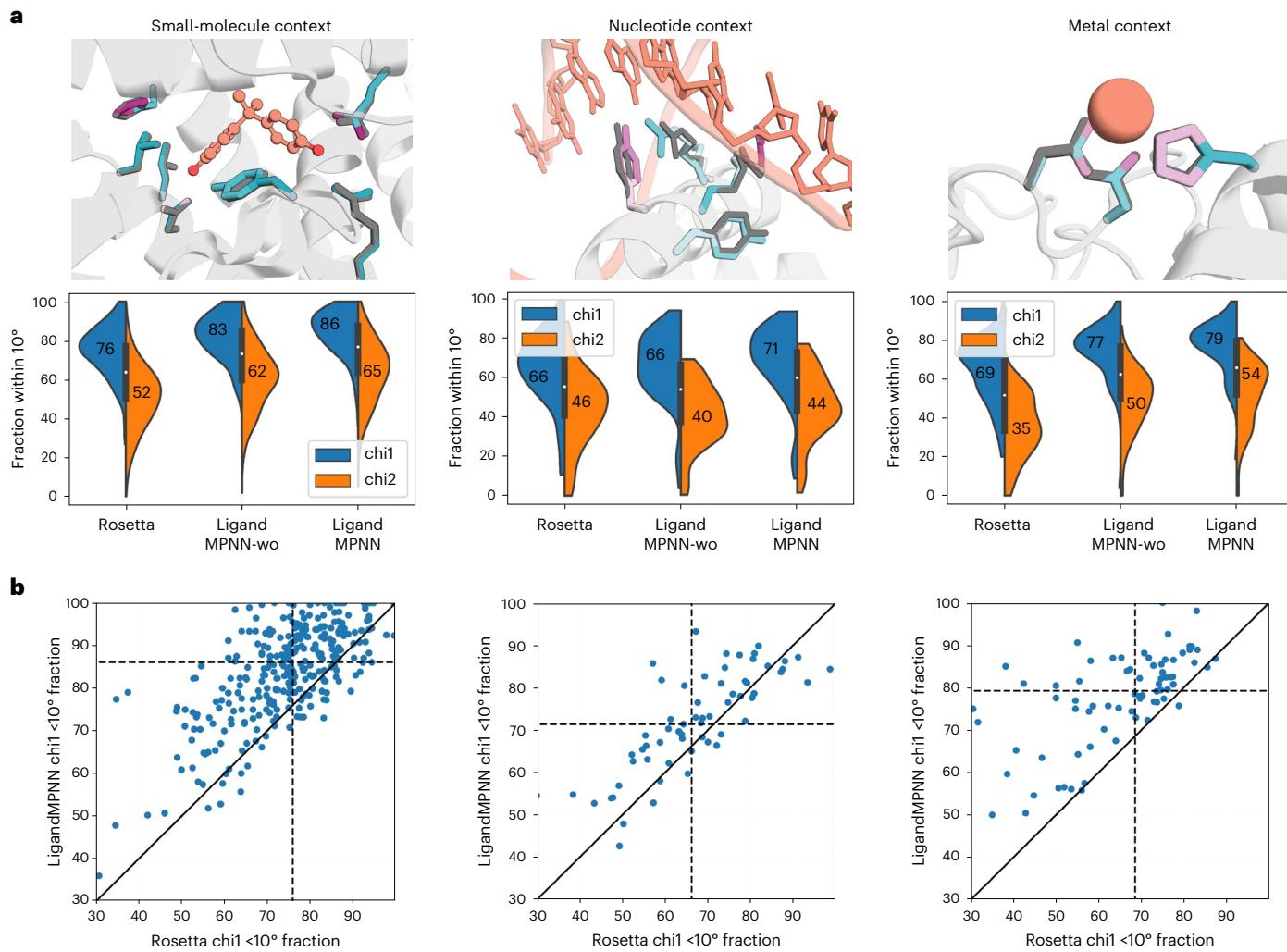


Fig. 3 | Evaluation of LigandMPNN sidechain packing accuracy. **a**, Comparison of crystal sidechain packing (gray) with LigandMPNN sidechain packing (colored sidechains by model confidence: teal is high and purple is low confidence per chi angle) for 2P7G, IBC8 and 1E4M proteins. The context atoms are shown in orange (small molecule, DNA and zinc). LigandMPNN has higher chi1 and chi2 torsion angle recovery (fraction of residues within 10° from native)

than Rosetta and LigandMPNN-wo. **b**, Per-protein comparison of chi1 fraction recovery for LigandMPNN versus Rosetta. One dot represents an average chi1 recovery over 10 sidechain packing samples for one protein for 317 small-molecule-, 67 nucleotide- and 76 metal-containing test proteins. The dashed lines show the mean values.

on the basis of the local geometry of the residue; crystal structure refinement programs introduce some memory of the native sequence into the local backbone. Training with 0.05 Å and 0.2 Å noise instead increased and decreased sequence recovery by about 2%, respectively (Supplementary Fig. 1c; when comparing performance across methods, similar levels of noising must be used). Ablating the protein–ligand and ligand graphs led to a 3% decrease in sequence recovery (Supplementary Fig. 1d). Training on sidechain context atoms only (no small molecules, nucleotides or metals) reduced sequence recovery around small molecules by 3.3% (Supplementary Fig. 1e). Finally, a model trained without chemical element types as input features had much lower sequence recovery near metals (8% difference; Supplementary Fig. 1d) but almost the same sequence recovery near small molecules and nucleic acids, suggesting that the model can to some extent infer chemical element identity from bonded geometry.

We evaluated LigandMPNN sidechain packing performance on the same dataset for residues within 5.0 Å from the context atoms. We generated ten sidechain packing examples with the fixed backbone and fixed ligand context using Rosetta, LigandMPNN and LigandMPNN without ligand context (LigandMPNN-wo in Fig. 3). The median chi1 fraction (within 10° from crystal packing) near small molecules was

76.0% for Rosetta, 83.3% for LigandMPNN-wo and 86.1% for LigandMPNN, near nucleotides 66.2%, 65.6% and 71.4% and near metals 68.6%, 76.7% and 79.3% for the three models, respectively (Fig. 3a). LigandMPNN has a higher chi1 fraction recovery compared with Rosetta on most of the test proteins (Fig. 3b), but only marginally better than LigandMPNN-wo (Supplementary Fig. 3c), suggesting that most of the information about sidechain packing is coming from the protein context rather than from the ligand context, consistent with binding site preorganization. All the models struggle to predict chi3 and chi4 angles correctly. For LigandMPNN, weighted average fractions of correctly predicted chi1, chi2, chi3 and chi4 angles for the small-molecule dataset were 84.0%, 64.0%, 28.3% and 18.7%, for Rosetta 74.5%, 50.5%, 24.1% and 8.1% and for LigandMPNN-wo 81.6%, 60.4%, 26.7% and 17.4% (Supplementary Fig. 3b). The sidechain root-mean-square deviations are similar between the different methods as shown in Supplementary Figs. 4 and 5. Comparing LigandMPNN-wo versus LigandMPNN, the biggest improvements in terms of root-mean-square deviation are obtained for glutamine (Q) in the small-molecule dataset, for arginine (R) in the nucleotide dataset and for histidine (H) in the metal context dataset (Supplementary Fig. 5), consistent with the important roles of interactions of these residues with the corresponding ligands.

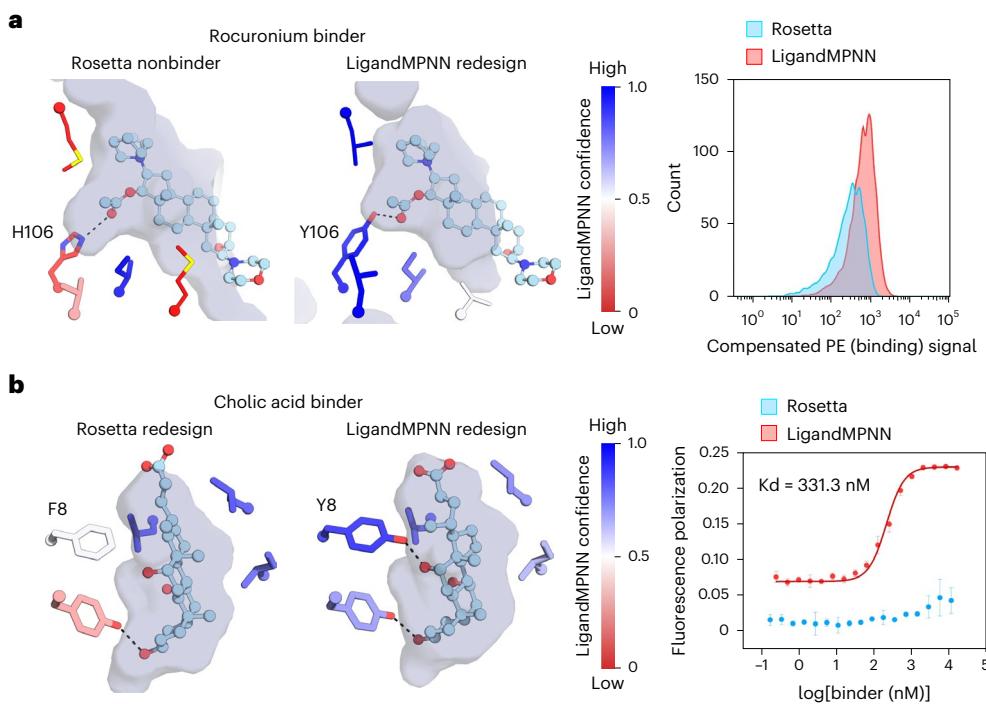


Fig. 4 | Rescue of Rosetta small-molecule binder designs using LigandMPNN. **a,b,** Weak or nonbinding designs made using Rosetta for rocuronium (**a**) and cholic acid (**b**) were redesigned using LigandMPNN. Left: sidechain–ligand interactions before and after redesign. The sidechains are predicted to be considerably more preorganized following redesign as indicated by the LigandMPNN amino-acid probabilities, which are colored from red (0) to blue (1). Sidechain atoms except for carbon are color-coded (O, red; N, blue; S, yellow).

Right: experimental measurement of binding. In **a**, flow cytometry of yeast is shown, with the designs following incubation with 1 μM biotinylated rocuronium and streptavidin phycoerythrin. In **b**, fluorescence polarization measurements of binding to cholic acid–fluorescein isothiocyanate are shown. The error bars show the mean and standard deviations for three LigandMPNN and two Rosetta measurements.

We tested the capability of LigandMPNN to design binding sites for small molecules starting from previously characterized designs generated using Rosetta that either bound weakly or not at all to their intended targets: the muscle relaxant rocuronium, for which no binding was previously observed (Fig. 4a) and the primary bile acid cholic acid (Fig. 4b) for which binding was very weak^{3,4}. LigandMPNN was used to generate sequences around the ligands using the backbone and ligand coordinates as input; these retain and/or introduce new sidechain–ligand hydrogen bonding interactions. LigandMPNN redesigns either rescued binding (Fig. 4a and Supplementary Fig. 6) or improved the binding affinity (Fig. 4b). A further example with cholic acid is described in ref. 4, where, starting from the crystal structure of a previously designed complex, LigandMPNN increased binding affinity 100-fold. As with the many other design successes with LigandMPNN (see below), these results indicate significant generalization beyond the PDB training set: there were no rocuronium-binding protein complex structures in the PDB training set, and the cholic-acid-binding protein in the PDB that is closest to our cholic-acid-binding design (PDB: 6JY3) has a quite different structure (template modeling score 0.59) with a totally different ligand-binding location (Supplementary Fig. 7).

Discussion

The deep-learning-based LigandMPNN is superior to the physically based Rosetta for designing amino acids to interact with nonprotein molecules. It is about 250 times faster (because the expensive Monte Carlo optimization over sidechain identities and compositions is completely bypassed), and the recoveries of native amino-acid identities and conformations around ligands are consistently higher. The method is also easier to use because no expert customizations are required for new ligands (unlike Rosetta and other physically based methods that can require new energy function or force field parameters for

new compounds). At the outset, we were unsure whether the accuracy of ProteinMPNN could extend to protein–ligand systems given the small amount of available training data, but our results suggest that, for the vast majority of ligands, there are sufficient data. Nevertheless, we suggest some care in using LigandMPNN for designing binders to compounds containing elements occurring rarely or not at all in the PDB (in the latter case it is necessary to map to the most closely occurring element). Hybridization of the physically based and deep-learning-based approaches may provide a better solution to the amino-acid and side-chain optimization problems in the low-data regime.

LigandMPNN has already been extensively used for designing interactions of proteins with nucleic acids and small molecules, and these studies provide considerable additional experimental validation of the method. In these studies, LigandMPNN was either used as a drop-in replacement for Rosetta sequence design retaining the backbone relaxation of RosettaFastDesign^{38,42}, or used independently without backbone relaxation. Glasscock et al.² developed a computational method for designing small sequence-specific DNA-binding proteins that recognize specific target sequences through interactions with bases in the major groove that uses LigandMPNN to design the protein–DNA interface. The crystal structure of a DNA-binding protein designed with LigandMPNN recapitulated the design model closely (deposited to the Research Collaboratory for Structural Bioinformatics Protein Data Bank as PDB ID 8TAC). Lee et al.³, An et al.⁴ and Krishna et al.⁵ used LigandMPNN to design small-molecule-binding proteins with scaffolds generated by deep-learning- and Rosetta-based methods. Iterative sequence design with LigandMPNN resulted in nanomolar-to-micromolar binders for the 17 α -hydroxyprogesterone, apixaban and SN-38 with NTF2-family scaffolds³, nanomolar binders for cholic acid, methotrexate and thyroxine⁴ in pseudocyclic scaffolds, and binders for digoxigenin, heme and bilin in

RFdiffusion_allatom-generated scaffolds⁵. In total, more than 100 protein–DNA binding interfaces and protein–small-molecule binding interfaces designed using LigandMPNN have been experimentally demonstrated to bind to their targets, and 5 co-crystal structures have been solved that in each case are very close to the computational design models^{3–5}. This extensive biochemical and structural validation provides strong support for the power of the approach.

As with ProteinMPNN, we anticipate that LigandMPNN will be widely useful in protein design, enabling the creation of a new generation of small-molecule-binding proteins, sensors and enzymes. To this end, we have made the code available via GitHub at <https://github.com/dauparas/LigandMPNN>.

Online content

Any methods, additional references, Nature Portfolio reporting summaries, source data, extended data, supplementary information, acknowledgements, peer review information; details of author contributions and competing interests; and statements of data and code availability are available at <https://doi.org/10.1038/s41592-025-02626-1>.

References

- Yeh, A. H. W. et al. De novo design of luciferases using deep learning. *Nature* **614**, 774–780 (2023).
- Glasscock, C. J. et al. Computational design of sequence-specific DNA-binding proteins. Preprint at bioRxiv <https://doi.org/10.1101/2023.09.20.558720> (2023).
- Lee, G. R. et al. Small-molecule binding and sensing with a designed protein family. Preprint at bioRxiv <https://doi.org/10.1101/2023.11.01.565201> (2023).
- An, L. et al. Binding and sensing diverse small molecules using shape-complementary pseudocycles. *Science* **385**, 276–282 (2024).
- Krishna, R. et al. Generalized biomolecular modeling and design with RoseTTAFold All-Atom. *Science* **384**, eadl2528 (2024).
- Silva, D. A. et al. De novo design of potent and selective mimics of IL-2 and IL-15. *Nature* **565**, 186–191 (2019).
- Cao, L. et al. Design of protein-binding proteins from the target structure alone. *Nature* **605**, 551–560 (2022).
- Wang, J. et al. Scaffolding protein functional sites using deep learning. *Science* **377**, 387–394 (2022).
- Wicky, B. I. M. et al. Hallucinating symmetric protein assemblies. *Science* **378**, 56–61 (2022).
- Praetorius, F. et al. Design of stimulus-responsive two-state hinge proteins. *Science* **381**, 754–760 (2023).
- Marcos, E. & Silva, D. A. Essentials of de novo protein design: methods and applications. *Wiley Interdisc. Rev. Comput. Mol. Sci.* **8**, e1374 (2018).
- Ovchinnikov, S. & Huang, P. S. Structure-based protein design with deep learning. *Curr. Opin. Chem. Biol.* **65**, 136–144 (2021).
- Ferruz, N. et al. From sequence to function through structure: deep learning for protein design. *Comput. Struct. Biotechnol. J.* **21**, 238–250 (2023).
- Kortemme, T. De novo protein design—from new structures to programmable functions. *Cell* **187**, 526–544 (2024).
- Anand, N. & Achim, T. Protein structure and sequence generation with equivariant denoising diffusion probabilistic models. Preprint at <https://arxiv.org/abs/2205.15019> (2022).
- Watson, J. L. et al. De novo design of protein structure and function with RFdiffusion. *Nature* <https://doi.org/10.1038/s41586-023-06415-8> (2023).
- Yim, J. et al. SE(3) diffusion model with application to protein backbone generation. Preprint at <https://arxiv.org/abs/2302.02277> (2023).
- Ingraham, J. B. et al. Illuminating protein space with a programmable generative model. *Nature* **623**, 1070–1078 (2023).
- Wang, C. et al. Proteus: exploring protein structure generation for enhanced designability and efficiency. Preprint at bioRxiv <https://doi.org/10.1101/2024.02.10.579791> (2024).
- Leaver-Fay, A. et al. Scientific benchmarks for guiding macromolecular energy function improvement. *Methods Enzymol.* **523**, 109–143 (2013).
- Ingraham, J., Garg, V., Barzilay, R. & Jaakkola, T. Generative models for graph-based protein design. *Adv. Neural Inf. Process. Syst* **32**, (2019).
- Zhang, Y. et al. ProDCoNN: protein design using a convolutional neural network. *Proteins Struct. Funct. Bioinf.* **88**, 819–829 (2020).
- Leman, J. K. et al. Macromolecular modeling and design in Rosetta: recent methods and frameworks. *Nat. Methods* **17**, 665–680 (2020).
- Qi, Y. & Zhang, J. Z. DenseCPD: improving the accuracy of neural-network-based computational protein sequence design with DenseNet. *J. Chem. Inf. Model.* **60**, 1245–1252 (2020).
- Jing, B., Eismann, S., Suriana, P., Townshend, R. J. L. & Dror, R. Learning from protein structure with geometric vector perceptrons. In *International Conference on Learning Representations* (2020).
- Strokach, A., Becerra, D., Corbi-Verge, C., Perez-Riba, A. & Kim, P. M. Fast and flexible protein design using deep graph neural networks. *Cell Syst.* **11**, 402–411 (2020).
- Anand, N. et al. Protein sequence design with a learned potential. *Nat. Commun.* **13**, 746 (2022).
- Dauparas, J. et al. Robust deep learning-based protein sequence design using ProteinMPNN. *Science* **378**, 49–56 (2022).
- Hsu, C. et al. Learning inverse folding from millions of predicted structures. In *Proc. 39th International Conference on Machine Learning* Vol. 162 (eds Chaudhuri, K. et al.) 8946–8970. (PMLR, 2022).
- Li, A. J. et al. Neural network-derived Potts models for structure-based protein design using backbone atomic coordinates and tertiary motifs. *Protein Sci.* **32**, e4554 (2023).
- Senior, A. W. et al. Improved protein structure prediction using potentials from deep learning. *Nature* **577**, 706–710 (2020).
- Yang, J. et al. Improved protein structure prediction using predicted interresidue orientations. *Proc. Natl Acad. Sci. USA* **117**, 1496–1503 (2020).
- Jumper, J. et al. Highly accurate protein structure prediction with AlphaFold. *Nature* **596**, 583–589 (2021).
- Baek, M. et al. Accurate prediction of protein structures and interactions using a three-track neural network. *Science* **373**, 871–876 (2021).
- Evans, R. et al. Protein complex prediction with AlphaFold-Multimer. Preprint at bioRxiv <https://doi.org/10.1101/2021.10.04.463034> (2021).
- Lin, Z. et al. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science* **379**, 1123–1130 (2023).
- Kuhlman, B. Designing protein structures and complexes with the molecular modeling program Rosetta. *J. Biol. Chem.* **294**, 19436–19443 (2019).
- Maguire, J. B. et al. Perturbing the energy landscape for improved packing during computational protein design. *Proteins Struct. Funct. Bioinf.* **89**, 436–449 (2021).
- Dou, J. et al. De novo design of a fluorescence-activating β-barrel. *Nature* **561**, 485–491 (2018).
- Alford, R. F. et al. The Rosetta all-atom energy function for macromolecular modeling and design. *J. Chem. Theory Comput.* **13**, 3031–3048 (2017).
- Park, H., Zhou, G., Baek, M., Baker, D. & DiMaio, F. Force field optimization guided by small molecule crystal lattice data enables consistent sub-angstrom protein–ligand docking. *J. Chem. Theory Comput.* **17**, 2000–2010 (2021).
- Tyka, M. D. et al. Alternate states of proteins revealed by detailed energy landscape mapping. *J. Mol. Biol.* **405**, 607–618 (2011).

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share

adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

© The Author(s) 2025

Methods

Methods for training LigandMPNN for sequence design

Training data. LigandMPNN was trained on a dataset similar to ProteinMPNN²⁸. We used protein assemblies in the PDB (as of 16 December 2022) determined by X-ray crystallography or cryo-electron microscopy to better than 3.5 Å resolution and with fewer than 6,000 residues. We parsed all residues present in the PDBs except ['HOH', 'NA', 'CL', 'K', 'BR']. Protein sequences were clustered at 30% sequence identity cut-off using mmseqs2 (ref. 43). We held out a nonoverlapping subset of proteins that have small-molecule contexts (a total of 317), nucleotide contexts (a total of 74) and metal contexts (a total of 83).

Optimizer and loss function. For optimization, we used Adam with beta1 of 0.9, beta2 of 0.98 and epsilon of 1e-9, the same as for ProteinMPNN. Models were trained with a batch size of 6,000 tokens, automatic mixed precision and gradient checkpointing on a single NVIDIA A100 graphics processing unit for 300,000 optimizer steps. We used categorical cross entropy for the loss function following the ProteinMPNN paper²⁸.

Input featurization and model architecture. We used the same input features as in the ProteinMPNN paper for the protein part. For the atomic context input features, we used one-hot-encoded chemical element types as node features for the ligand graph and the radial basis function-encoded distances between the context atoms as edges for the ligand graph. To encode the interaction between protein-context atoms, we used distances between N, Cα, C, O and virtual Cβ atoms and context atoms. In addition, we added angle-based sin/cos features describing context atoms in the frame of N–Cα–C atoms.

We used the same MPNN architecture as used in ProteinMPNN for the encoder, decoder and protein–ligand encoder blocks. Encoder and decoder blocks work on protein nodes and edges, that is, mapping vertices [N] and edges [N, K] to updated vertices [N] and edges [N, K] where N is the number of residues and K is the number of direct neighbors per residue. We choose M context atoms per residue resulting in [N, M] protein–atom interactions. The ligand graph blocks map vertices of size [N, M] and edges of size [N, M, M] (fully connected context atoms) to updated vertices [N, M]. The updated [N, M] representation is used in the protein–ligand graph to map vertices [N] and edges [N, M] into updated vertices [N]. For more details, refer to the LigandMPNN code.

Model algorithms. We provide a list of algorithms and model layers used by the LigandMPNN model. The model is based on the autoregressive encoder-decoder architecture. Algorithm 10 describes how the input features such as protein atom coordinates (X), ligand coordinates (Y), ligand mask (Y_m), and ligand atom types (Y_t) are converted into the input features. Protein and ligand geometric features are encoded using the algorithm 11, and it returns final protein node and edge features. Finally, algorithm 12 decodes protein sequence by predicting log probabilities for all amino acids. During the inference, we sample from these probabilities with some temperate (T) (algorithm 13) and iteratively run algorithm 12 to populate the designed sequence (S).

Notation:

X ∈ ℝ^{L×4×3}- protein backbone coordinates for N, Cα, C and O atoms with L residues

Y ∈ ℝ^{L×M×3}- coordinates of the closest M ligand atoms from the virtual Cβ atom in the protein

Y_m ∈ ℝ^{L×M}- ligand atom mask

Y_t ∈ ℝ^{L×M}- ligand atom type

Algorithm 1. Linear layer

def Linear(x ∈ ℝⁿ; W ∈ ℝ^{m×n}, b ∈ ℝ^m):

1: x ← Wx+b, x ∈ ℝ^m

2: **return** x

Algorithm 2. Non-linear layer⁴⁴

def GELU(x ∈ ℝⁿ):

1: x ← 0.5·x·(1+tanh(2/π·(x + 0.044715·x³))), x ∈ ℝⁿ

2: **return** x

Algorithm 3. Normalization layer

def LayerNorm(x ∈ ℝⁿ; γ ∈ ℝⁿ, β ∈ ℝⁿ):

1: μ = E[x]=(x₁ + x₂ + ...+x_n)/n, μ ∈ ℝⁿ

2: σ² = E[(x-μ)²], σ² ∈ ℝⁿ

3: x ← γ·(x-μ)/σ + β, x ∈ ℝⁿ

4: **return** x

Algorithm 4. Dropout layer

def Dropout(x ∈ ℝⁿ; p ∈ ℝ, training: bool):

1: if training:

2: mask = Binomial[1-p](x.shape), mask ∈ ℝⁿ

3: x ← mask·x/(1-p), x ∈ ℝⁿ

4: **return** x

5: else:

6: **return** x

Algorithm 5. Position wise feed-forward

def PositionWiseFeedForward (v_i ∈ ℝⁿ; n = 128, m = 512):

1: v_i ← Linear[n,m](v_i), v_i ∈ ℝ^m

2: v_i ← GELU(v_i), v_i ∈ ℝ^m

3: v_i ← Linear[m,n](v_i), v_i ∈ ℝⁿ

4: **return** v_i

Algorithm 6. Positional encoding layer

def PositionalEncodings(offset ∈ ℝ^{L×K}, mask ∈ ℝ^{L×K}; n = 16, max_offset = 32):

#offset - protein residue to residue distances for all chains

#mask - mask if two residues are from the same chain

#n - number of dimensions to embed the offset to

#max_offset - maximum distance between two residues

1: d = mask·clip[0, 2·max_offset](offset + max_offset), d ∈ ℝ^{L×L}

2: f = (1-mask)·(2·max_offset + 1), f ∈ ℝ^{L×L}

3: g = d + f, g ∈ ℝ^{L×L}

4: g_one_hot = one_hot[2·max_offset + 2](g), g_one_hot ∈ ℝ^{L×L×2·max_offset+2}

5: e ← Linear[2·max_offset + 2, n](g_one_hot), e ∈ ℝ^{L×L×n}

6: **return** e

Algorithm 7. Encoder Layer

def EncLayer(v ∈ ℝ^{L×n}, e ∈ ℝ^{L×K×n}, e_idx ∈ ℝ^{L×K}; n = 128, m = 128, p = 0.1, s = 30.0):

#v - vertex embedding for L residues

#e - edge embedding for L residues with K neighbors per residue

#e_idx - integers specifying protein residue neighbor positions

#n - input dimension

#m - hidden dimension

#p - dropout probability

#s - scaling factor

1: q_{ij} = concatenate[e_idx_{ij}](v_i, v_j, e_{ij}), q_{ij} ∈ ℝ^{L×K×3·n}, q_{ij} ∈ ℝ^{3·n},

2: q_{ij} ← GELU{Linear[3n,m](q_{ij})}, q_{ij} ∈ ℝ^m,

3: q_{ij} ← GELU{Linear[m,m](q_{ij})}, q_{ij} ∈ ℝ^m,

4: q_{ij} ← Linear[m,m](q_{ij}), q_{ij} ∈ ℝ^m,

5: dh_i ← Σ_j q_{ij}/s, dh_i ∈ ℝ^m,

6: v_i ← LayerNorm{v_i+Dropout[p](dh_i)}, v_i ∈ ℝ^m,

7: q_{ij} = concatenate[e_idx_{ij}](v_i, v_j, e_{ij}), q_{ij} ∈ ℝ^{L×K×3·n}, q_{ij} ∈ ℝ^{3·n},

8: q_{ij} ← GELU{Linear[3n,m](q_{ij})}, q_{ij} ∈ ℝ^m,

9: q_{ij} ← GELU{Linear[m,m](q_{ij})}, q_{ij} ∈ ℝ^m,

10: q_{ij} ← Linear[m,m](q_{ij}), q_{ij} ∈ ℝ^m,

11: e_{ij} ← LayerNorm{e_{ij}+Dropout[p](q_{ij})}, v_i ∈ ℝ^m,

12: **return** v, e

Algorithm 8. Decoder Layer

```

def DecLayer( $v \in \mathbf{R}^{L \times n}$ ,  $e \in \mathbf{R}^{L \times K \times 2n}$ ;  $n = 128, m = 128, p = 0.1, s = 30.0$ ):
    #v - vertex embedding for L residues
    #e - edge embedding for L residues with K neighbors
    #n - input dimension
    #m - hidden dimension
    #p - dropout probability
    #s - scaling factor
    1:  $q_{ij} = \text{concatenate}(v_i, e_{ij})$ ,  $q \in \mathbf{R}^{L \times K \times 3 \cdot n}$ ,  $q_{ij} \in \mathbf{R}^{3 \cdot n}$ 
    2:  $q_{ij} \leftarrow \text{GELU}\{\text{Linear}[3n, m](q_{ij})\}$ ,  $q_{ij} \in \mathbf{R}^m$ ,
    3:  $q_{ij} \leftarrow \text{GELU}\{\text{Linear}[m, m](q_{ij})\}$ ,  $q_{ij} \in \mathbf{R}^m$ ,
    4:  $q_{ij} \leftarrow \text{Linear}[m, m](q_{ij})$ ,  $q_{ij} \in \mathbf{R}^m$ ,
    5:  $dh_i \leftarrow \sum_j q_{ij}/s$ ,  $dh_i \in \mathbf{R}^m$ ,
    6:  $v_i \leftarrow \text{LayerNorm}[v_i + \text{Dropout}[p](dh_i)]$ ,  $v_i \in \mathbf{R}^m$ ,
    7: return v

```

Algorithm 9. Context Decoder Layer

```

def DecLayerJ( $v \in \mathbf{R}^{L \times M \times n}$ ,  $e \in \mathbf{R}^{L \times M \times M \times 2n}$ ;  $n = 128, m = 128, p = 0.1, s = 30.0$ ):
    #v - vertex embedding for L residues with M atoms per residue
    #e - edge for L residues with M atoms and M neighbors per atom
    #n - input dimension
    #m - hidden dimension
    #p - dropout probability
    #s - scaling factor
    1:  $q_{ijk} = \text{concatenate}(v_{ij}, e_{ijk})$ ,  $q \in \mathbf{R}^{L \times M \times M \times 3 \cdot n}$ ,  $q_{ijk} \in \mathbf{R}^{3 \cdot n}$ 
    2:  $q_{ijk} \leftarrow \text{GELU}\{\text{Linear}[3n, m](q_{ijk})\}$ ,  $q_{ijk} \in \mathbf{R}^m$ ,
    3:  $q_{ijk} \leftarrow \text{GELU}\{\text{Linear}[m, m](q_{ijk})\}$ ,  $q_{ijk} \in \mathbf{R}^m$ ,
    4:  $q_{ijk} \leftarrow \text{Linear}[m, m](q_{ijk})$ ,  $q_{ijk} \in \mathbf{R}^m$ ,
    5:  $dh_{ij} \leftarrow \sum_k q_{ijk}/s$ ,  $dh_{ij} \in \mathbf{R}^m$ ,
    6:  $v_{ij} \leftarrow \text{LayerNorm}[v_{ij} + \text{Dropout}[p](dh_{ij})]$ ,  $v_i \in \mathbf{R}^m$ ,
    7: return v

```

Algorithm 10. Protein and ligand featurization

```

def ProteinFeaturesLigand( $Y \in \mathbf{R}^{L \times M \times 3}$ ,  $Y_m \in \mathbf{R}^{L \times M}$ ,  $Y_t \in \mathbf{R}^{L \times M}$ ,  $X \in \mathbf{R}^{L \times 4 \times 3}$ ,
 $R\_idx \in \mathbf{R}^L$ ,  $\text{chain\_labels} \in \mathbf{R}^L$ ;  $\text{noise\_level} = 0.1, K = 32, m = 128, r = 16$ ):
    #Y, Y_m, Y_t - ligand atom coordinates, mask, and chemical
    #atom type
    #X - protein coordinates for N, Cα, C, O atoms in this order
    #R_idx - protein residue indices
    #chain labels - integer labels for protein chains
    #noise_level - standard deviation of Gaussian noise
    #K - number of nearest Cα neighbors for protein
    #m - hidden dimension size
    #r - radial basis function number
    1:  $X \leftarrow X + \text{noise\_level} \cdot \text{GaussianNoise}(X.\text{shape})$ ,  $X \in \mathbf{R}^{L \times 4 \times 3}$ ,
    2:  $Y \leftarrow Y + \text{noise\_level} \cdot \text{GaussianNoise}(Y.\text{shape})$ ,  $Y \in \mathbf{R}^{L \times M \times 3}$ ,
    3:  $C\beta = -0.5827 \cdot [(C\alpha-N)^*(C-C\alpha)] + 0.5680 \cdot (C\alpha-N) -$ 
         $0.5407 \cdot (C-C\alpha) + C\alpha, N, C\alpha, C, C\beta \in \mathbf{R}^{L \times 3}$ ,
    4:  $e\_idx = \text{top\_k}[K](||C\alpha_i - C\alpha_j||_2)$ ,  $e\_idx \in \mathbf{R}^{L \times K}$ ,
    5:  $rbf = []$ 
    6: for a in [N, Cα, C, O Cβ]:
        7: for b in [N, Cα, C, O Cβ]:
            8:  $rbf\_tmp = rbf\_f\{\text{get\_edges}[e\_idx](||a_i - b_j||_2)\}$ ,  $rbf\_tmp \in \mathbf{R}^{L \times K \times r}$ ,
            9:  $rbf.append(rbf\_tmp)$ 
        10:  $rbf \leftarrow \text{concatenate}(rbf)$ ,  $rbf \in \mathbf{R}^{L \times K \times 25 \cdot r}$ ,
        11:  $\text{offset} = \text{get\_edges}[e\_idx](R\_idx, R\_idx_j)$ ,  $\text{offset} \in \mathbf{R}^{L \times K}$ ,
        12:  $\text{offset\_m} = \text{get\_edges}[e\_idx](\text{chain\_labels}_i, \text{chain\_labels}_j == 0)$ ,
             $\text{offset\_m} \in \mathbf{R}^{L \times K}$ ,
        13:  $\text{pos\_enc} = \text{PositionalEncodings}(\text{offset}, \text{offset\_m})$ ,  $\text{pos\_enc} \in \mathbf{R}^{L \times K \times r}$ 
        14:  $e \leftarrow \text{LayerNorm}\{\text{Linear}[r + 25 \cdot r, m](\text{concat}[\text{pos\_enc}, rbf])\}$ ,  $e \in \mathbf{R}^{L \times K \times m}$ 
        15:  $Y\_t\_g = \text{chemical\_group}(Y\_t)$ ,  $Y\_t\_g \in \mathbf{R}^{L \times M}$ 
        16:  $Y\_t\_p = \text{chemical\_period}(Y\_t)$ ,  $Y\_t\_p \in \mathbf{R}^{L \times M}$ 
        17:  $Y\_t\_1hot = \text{Linear}[64, 147](\text{onehot}[\text{concat}(Y\_t, Y\_t\_g, Y\_t\_p)])$ ,
             $Y\_t\_1hot \in \mathbf{R}^{L \times M \times 64}$ 
        18:  $rbf\_N\_Y = rbf\_f\{||N-Y||_2\}$ ,  $rbf\_N\_Y \in \mathbf{R}^{L \times M \times r}$ 

```

```

19:  $rbf\_C\alpha\_Y = rbf\_f\{||C\alpha-Y||_2\}$ ,  $rbf\_C\alpha\_Y \in \mathbf{R}^{L \times M \times r}$ 
20:  $rbf\_C\_Y = rbf\_f\{||C-Y||_2\}$ ,  $rbf\_C\_Y \in \mathbf{R}^{L \times M \times r}$ 
21:  $rbf\_O\_Y = rbf\_f\{||O-Y||_2\}$ ,  $rbf\_O\_Y \in \mathbf{R}^{L \times M \times r}$ 
22:  $rbf\_C\beta\_Y = rbf\_f\{||C\beta-Y||_2\}$ ,  $rbf\_C\beta\_Y \in \mathbf{R}^{L \times M \times r}$ 
23:  $rbf\_Y = \text{concat}(rbf\_N\_Y, rbf\_C\alpha\_Y, rbf\_C\_Y, rbf\_O\_Y, rbf\_C\beta\_Y)$ ,
     $rbf\_Y \in \mathbf{R}^{L \times M \times 5 \cdot r}$ 
24:  $\text{angles\_Y} = \text{make\_angle\_features}(N, C\alpha, C, Y)$ ,  $\text{angles\_Y} \in \mathbf{R}^{L \times M \times 4}$ 
25:  $v = \text{concat}(rbf\_Y, Y\_t\_1hot, \text{angles\_Y})$ ,  $v \in \mathbf{R}^{L \times M \times 4}$ 
26:  $v \leftarrow \text{LayerNorm}\{\text{Linear}[5 \cdot r + 64 + 4, m](v)\}$ ,  $v \in \mathbf{R}^{L \times M \times m}$ 
27:  $Y\_edges = rbf\_f\{||Y_i - Y_j||_2\}$ ,  $Y\_edges \in \mathbf{R}^{L \times M \times M \times r}$ 
28:  $Y\_edges = \text{LayerNorm}\{\text{Linear}[r, m](Y\_edges)\}$ ,
     $Y\_edges \in \mathbf{R}^{L \times M \times M \times r}$ 
29:  $Y\_nodes = \text{LayerNorm}\{\text{Linear}[147, m](\text{onehot}[\text{concat}(Y\_t, Y\_t\_g, Y\_t\_p)])\}$ ,
     $Y\_nodes \in \mathbf{R}^{L \times M \times m}$ 
30: return v, e, e_idx, Y_nodes, Y_edges

```

Algorithm 11. LigandMPNN encode function

```

def LigandMPNN_encode( $Y \in \mathbf{R}^{L \times M \times 3}$ ,  $Y_m \in \mathbf{R}^{L \times M}$ ,  $Y_t \in \mathbf{R}^{L \times M}$ ,
 $X \in \mathbf{R}^{L \times 4 \times 3}$ ,  $R\_idx \in \mathbf{R}^L$ ,  $\text{chain\_labels} \in \mathbf{R}^L$ ;  $\text{num\_layers} = 3, c\_num\_layers = 2, m = 128$ ):
    1:  $v\_y, e, e\_idx, Y\_nodes, Y\_edges = \text{ProteinFeaturesLigand}(Y, Y_m, Y_t, X, R\_idx, chain\_labels)$ 
    2:  $v\_y = \text{Linear}[m, m](v\_y)$ ,  $v\_y \in \mathbf{R}^{L \times m}$ ,
    3:  $v = \text{zeros}(L, m)$ ,  $v \in \mathbf{R}^{L \times m}$ ,
    4: for i in range(num_layers):
        5:  $v, e \leftarrow \text{EncLayer}(v, e, e\_idx)$ ,  $v \in \mathbf{R}^{L \times m}$ ,  $e \in \mathbf{R}^{L \times K \times m}$ 
        6:  $v\_c = \text{Linear}[m, m](v)$ ,  $v\_c \in \mathbf{R}^{L \times m}$ ,
        7:  $Y\_m\_edges = Y\_m\_i \cdot Y\_m\_j$ ,  $Y\_edges \in \mathbf{R}^{L \times M \times M}$ ,
        8:  $Y\_nodes = \text{Linear}[m, m](Y\_nodes)$ ,  $Y\_nodes \in \mathbf{R}^{L \times M \times m}$ ,
        9:  $Y\_edges = \text{Linear}[m, m](Y\_edges)$ ,  $Y\_edges \in \mathbf{R}^{L \times M \times M \times m}$ ,
        10: for i in range(c_num_layers):
            11:  $Y\_nodes \leftarrow \text{DecLayerJ}(Y\_nodes, Y\_edges, Y\_m, Y\_m\_edges)$ 
                #atom graph
            12:  $Y\_nodes\_c = \text{concat}(v\_y, Y\_nodes)$ 
            13:  $v\_c \leftarrow \text{DecLayer}(v\_c, Y\_nodes\_c, \text{mask}, Y\_m)$  #protein graph
            14:  $v\_c \leftarrow \text{Linear}[m, m](v\_c)$ 
            14:  $v \leftarrow v + \text{LayerNorm}[\text{Dropout}[p]](v\_c)$ 
        15: return v, e, e_idx

```

Algorithm 12. LigandMPNN decode function

```

def LigandMPNN_decode( $S \in \mathbf{R}^L$ ,  $Y_m \in \mathbf{R}^{L \times M}$ ,  $Y_t \in \mathbf{R}^{L \times M}$ ,  $X \in \mathbf{R}^{L \times 4 \times 3}$ ,  $R\_idx \in \mathbf{R}^L$ ,
 $\text{chain\_labels} \in \mathbf{R}^L$ ,  $\text{decoding\_order} \in \mathbf{R}^L$ ;  $\text{num\_layers} = 3, m = 128$ ):
    1:  $h\_V, e, e\_idx = \text{LigandMPNN\_encode}(Y, Y_m, Y_t, X, R\_idx, chain\_labels)$ 
    2:  $\text{causal\_mask} = \text{upper\_triangular}[\text{decoding\_order}](L, L)$ 
    3:  $h\_S = \text{Linear}[21, m](\text{onehot}(S))$ ,  $h\_S \in \mathbf{R}^{L \times m}$ ,
    4:  $h\_ES = \text{concat}(h\_S, e, e\_idx)$ ,  $h\_ES \in \mathbf{R}^{L \times K \times 2m}$ ,
    5:  $h\_EX\_encoder = \text{concat}(\text{zeros}(h\_S), e, e\_idx)$ ,  $h\_EX\_encoder \in \mathbf{R}^{L \times K \times 2m}$ ,
    6:  $h\_EXV\_encoder = \text{concat}(h\_V, h\_EX\_encoder, e\_idx)$ ,  $h\_EXV\_encoder \in \mathbf{R}^{L \times K \times 3m}$ ,
    7:  $h\_EXV\_encoder\_fw = (1 - \text{causal\_mask}) \cdot h\_EXV\_encoder$ 
    8: for i in range(num_layers):
        9:  $h\_ESV = \text{concat}(h\_V, h\_ES, e\_idx)$ 
        10:  $h\_ESV \leftarrow \text{causal\_mask} \cdot h\_ESV + h\_EXV\_encoder\_fw$ 
        11:  $h\_V \leftarrow \text{DecLayer}(h\_V, h\_ESV)$ 
        12:  $\text{logits} = \text{Linear}[m, 21](h\_V)$ ,  $\text{logits} \in \mathbf{R}^{L \times 21}$ ,
        13:  $\text{log\_probs} = \text{log\_softmax}(\text{logits})$ 
        14: return logits, log_probs

```

Algorithm 13. Amino-acid sampling with temperature

```

def sampling(logits  $\in \mathbf{R}^{21}$ ,  $T \in \mathbf{R}$ , bias  $\in \mathbf{R}^{21}$ ):
    1:  $p = \text{softmax}((\text{logits} + \text{bias}) / T)$ 
    2:  $S = \text{categorical\_sample}(p)$ 
    3: return S

```

Algorithm 14. Outline of LigandMPNN sidechain decode function

```

def LigandMPNN_sc_decode(Y_m ∈ ℝL×M, Y_t ∈ ℝL×M, X ∈ ℝL×4×3,
R_idx ∈ ℝL, chain_labels ∈ ℝL, decoding_order ∈ ℝL; num_layers=3,
m = 128):
1: h_V_enc, h_E_enc, e_idx = LigandMPNN_encode(Y, Y_m, Y_t, X,
R_idx, chain_labels)
2: h_V_dec, h_E_dec = LigandMPNN_encode(Y, Y_m, Y_t, X, R_idx,
chain_labels)
3: causal_mask = upper_triangular[decoding_order](L,L)
4: h_EV_encoder = concat(h_V_enc, h_E_enc, e_idx)
5: h_E_encoder_fw = (1-causal_mask)·h_EV_encoder
6: h_EV_decoder = concat(h_V_dec, h_E_dec, e_idx)
7: h_V = h_V_enc
8: for i in range(num_layers):
9:   h_EV = concat(h_V, h_E_decoder, e_idx)
10:  h_ECV ← causal_mask·h_EV + h_E_encoder_fw
11:  h_V ← DecLayer(h_V, h_ECV)
12:  torsions = Linear[m,4-3-3](h_V).reshape(L,4,3,3), torsions
    ∈ ℝL×4×3×3,
13:  mean = torsions[...,0], mean ∈ ℝL×4×3,
14:  concentration = 0.1 + softplus(torsions[...,1]), concentration
    ∈ ℝL×4×3
15:  mix_logits = torsions[...,2], mix_logits ∈ ℝL×4×3
16:  predicted_distribution = VonMisesMixture(mean, concentration,
    mix_logits)
17:  return predicted_distribution

```

ProteinMPNN and LigandMPNN share the idea of using autoregressive sequence decoding with a sparse residue graph with ref. 21. However, there are many differences between the models. First, ProteinMPNN is trained on biological protein assemblies, and LigandMPNN on the biological protein assemblies with small molecules, nucleotides, metals and other atoms in the PDB, whereas ref. 21 was trained on single chains only. Second, we wanted our models to work well with novel protein backbones as opposed to crystal backbones, and for this reason, we added Gaussian noise to all the protein and other atom coordinates to blur out fine-scale details that would not be available during the design. Furthermore, we innovated by using a random autoregressive decoding scheme that fits more naturally protein sequences as opposed to left-to-right decoding used in language models and ref. 21. Also, we simplified input geometric features by keeping only distances between N, Ca, C, O and inferred Cβ atoms and added positional encodings that allowed us to design multiple protein chains at the same time, as opposed to using backbone local angles as in ref. 21. Both ProteinMPNN and LigandMPNN can design symmetric and multistate proteins by choosing an appropriate decoding order and averaging out predicted probabilities. Also, we added expressivity to our MPNN encoder layers, allowing both graph nodes and edges to be updated. LigandMPNN further builds on top of ProteinMPNN by incorporating local atomic context into the protein residue local environment using invariant features. We pass messages between protein residues and context atoms to encode possible sequence combinations. Finally, LigandMPNN can also predict with uncertainty multiple sidechain packing combinations of a newly designed sequence near nucleotides, metals and small molecules, which can help designers to choose sequences that make desired interactions with the ligand of interest. LigandMPNN can also take sidechain conformations as an input, which allows the design sequence to stabilize given ligand and selected protein sidechains.

Algorithms 1, 2, 3, 4 and 5 are commonly used in many machine learning models. Algorithms 6, 7, 8 and 13 were used in the ProteinMPNN model. Algorithms 9, 10, 11, 12 and 14 are novel and specific to LigandMPNN.

Reporting summary

Further information on research design is available in the Nature Portfolio Reporting Summary linked to this article.

Data availability

All data are available in the Article or its Supplementary Information. PDB structures used for training were obtained from RCSB. The following PDB IDs were used in the Article: 8VEI, 8BEJ, 8VEZ, 8VFQ, 8TAC, 6JY3, 2P7G, 1BC8 and 1E4M. (<https://www.rcsb.org/docs/programmatic-access/file-download-services>). Source data are provided with this paper.

Code availability

The LigandMPNN code is available via GitHub at <https://github.com/dauparas/LigandMPNN>. The neural network was developed with PyTorch 1.11.0, cuda 11.1, NumPy v1.21.5, Matplotlib v3.5.1 and Python v3.9.12. MMseqs2 version 13-45111+ds-2 was used to cluster PDB chains, and mmcif version 0.84 (<https://pypi.org/project/mmcif/>) and rdkit version 2022.03.2 were used to parse PDB files. The flow cytometry data were analyzed using the software FlowJo v10.9.0.

References

43. Steinegger, M. & Söding, J. MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nat. Biotechnol.* **35**, 1026–1028 (2017).
44. Hendrycks, D. & Gimpel, K. Gaussian error linear units (GELUs). Preprint at <https://arxiv.org/abs/1606.08415> (2016).

Acknowledgements

We thank S. Pellock, Y. Kipnis, J. Wenckstern, A. Goncharenko, N. Hanikel, W. Ahern, P. Sturmels, R. Krishna, D. Juergens, R. McHugh, P. Kim and I. Kalvet for helpful discussions. This research was supported by the Department of the Defense, Defense Threat Reduction Agency grant (grant no. HDTRA1-21-1-0007 to I.A.); National Science Foundation (grant no. CHE-2226466 for R.P.); Spark Therapeutics (Computational Design of a Half Size Functional ABCA4 to I.A.); The Audacious Project at the Institute for Protein Design (to L.A. and C.G.); Microsoft (to J.D. and I.A.); the Washington Research Foundation, Innovation Fellows Program (to G.R.L.); the Washington Research Foundation and Translational Research Fund (to L.A.); a Washington Research Foundation Fellowship (to C.G.); Howard Hughes Medical Institute (G.R.L., I.A. and D.B.); National Institute of Allergy and Infectious Diseases (NIAID) (contract nos. HHSN272201700059C and 75N93022C00036 to I.A.); the Open Philanthropy Project Improving Protein Design Fund (to J.D. and G.R.L.); and the Bill & Melinda Gates Foundation Grant INV-037981 (to G.R.L.).

Author contributions

Conceptualization: J.D., G.R.L., L.A. and I.A.; methodology: J.D., G.R.L., L.A., I.A., R.P. and C.G.; software: J.D. and I.A.; validation: G.R.L., L.A., R.P. and C.G.; formal analysis: J.D. and G.R.L.; resources: J.D. and D.B.; data curation: I.A., J.D., G.R.L., L.A. and R.P.; writing—original draft: J.D. and D.B.; writing—review and editing: J.D. and D.B.; visualization: J.D., G.R.L. and L.A.; supervision: D.B.; funding acquisition: J.D. and D.B.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at <https://doi.org/10.1038/s41592-025-02626-1>.

Correspondence and requests for materials should be addressed to David Baker.

Peer review information *Nature Methods* thanks Claus Wilke and the other, anonymous, reviewer(s) for their contribution to the peer review

of this work. Primary Handling Editor: Arunima Singh, in collaboration with the *Nature Methods* team.

Reprints and permissions information is available at www.nature.com/reprints.

Reporting Summary

Nature Portfolio wishes to improve the reproducibility of the work that we publish. This form provides structure for consistency and transparency in reporting. For further information on Nature Portfolio policies, see our [Editorial Policies](#) and the [Editorial Policy Checklist](#).

Statistics

For all statistical analyses, confirm that the following items are present in the figure legend, table legend, main text, or Methods section.

n/a Confirmed

- The exact sample size (n) for each experimental group/condition, given as a discrete number and unit of measurement
- A statement on whether measurements were taken from distinct samples or whether the same sample was measured repeatedly
- The statistical test(s) used AND whether they are one- or two-sided
Only common tests should be described solely by name; describe more complex techniques in the Methods section.
- A description of all covariates tested
- A description of any assumptions or corrections, such as tests of normality and adjustment for multiple comparisons
- A full description of the statistical parameters including central tendency (e.g. means) or other basic estimates (e.g. regression coefficient) AND variation (e.g. standard deviation) or associated estimates of uncertainty (e.g. confidence intervals)
- For null hypothesis testing, the test statistic (e.g. F , t , r) with confidence intervals, effect sizes, degrees of freedom and P value noted
Give P values as exact values whenever suitable.
- For Bayesian analysis, information on the choice of priors and Markov chain Monte Carlo settings
- For hierarchical and complex designs, identification of the appropriate level for tests and full reporting of outcomes
- Estimates of effect sizes (e.g. Cohen's d , Pearson's r), indicating how they were calculated

Our web collection on [statistics for biologists](#) contains articles on many of the points above.

Software and code

Policy information about [availability of computer code](#)

Data collection

The neural network was developed with PyTorch 1.11.0 (<https://pytorch.org/get-started/locally/>), cuda 11.1 (<https://developer.nvidia.com/cuda-11.1.0-download-archive>), NumPy v1.21.5 (<https://github.com/numpy/numpy>), Matplotlib v3.5.1 (<https://github.com/matplotlib/matplotlib>), Python v3.9.12 (<https://www.python.org/>).

MMseqs2 version 13-45111+ds-2 (<https://github.com/soedinglab/MMseqs2>) was used to cluster PDB chains, and mmcif version 0.84 (<https://pypi.org/project/mmcif/>), rdkit version 2022.03.2 (<https://github.com/rdkit/rdkit>) was used to parse PDB files.

Data analysis

Data analysis used Python v3.9.12 (<https://www.python.org/>), Matplotlib v3.5.1 (<https://github.com/matplotlib/matplotlib>), NumPy v1.21.5 (<https://github.com/numpy/numpy>). Structure visualizations were created in Pymol v2.3.5 (<https://github.com/schrodinger/pymol-open-source>). Flow cytometry data was analyzed using the software FlowJo v10.9.0.

For manuscripts utilizing custom algorithms or software that are central to the research but not yet described in published literature, software must be made available to editors and reviewers. We strongly encourage code deposition in a community repository (e.g. GitHub). See the Nature Portfolio [guidelines for submitting code & software](#) for further information.

Data

Policy information about [availability of data](#)

All manuscripts must include a [data availability statement](#). This statement should provide the following information, where applicable:

- Accession codes, unique identifiers, or web links for publicly available datasets
- A description of any restrictions on data availability
- For clinical datasets or third party data, please ensure that the statement adheres to our [policy](#)

All input data are freely available from public sources.

PDB structures used for training were obtained from RCSB (<https://www.rcsb.org/docs/programmatic-access/file-download-services>). The PDB ids used in the paper: 8VEI, 8BEJ, 8VEZ, 8VFQ, 8TAC, 6JY3, 2P7G, 1BC8, 1E4M.

Human research participants

Policy information about [studies involving human research participants](#) and [Sex and Gender in Research](#).

Reporting on sex and gender

N/A

Population characteristics

N/A

Recruitment

N/A

Ethics oversight

N/A

Note that full information on the approval of the study protocol must also be provided in the manuscript.

Field-specific reporting

Please select the one below that is the best fit for your research. If you are not sure, read the appropriate sections before making your selection.

Life sciences Behavioural & social sciences Ecological, evolutionary & environmental sciences

For a reference copy of the document with all sections, see nature.com/documents/nr-reporting-summary-flat.pdf

Life sciences study design

All studies must disclose on these points even when the disclosure is negative.

Sample size

No sample size was chosen; the method was evaluated on PDB chains not in the training set (subject to the exclusions noted below).

Data exclusions

The PDB dataset was filtered removing all entries with lower than 3.5 angstrom resolution, chain with too few resolved residues, biological units with more than 6000 residues. This set was also redundancy reduced by clustering chains using 30% sequence identity, 80% coverage using MMseqs2 version 13-45111+ds-2 (<https://github.com/soedinglab/MMseqs2>).

No sample was excluded from the experimental data analysis.

Replication

The biotinylated rocuronium binding experiment using yeast cell surface display and flow cytometry was replicated twice to validate the binding signal. The fluorescence polarization experiments were performed independently twice and all replications were successful.

Randomization

Randomization was not needed for the binding experiments.

Blinding

Blinding was not needed for the binding experiments.

Reporting for specific materials, systems and methods

We require information from authors about some types of materials, experimental systems and methods used in many studies. Here, indicate whether each material, system or method listed is relevant to your study. If you are not sure if a list item applies to your research, read the appropriate section before selecting a response.

Materials & experimental systems

n/a	Involved in the study
<input type="checkbox"/>	<input checked="" type="checkbox"/> Antibodies
<input checked="" type="checkbox"/>	<input type="checkbox"/> Eukaryotic cell lines
<input checked="" type="checkbox"/>	<input type="checkbox"/> Palaeontology and archaeology
<input checked="" type="checkbox"/>	<input type="checkbox"/> Animals and other organisms
<input checked="" type="checkbox"/>	<input type="checkbox"/> Clinical data
<input checked="" type="checkbox"/>	<input type="checkbox"/> Dual use research of concern

Methods

n/a	Involved in the study
<input checked="" type="checkbox"/>	<input type="checkbox"/> ChIP-seq
<input type="checkbox"/>	<input checked="" type="checkbox"/> Flow cytometry
<input checked="" type="checkbox"/>	<input type="checkbox"/> MRI-based neuroimaging

Antibodies

Antibodies used

anti-cMyc-PE (Cell Signaling Technology, Myc-Tag (9B11) Mouse mAb (PE Conjugate) #3739) in 1:50 dilution, anti-cMyc-FITC (Immunology Consultants Laboratory, CMYC-45F) in 1:100 dilution, Streptavidin R-Phycoerythrin Conjugate (SAPE)

Validation

anti-cMyc-PE (<https://www.cellsignal.com/products/antibody-conjugates/myc-tag-9b11-mouse-mab-pe-conjugate/3739>): "Myc-Tag (9B11) Mouse mAb (PE Conjugate) detects exogenously expressed proteins containing the Myc epitope tag. This antibody recognizes the Myc tag fused to either the amino or carboxy terminus of targeted proteins in transfected cells. Myc-Tag (9B11) Mouse mAb (PE Conjugate) detects exogenously expressed Myc-tagged proteins in cells expressed under a CMV promoter. Expression under other promoters has not been evaluated. The antibody may cross-react with c-myc protein."

anti-cMyc-FITC (<https://www.icllab.com/anti-c-myc-antibody-chicken-fitc-conjugated-cmcy-45f.html>) "This antibody will react with EQKLISEEDL as determined by ELISA and IEP techniques. It is suitable for blotting, ELISA and IF applications. Optimal working dilutions should be determined experimentally by the investigator"

Flow Cytometry

Plots

Confirm that:

- The axis labels state the marker and fluorochrome used (e.g. CD4-FITC).
- The axis scales are clearly visible. Include numbers along axes only for bottom left plot of group (a 'group' is an analysis of identical markers).
- All plots are contour plots with outliers or pseudocolor plots.
- A numerical value for number of cells or percentage (with statistics) is provided.

Methodology

Sample preparation

EBY100 yeast strain was used to clone in gene fragments of the designed sequences with pETCON3 vector for cell surface display.

Instrument

Attune NxT Flow Cytometer (Thermo Fisher) was used.

Software

The provided operating software of Attune was used to collect data, and the software FlowJo was used for data analysis and visualization.

Cell population abundance

We collected data of at least 30,000 cells per analysis.

Gating strategy

We applied gates to exclude outliers using FSC-A/SSC-A, followed by a gating strategy using FSC-A/FSC-H to exclude doublets.

Tick this box to confirm that a figure exemplifying the gating strategy is provided in the Supplementary Information.