ProteinDJ: a high-performance and modular protein design pipeline

Dylan Silke[1,2], Julie Iskander[1,2], Junqi Pan[1,2], Andrew P. Thompson[1,2], Anthony T. Papenfuss[1,2], Isabelle S. Lucet[1,2*], Joshua M. Hardy[1,2*]

1. The Walter and Eliza Hall Institute of Medical Research, Parkville, VIC, Australia.
2. Department of Medical Biology, University of Melbourne, Parkville, VIC, Australia.

*Co-corresponding authors: Isabelle S. Lucet (lucet.i@wehi.edu.au), Joshua M. Hardy (hardy.j@wehi.edu.au)

Abstract

Leveraging artificial intelligence and deep learning to generate proteins *de novo* (a.k.a. 'synthetic proteins') has unlocked new frontiers of protein design. Deep learning models trained on protein structures can generate novel protein designs that explore structural landscapes unseen by evolution. This approach enables the development of bespoke binders that target specific proteins and domains through new protein-protein interactions. However, successful binder generation can suffer from low *in silico* success rates, often requiring thousands of designs and hundreds of GPU hours to obtain enough hits for experimental testing. While commercial web-apps and workstation implementations are available for binder design, these are limited in both scalability and throughput. There is a lack of efficient open-source protein design pipelines for high-performance computing (HPC) systems that can maximise hardware resources and parallelise the workflow efficiently to generate successful binders.

Here, we present 'ProteinDJ'—an implementation of a synthetic protein design workflow that is deployable on HPC systems using the Nextflow portable workflow management system and Apptainer containerisation. It parallelises the workload across both GPUs and CPUs, facilitating generation and testing of hundreds of designs per hour, dramatically accelerating the discovery process. ProteinDJ is designed to be modular and currently includes RoseTTAFold Diffusion (RFdiffusion) for fold generation, ProteinMPNN or Full-Atom MPNN (FAMPNN) for sequence design, and AlphaFold2 or Boltz-2 for prediction and validation of designs and binder-target interfaces, with supporting packages for structural evaluation of designs. ProteinDJ democratises protein binder design through its robust and user-friendly implementation and provides a framework for future protein design software pipelines.

Introduction

Proteins are fundamental biological molecules that perform a wide range of functions in nature, including enzymes that catalyse chemical reactions, transcription factors that regulate gene expression, and antibodies that recognise foreign pathogens. These functions can be regulated through protein-protein interactions (PPIs)[1]. Consequently, the production of novel PPI binders that can specifically activate or inhibit protein function is of immense biomedical and therapeutic interest. This is exemplified by the joint award of the 2024 Nobel Prize in Chemistry to David Baker, Demis Hassabis, and John Jumper for computational protein design and protein structure prediction[2].

Recent advances in artificial intelligence, particularly deep learning approaches, have demonstrated remarkable ability to navigate the sparse solution space of protein binder design by learning underlying patterns and relationships from known protein structures. The current state-of-the-art process for *de novo* protein binder generation includes (1) RFdiffusion[3] for structure generation, (2) ProteinMPNN[4]

for sequence assignment and (3) AlphaFold2 Initial Guess[5,6] for filtering and interaction prediction. Though newer methods like BindCraft[7] and AlphaProteo[8] demonstrate promising results, RFdiffusion remains the most validated benchmark for structure generation. Similarly, alternate software for filtering and interaction prediction exists, including AlphaFold3, Boltz-2 and Chai-1[9-11]. The rapidly advancing nature of this field necessitates versatile and modular workflows to evaluate and implement continuing developments.

Structure generation by RFdiffusion remains a foundational technology for protein design. It employs a diffusion model – a deep learning model that learns to progressively build plausible protein structures from Gaussian noise[3]. For binder generation, this construction process is guided by the target protein surface to create complementary interfaces. It has demonstrated success in generating binders with nanomolar affinity for diverse therapeutic targets, including binders for pathogenic proteins (e.g. influenza hemagglutinin, snake venom) and human cell surface receptors (e.g. insulin receptor, programmed death-ligand 1, platelet-derived growth factor receptor α)[3,12,13].

The success of synthetic binder generation can vary greatly depending on the target protein interface. Achieving enough designs that pass *in silico* metrics and succeed in experimental testing can require thousands of designs and hundreds of GPU hours. While commercial web apps and workstation implementations are available for binder design, these are limited in both scalability and throughput required for the broad application of protein binder design. To meet these needs, *de novo* protein design pipelines must be adapted to HPC systems that can maximise utilisation of hardware resources and parallelise the workflow efficiently. Currently, there is a lack of efficient implementations of such pipelines. Additionally, deployment of these computational tools presents significant technical barriers, requiring careful management of various CUDA versions and Python environments, which limits their accessibility to non-expert users. Finally, metadata from these tools often cannot be easily parsed and analysed using common data analysis tools like Microsoft Excel or pandas, which impedes efficient data processing workflows.

Motivated by these limitations, we developed ProteinDJ (https://github.com/PapenfussLab/proteindj) — a software package that utilises Nextflow[14] to parallelize protein design tools for use with high-performance computing systems. It automates dependency management via Apptainer[15] containers to reduce setup time and efficiently distributes computation across both CPUs and GPUs. ProteinDJ introduces several key improvements to the standard pipeline with guidance for different types of protein designs. ProteinDJ includes a subprogram 'Bindsweeper' for parameter sweeping, allowing users to systematically explore different settings and identify optimal parameter combinations. Structure-guided filtering rejects non-viable designs early in the pipeline, reducing downstream computational costs. To facilitate analysis, ProteinDJ also provides improved design metrics and summary statistics for the filtered outputs. Ultimately, this pipeline aims to improve accessibility and performance of protein binder design and enable uptake of these powerful yet complex software tools.

Results

ProteinDJ is a high-performance and parallelisable protein design pipeline

The ProteinDJ pipeline is divided into four major stages: (1) fold design, (2) sequence design, (3) structure prediction, and (4) analysis and reporting (Figure 1). It integrates different software packages with options for sequence design and structure prediction (Supplementary Table 1). The user can choose from sequence design with ProteinMPNN or the more recent tool FAMPNN that extends

ProteinMPNN to iteratively fit side-chains during sequence design[16]. There is only one set of weights available for Full-atom MPNN, but ProteinMPNN can access two different models: 'vanilla' and 'soluble' (a.k.a. SolMPNN)[17]. ProteinMPNN can also optionally perform an energy minimisation of the binder structure (using the Rosetta FastRelax protocol)[5]. This is a CPU intensive and relatively slow step but allows for iterative sequence design with cycles of ProteinMPNN and Rosetta FastRelax. For structure prediction, there is a choice of Boltz-2[9] or a modified form of AlphaFold2 that can guide prediction with information from the input structure (known as AlphaFold2 Initial Guess)[5], both with GPU acceleration.

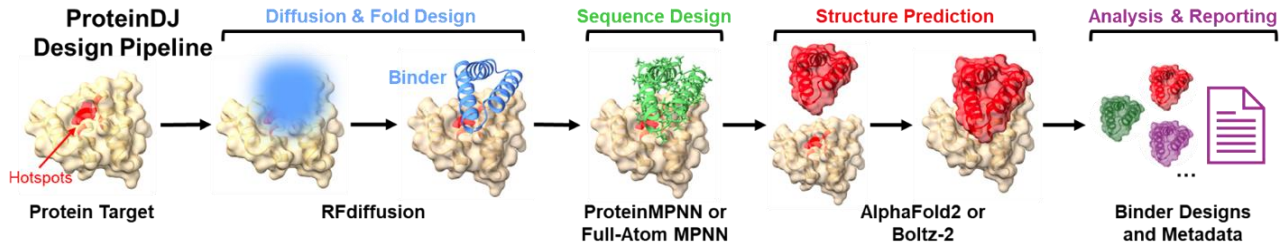


Figure 1 – Overview of the ProteinDJ pipeline for *de novo* binder design. RFdiffusion is used to diffuse a fold/backbone near hotspots on a protein target. The user can choose from ProteinMPNN or Full-Atom MPNN for sequence design, and AlphaFold2 Initial Guess or Boltz-2 for structure prediction and validation.

We designed the pipeline to parallelize stages across available GPUs and CPUs using Nextflow and HPC schedulers (e.g. SLURM), with synchronization points across GPU to CPU stage transitions. ProteinDJ is highly configurable via a centralized Nextflow configuration file enabling users to fine-tune each aspect of the design pipeline. A comprehensive overview of each configuration parameter, their descriptions, and default values is provided in the Methods.

The workflow begins by passing configuration parameters and any input files (Figure 2a) to RFdiffusion to generate new secondary and tertiary structures as poly-glycine folds (Figure 2b). Key configuration parameters include the number of requested designs, the choice of sequence design or structure prediction software, and the output directory (see Methods). The total number of requested designs is divided into batches per GPU to maximise parallel throughput. Generated folds then transition to a filtering stage, where the radius of gyration (compactness) and the number of alpha-helices and beta-strands are counted, allowing rejection of low complexity folds (e.g. single alpha-helices) and compact or elongated binders depending on the need of the design case. Folds meeting these criteria then proceed to either ProteinMPNN (with or without FastRelax) or FAMPNN for sequence assignment of the RFdiffusion poly-glycine binders (Figure 2c). The user determines the number of sequences per fold to be generated, and the models are distributed across available CPUs for ProteinMPNN or GPUs for FAMPNN. The outputs of the sequence generation step move to a second filtering stage where designs with poor sequence scores are rejected. Examples of criteria for rejection include negative log-likelihood for ProteinMPNN or predicted sidechain error for FAMPNN.

In the next stage of the pipeline, sequence designs are validated by structure prediction using GPU-parallelized Boltz-2 or AlphaFold2 Initial Guess (Figure 2d). As RFdiffusion runtime scales exponentially with target size for binder design tasks, it is common practice to truncate input target structures. While this improves runtime, it can inadvertently lead to the design of binders that clash with truncated components of the target structure or that interact with exposed hydrophobic residues. To address this, we have designed and implemented optional functionality that replaces the target chain with an uncropped target structure before structure prediction. A final filtering stage is used to filter

out designs with low prediction confidence (e.g. predicted aligned error) or with significant differences to the input RFdiffusion fold.

Designs that pass structure prediction filters are aligned and proceed to a final analysis stage (Figure 2e), that uses PyRosetta[18] and Biopython[19] to generate additional metrics, as described in Supplementary Table 3. Metrics from the entire pipeline are subsequently collated into a CSV file for user perusal. The best designs are output to the results directory as aligned PDB files for easy comparison and visualisation (e.g. in ChimeraX[20]). A summary of the workflow and success rates is printed to the log and output (Supplementary Figure 1).

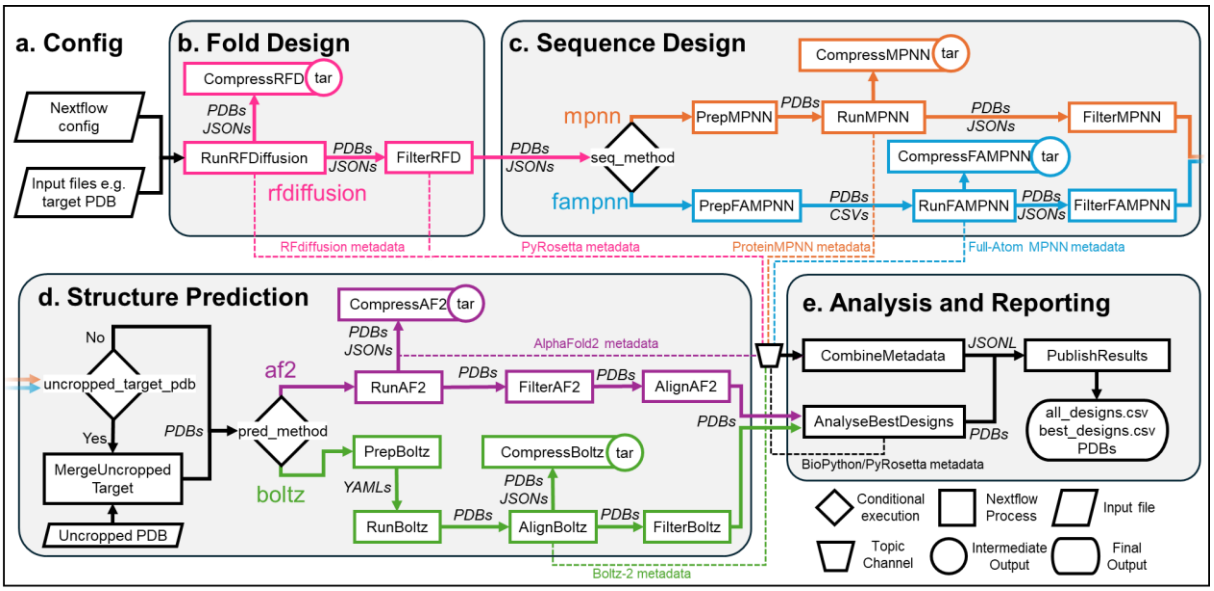


Figure 2 – Data flow and connectivity diagram of ProteinDJ. a. User parameters and files are provided to Nextflow to execute a ProteinDJ rim. **b.** In the first stage, RFdiffusion is used to generate folds (RunRFDiffusion). All output PDBs and metadata (JSON files) are captured and compressed as intermediate results (CompressRFD) and passed to a filtering stage (FilterRFD). Metadata from both RunRFDiffusion and FilterRFD are exported to a shared 'topic' channel for metadata collection and integration. **c.** In the sequence design stage, the user can select between two packages using the 'seq_method' parameter. Each has a preparatory stage (PrepMPNN/PrepFAMPNN) in which metadata from RFdiffusion is used to mask residues during sequence design (for RunMPNN and RunFAMPNN respectively). Similarly to RFdiffusion, intermediate outputs are captured and compressed (CompressMPNN/CompressFAMPNN), before filtering (FilterMPNN/FilterFAMPNN). **d.** The user can optionally provide an uncropped target PDB file for structure prediction that is merged with designs in MergeUncroppedTarget. There are two methods available for structure prediction (RunAF2 and RunBoltz), selected using the 'pred_method' parameter. When selecting Boltz-2 for prediction, the PrepBoltz stage is used to generate a YAML containing the sequences of the designs. For ease of visualisation, the designs are aligned to each other (AlignAF2/AlignBoltz), and a third and final filtering step is optionally performed (FilterAF2/FilterBoltz). **e.** In the analysis and reporting stage, the remaining PDB files are subjected to further analysis (AnalyseBestDesigns), and these metrics are combined with other metrics from other stages into a CombineMetadata process. This generates two CSV files, one with metadata of all designs (all designs.csv) and designs that passed filtering if enabled (best_designs.csv) and is published along with the PDB files that survived filtering (PublishResults).

Despite the complexity of the workflow, ProteinDJ exhibits scaled efficiently across multiple GPUs, achieving near-linear reductions in runtime compared to the serial baseline, with minimal loss in parallel efficiency. Using 8 A30 GPUs, ProteinDJ generated and evaluated 500 binder folds and 4000 unique sequences using ProteinMPNN and AlphaFold2 in a run time of 2 hours and 26 minutes, compared to 16 hours and 55 minutes when using a single GPU (See Methods and Supplementary

Figure 2a). We observed a parallel efficiency of 97.0%, 93.6%, and 86.5% over 2, 4, and 8 NVIDA A30 GPUs respectively (Supplementary Figure 2b).

We also compared the efficiency of each software package in generating 1,000 binders (100 folds, 10 sequences per fold) against a benchmarking target, Programmed cell death 1 ligand 1 (PD-L1) (Supplementary Table 6). RFdiffusion generated a fold every 54.9 seconds on average. ProteinMPNN designed a sequence every 1.3 seconds and slowed to 56.9 seconds per design when performing one FastRelax cycle, whereas FAMPNN required only 4.8 seconds per sequence. Boltz-2 was more efficient than AlphaFold2 Initial Guess and generated a prediction every 8.6 seconds on average compared to an average of 16.4 seconds per prediction with AlphaFold2 Initial Guess.

Modular design enables future software integrations

To ensure modularity of the workflow and facilitate integration of new software, we standardised the data and metadata formats. Each design is passed as a single PDB file, and rather than embed the metadata into the PDB file, which can be difficult to extract later, we embed all metadata from the stage into a JSON file. To pair designs with metadata, we used a unique fold number (fold_id), to differentiate each RFdiffusion design, and a unique sequence number (seq_id) once a sequence has been generated for a fold. Each metadata filename and entry contain these ID fields so that metadata can be merged at the end of the pipeline into a user-friendly CSV format. The IDs also allow batching of each PDB and JSON file pair into different batch sizes as appropriate for CPU or GPU processing.

Rather than pass each metadata file to the next software package, we collect the metadata using a Nextflow topic channel that can consume outputs from different processes. This ensures metadata is published at each stage of the workflow and captured even if the pipeline is interrupted. By only passing the PDB files between stages, we decoupled metadata processing between different software packages, allowing for alternative workflows and software options. Although compressed and storage efficient formats exist for PDB files (e.g. the 'silent' format from https://github.com/bcov77/silent_tools), we did not implement them as it complicates batching of PDB files in Nextflow. To minimize impact on storage, however, we compressed intermediate PDB and metadata files before publishing to the output directory.

Implementation of modes to RFdiffusion delineate and simplify protein design workflows

RFdiffusion is a flexible protein design software that contains a plethora of optional parameters. To increase accessibility of the pipeline to new users, we have divided the pipeline into monomer and binder design workflows with four modes for each: '*de novo*', 'fold conditioning', 'partial diffusion' and 'motif scaffolding' (Figure 3). These modes organise the existing optional parameters of RFdiffusion into defined applications where they are relevant. By streamlining the pipeline entry-points this way, we aim to reduce unexpected conflicts and errors during execution from incompatible RFDiffusion parameters (Supplementary Table 4).

The simplest *de novo* mode, 'monomer_denovo', diffuses a single chain in each design and requires only specification of the chain length. The 'binder_denovo' mode diffuses binders to a target protein derived from an input PDB file with the option to provide specific residues, called 'hotspots', to focus the design on (Figure 3a). This approach can be used to generate new binders to targets that lack biological adaptors or existing binders. The partial diffusion modes, 'monomer_partialdiff' and 'binder_partialdiff', will noise and denoise an input monomer or binder (or parts thereof) to generate

variants, according to the number of diffusion timesteps provided (Figure 3b). Partial diffusion can be used to optimise candidate binders from *de novo* methods.

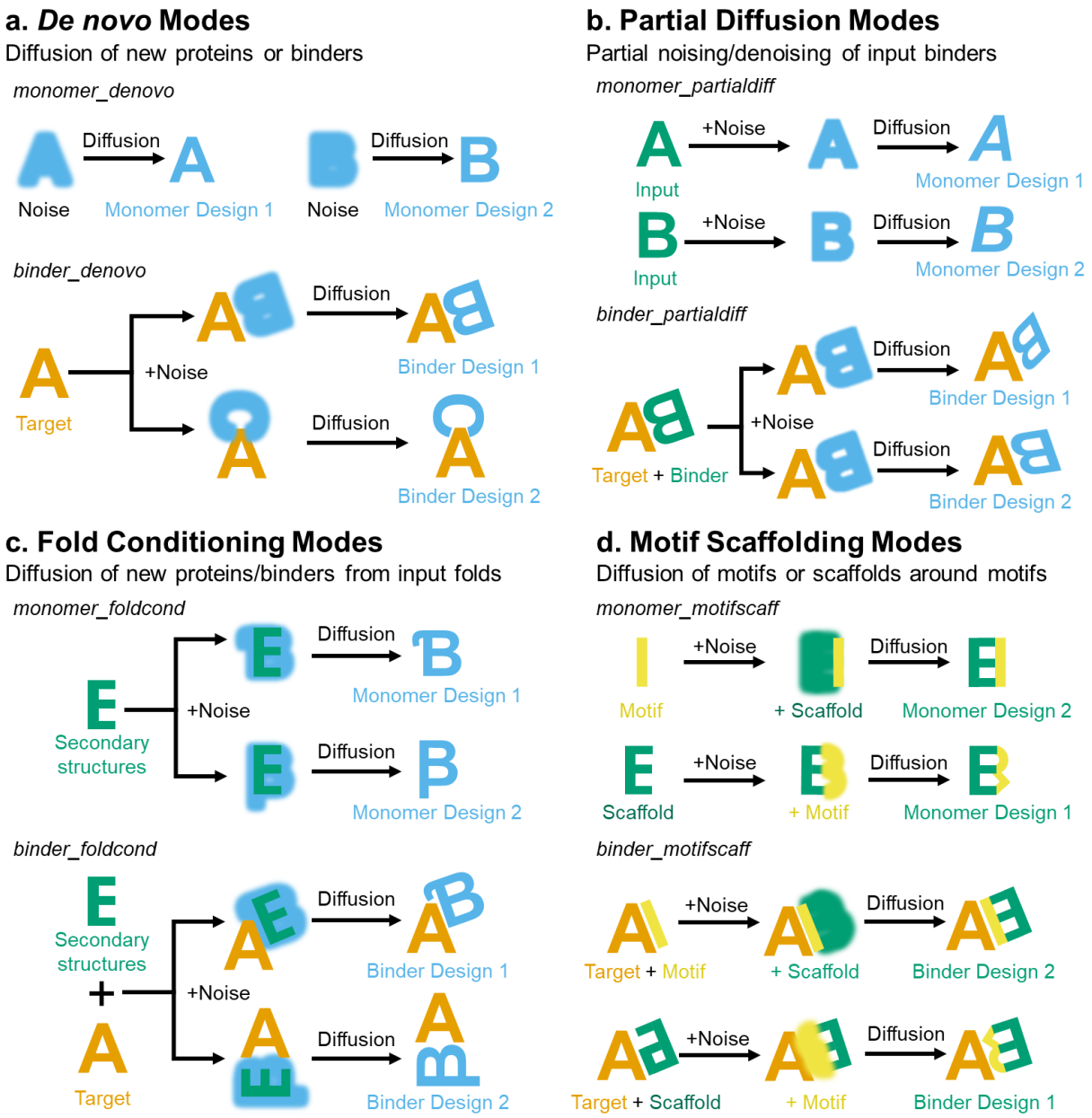


Figure 3 – Protein design modes for ProteinDJ. a. *De novo* protein design is used to diffuse new proteins from noise, either in isolation ('monomer_denovo') or with an input chain ('binder_denovo'). **b.** Partial diffusion can be used to generate variations of input proteins ('monomer_partialdiff') or binders ('binder_partialdiff') by noising and denoising structures. **c.** Fold conditioning is a constrained form of *de novo* protein design in which the designs must follow secondary structure definitions provided during inference and can be used for monomers (monomer_foldcond) or binders (binder_foldcond). **d.** Motif scaffolding provides the ability to inpaint or outpaint existing structures, while preserving key regions of the input structure.

The fold conditioning modes, 'monomer_foldcond' and 'binder_foldcond', allow users to guide diffusion of new proteins with specific folds (Figure 3c). This requires generation of files containing secondary structure information from a library of scaffold structures. To assist users in binder design, we have generated scaffold libraries of four fold types using PDB files from a previously published minibinder study[21]: 3-helical and 4-helical bundles, and two mixed alpha-helical and beta-stranded

folds based on ferredoxin and a synthetic fold (Figure 4). Although this mode lacks the variability of designs produced by *de novo* modes, the folds are more likely to be recapitulated during structure prediction and this increases the *in silico* success rate (as previously observed[3]). RFdiffusion also has a strong bias towards alpha-helical folds when performing *de novo* fold design that is somewhat mitigated by a 'beta' checkpoint model for RFdiffusion[3], but fold conditioning allows for explicit use of beta-sheet rich structures that may be preferred in certain design contexts.

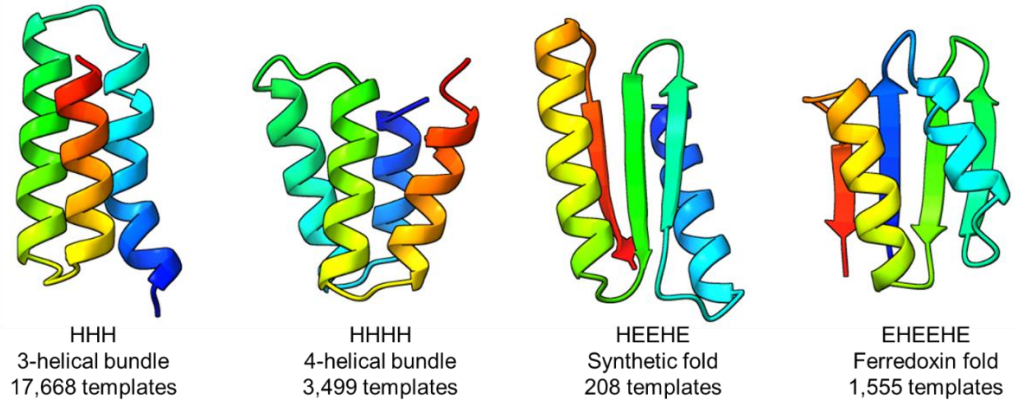


Figure 4 – Examples of the four scaffold types provided for binder fold conditioning containing alpha-helical (H) and beta-strand (E) secondary structures. Structures are rendered as cartoons with rainbow colouring from N- to C- termini in ChimeraX[20].

The motif scaffolding modes, 'monomer_motifscaff' and 'binder_motifscaff', are the most complex and require specification of specific residues or regions for diffusion (Figure 3d). These modes can be used to modify regions of input proteins while preserving the rest of the structure and/or sequence. This can be used to diffuse a scaffold around a binding motif, or to diffuse a binding motif into a scaffold. To facilitate this, we directly map the RFdiffusion sequence masks to ProteinMPNN/FAMPNN to allow for partial changes to design sequences (see Methods).

Our implementation of modes for RFdiffusion allow for differentiation in the downstream analysis of designs. For example, in monomer design options, the metrics can simply be calculated for all residues because there is only one chain, whereas in binder modes, metrics can be calculated for residues from the binder chain only and ignore target residues if applicable. The modes also facilitate the use of downstream software that have strict input requirements, such as a single target chain and binder chain only.

Some parameters are common to each mode, such as the number of designs and the output directory, but most optional parameters are restricted to specific modes (see Supplementary Table 4). For example, when performing partial diffusion of binders, the binder position is already known and thus specifying 'hotspots' is not meaningful, so this parameter is ignored. Likewise, providing 'timesteps' for partial diffusion is only relevant when a structure is provided and it is therefore disabled for *de novo* design. We have created schema files for running on the Nextflow Seqera Platform[14] that show only relevant parameters for each mode with input validation and format checks during job configuration. We have also implemented input validation of files and syntax before launching RFdiffusion, for example, checking that the hotspot residue numbers are within the residue ranges provided for the input protein, a common error for new users.

Development of a parameter sweeping algorithm for binder design - Bindsweeper

The most efficient parameters for binder design can vary between target proteins and identifying the ideal parameters can be laborious, requiring manual exploration of key parameters such as target hotspots. To enable users to automatically and systematically explore design parameters, we developed Bindsweeper – a Python-based command-line tool that can coordinate multiple ProteinDJ runs. Bindsweeper processes ProteinDJ parameters via command-line inputs and a YAML configuration file (Figure 5a), providing flexible execution options including a quick testing mode that tests each parameter combination with a smaller set of designs.

Bindersweeper generates a multi-dimensional parameter matrix by computing all possible combinations of specified sweep parameters (Figure 5b), supporting both list sweeps for discrete values and range sweeps for continuous values defined by start, stop, and step parameters. During execution (Figure 5c), Bindsweeper creates Nextflow profiles for each parameter combination, and executes pipeline runs with detailed timing and success tracking. After completing all parameter combinations, the results processor extracts putative binder PDB files, merges CSV outputs from each parameter combination, and performs success analysis to identify best-performing parameter combinations, with designs organized for downstream analysis and validation.

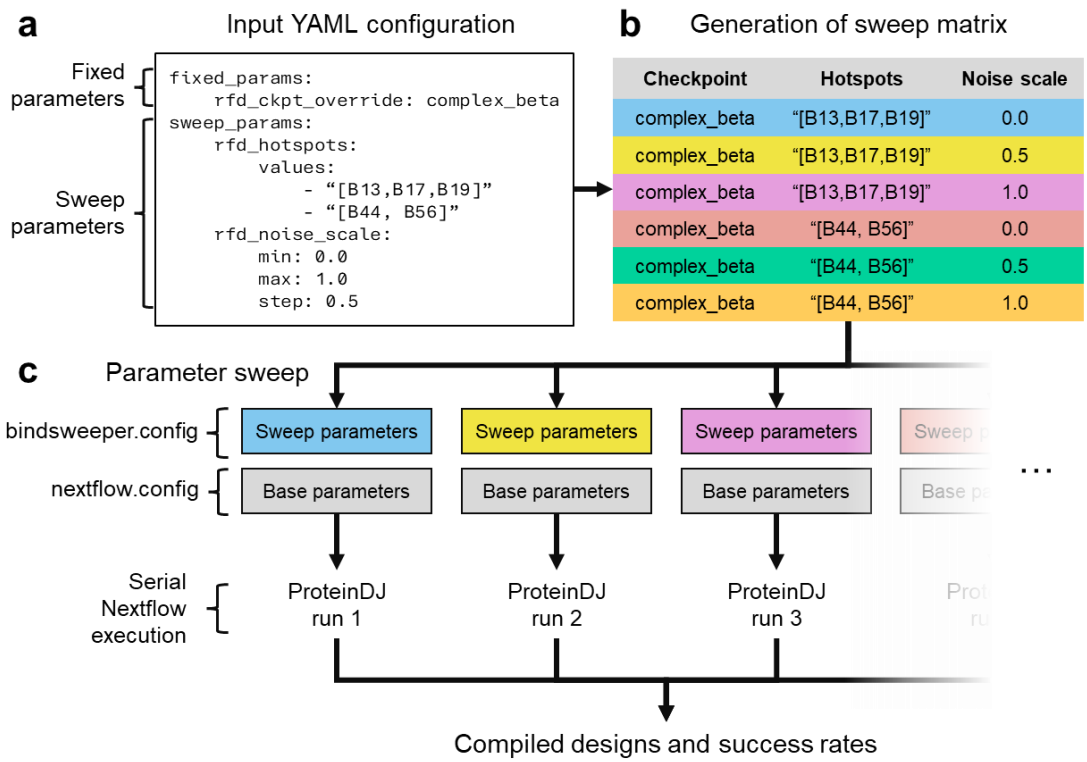


Figure 5 – Workflow diagram for Bindsweeper parameter sweep algorithm. Bindsweeper uses an input YAML file (a) to generate a multi-dimensional matrix of sweep parameters (b), that can be used to perform sequential runs of ProteinDJ (c).

Bindsweeper facilitates benchmarking and comparison of protein design software

To demonstrate the utility of Bindsweeper, we used it to benchmark and compare sequence design software. We used RFdiffusion to generate 100 *de novo* binder designs against each of the five benchmarking target proteins used in the RFdiffusion publication[3] (Supplementary Table 6) and provided these to ProteinMPNN to generate 8 sequences for each, with and without a cycle of

FastRelax, and with the standard ProteinMPNN model (a.k.a. 'vanilla' model) and the SolMPNN model. The same folds were also provided to FAMPNN and sequences were designed with and without fixing the position of target side-chains during design. After sequence design, structure prediction with AlphaFold2 Initial Guess was used to determine success rates using previously published metrics and thresholds (see Methods). The success rates varied dramatically between target proteins, in line with previous reports[3]. However, Bindsweeper revealed that the sequence design method also resulted in significant differences across the targets. With ProteinMPNN, one cycle of FastRelax improved the success rates for all targets, and the SolMPNN model outperformed the vanilla model for 3 of 5 targets. FAMPNN had the lowest success rates for 2 targets but performed better or similarly to ProteinMPNN in others. For most targets, fixing the side-chains on the target during sequence design with FAMPNN lowered the success rate.

Table 1 – Comparison of relative success rates of sequence inference methods for *de novo* binder design against each target. For each target, the same 100 folds were provided to each sequence design combination to generate 8 sequences each (n = 800). The highest success rate using AlphaFold Initial Guess for each target is bolded. HA = Influenza A H1 haemagglutinin, IL-7Rα = Interleukin-7 receptor alpha, InsR = Insulin Receptor, PD-L1 = Programmed cell death 1 ligand 1, TrkA = Tropomyosin receptor kinase A.

Sequence Design Method	HA	IL7Rα	IR	PD-L1	TrkA
ProteinMPNN(Vanilla)	3.9%	6.0%	21.3%	30.6%	5.4%
ProteinMPNN(Vanilla) + FastRelax	**7.0%**	**11.1%**	26.4%	47.5%	12.1%
ProteinMPNN(Soluble)	5.0%	5.8%	27.6%	33.9%	8.5%
ProteinMPNN(Soluble) + FastRelax	6.1%	9.1%	**29.6%**	**52.3%**	**18.1%**
FAMPNN	0.1%	10.9%	11.6%	30.9%	7.5%
FAMPNN (target side-chains fixed)	1.6%	4.8%	8.0%	9.9%	4.1%

Discussion

Recent breakthroughs in artificial intelligence, particularly in deep learning, has significantly advanced our ability to predict protein structures with accuracy. These advancements, exemplified by AlphaFold and RFdiffusion have transformed the landscape of computational protein design and now allow for precise design of new PPIs. However, the efficiency of this task has been hindered by incompatibility between programs and significant effort in installing, configuring, and maintaining protein design software workflows. Furthermore, current public implementations of these software workflows are designed for workstation use, which operate at a sub-optimal capacity for efficiently generating the volume of synthetic designs required for success. For research institutes with several research groups designing binders against multiple targets, scalable workflows are crucial in preventing bottlenecks and delivering timely results.

In this paper, we present a robust Nextflow pipeline for protein design that brings together diverse software packages in a unified and efficient framework. The pipeline incorporates several features that simplify and streamline protein design workflows. It scales with 86.5% efficiency across 8 GPUs, reducing the time to generate designs by almost an order of magnitude. Automated and detailed metadata management and reporting have been implemented to facilitate modularity and user control. Our uncropped structure prediction functionality reintroduces biological and structural context to the design process. Finally, we have delineated the software into simplified protein design tasks, with automated parameter selection, tailored to the user's expertise. The software is supported by detailed documentation and examples of different design tasks.

ProteinDJ implements filtering steps after each design stage, which can be adjusted depending on design objectives, allowing the pipeline to be adapted for different protein design challenges. The filters also automatically reject designs without manual curation, saving valuable time. This is particularly valuable when targeting different protein-protein interaction types or when exploring novel fold spaces where standard filtering criteria may need adjustment. We provide metadata for rejected designs in one of the output CSV files (all_designs.csv), allowing users to see the scores before they were rejected and adjust the filters if needed to include. Additionally, we added new metadata on the biophysical and sequence characteristics of designs, such as the extinction co-efficient (i.e. whether the protein is detectable by UV spectroscopy) and the isoelectric point (pI) and molecular weight of the sequence, to support subsequent *in vitro* expression, purification and validation.

Our new tool Bindsweeper gives users the ability to automatically sweep parameter combinations for binder design. This enables systematic small-scale optimisation of parameters for target proteins such as the identification of hotspot residues, before embarking on large-scale design campaigns. This is especially important when success rates are so low as to prevent practical generation of sufficient binders for testing. Researchers must carefully balance the time spent on computational design prior to the more costly experimental validation process.

Bindsweeper also facilitates comparison and benchmarking of software packages within ProteinDJ, as we have demonstrated above. For example, we observed higher in silico success rates when using FastRelax in combination with ProteinMPNN compared to using ProteinMPNN alone, with the same input folds (Table 1). This improvement to success rates with RFdiffusion folds was similar to that observed by Watson et al.[3], but while they reported inconsistent improvement to success rates across benchmarking targets, we observed a consistent improvement for the same targets. This could be due to differences in our benchmarking setups and whether the same input folds were used for each run. Whilst FastRelax demonstrates higher success rate, it does at the cost of increased run times per design. Further experimental studies are needed to evaluate whether this enhances success rates in vitro.

We have extended the functionality of Full-Atom MPNN and Boltz-2 for binder design with additional metrics and parameters. Although Full-Atom MPNN extends ProteinMPNN and performs additional modelling of side-chains during sequence design, there was not a consistent improvement to success rates (Table 1). However, for some targets, it outperformed ProteinMPNN, highlighting the benefits of comparing sequence design software for different target classes. Boltz-2 was more efficient than AlphaFold2 in generating structure predictions, but experimental testing is needed to compare their effectiveness in identifying promising designs. In a recent preprint containing a retrospective analysis of previous binder design campaigns, a previous version of Boltz (Boltz-1) performed similarly to AlphaFold2 in identifying successful binders when using metrics derived from the prediction aligned error of interface residues[22]. This study also indicates that when prediction scores are combined with biophysical metrics such as shape complementary and buried surface area, metrics captured in our analysis pipeline, they can be powerful selectors for successful designs.

Our protein design pipeline is constructed with several core design principles in mind. ProteinDJ assumes a linear design process i.e. sequential design steps without recycling of designs back through the pipeline. An automated iterative pipeline could be useful for some design cases to refine sequences and folds, if the pipeline avoids combinatorial explosion and can perform effective refinement unsupervised. However, we suggest that it would be preferable to intermittently inspect cycle outputs before embarking on computationally expensive large-scale design campaign.

Currently, ProteinDJ pipeline performs amino acid design of a single protein chain. In the future, we aim to implement recent tools such as RFdiffusion2 and LigandMPNN that can also perform protein design with small molecules. Although RFdiffusion can perform design of symmetric oligomers, we found that ProteinMPNN and FAMPNN were unable to perform symmetric sequence design without significant reprogramming, and so symmetric design was not enabled. However, as ProteinDJ was designed with modularity in mind, we envision future versions of this pipeline that include new and promising technologies.

In designing ProteinDJ we aimed to address the current lack of standardised file formats and efficient data management, and establish consistent format conventions for protein design tools, structures and metadata. Due to the scale of binder design tasks, which can generate hundreds or thousands of structure files, tools are available to convert PDB/mmCIF files to more compressed formats, greatly reducing file size at the cost of interoperability between tools. In our pipeline, we chose to compress intermediate results into standard gzip archives, while keeping temporary files that are passed between stages as uncompressed PDB files. Additionally, the organisation and encoding of metadata varies dramatically between protein design tools and often requires custom scripts to compile metadata and convert it to human-friendly formats. To establish consistency, we opted to use JSON files for all metadata, which was key for our modular approach and in our downstream processing and compilation. These approaches maximise interoperability and ease integration of future software packages.

In conclusion, our goal with this pipeline was two-fold: first, to package and implement protein design tools in a HPC environment at maximal efficiency; and secondly, to reduce the prohibitive complexity of these tools by providing an accessible interface with curated defaults and guidance for different design tasks. By incorporating different tools for sequence design and structure prediction, we allow for easy exploration and comparison of software tools for a design problem. Our open-source framework in ProteinDJ enables collaborative efforts to build similar workflows and incorporate future tools in this rapidly growing field of protein design.

Methods

Construction of ProteinDJ pipeline

We containerised all pipeline dependencies using Apptainer/Singularity[15], summarized in Supplementary Table 1. The container images are available on the Australian Research Data Commons (ARDC) Nectar Research Cloud[23]. ProteinDJ Nextflow profiles are set to pull the images for deployment on different HPC systems. Integration and containerisation required modification to the source code of some software packages as described below. Bindsweeper utilises Python3 packages including Click, PyYAML, and Pandas that requires installation of a python environment locally.

Generative AI coding tools (ChatGPT, Claude) were used to review and augment scripts and assist with troubleshooting. All outputs were critically reviewed, tested, and adapted by the authors. These tools were not used for writing or editing of the manuscript or for data analysis.

Nextflow pipeline

We built the Nextflow pipeline for ProteinDJ with four main stages described in Figure 1 and implemented several high level parameters that control the execution mode and what programs are utilised for sequence design and structure prediction (Table 2). We also created parameters to allow

skipping of stages or to run RFdiffusion only and exit, that are useful for testing or running comparisons e.g. structure prediction tools on the same set of binder designs.

Table 2 – Parameters for controlling Nextflow workflow

Parameter	Default	Description
out_dir	./pdj_results	Output directory for results. Existing results will be overwritten
seq_method	'mpnn'	Sequence design method. Options: 'mpnn' (ProteinMPNN FastRelax) or 'fampnn' (Full-Atom MPNN).
pred_method	'af2'	Structure prediction method. Options: 'af2' (AlphaFold2 Initial Guess) or 'boltz' (Boltz-2).
run_rfd_only	false	Whether to run only RFdiffusion and skip sequence design, prediction, and analysis.
skip_rfd	false	Skip RFdiffusion, run sequence design, prediction, and analysis only. Requires valid skip_input_dir containing PDB and JSON files with metadata. Binder design PDBs must have binder as chain A and target as chain B.
skip_rfd_seq	false	Skip RFdiffusion and sequence design, run structure prediction and analysis only. Requires valid skip_input_dir containing PDB files. Binder design PDBs must have binder as chain A and target as chain B.
skip_rfd_seq_pred	false	Skip RFdiffusion, sequence design, and prediction, run analysis only. Requires valid skip_input_dir containing PDB files. Binder design PDBs must have binder as chain A and target as chain B.
skip_input_dir	null	Directory path for files when skipping stages (e.g. './rfd_results').
container_dir	Cloud container location or local path	Path to the pipeline containers directory
rfd_models	‘./models/rfd’	Path to the RFdiffusion model checkpoints
af2_models	‘./models/af2’	Path to the AlphaFold2 models
boltz_models	‘./models/boltz’	Path to the Boltz-2 models
gpu_model	‘A30’	GPU model requested (e.g. 'A30')
gpu_queue	null	GPU queue name or partition
gpus	4	Number of GPUs to request
cpus_per_gpu	8	Number of CPUs requested per GPU
cpus	24	Number of CPUs requested for CPU-only jobs
memory_gpu	‘24GB’	Memory requested for GPU jobs
memory_cpu	’24GB’	Memory requested for CPU-only jobs

RFdiffusion

The RFdiffusion environment configuration file was modified to resolve issues with a deprecated mode of installing PyTorch and NumPy[24], and we also fixed a bug for fold-conditioning. We implemented a Groovy script that both validates and converts input parameters from Nextflow to RFdiffusion (see parameter list in Table 3). This script allows for segmentation of RFdiffusion runs into different modes and prevents conflict of parameter choices. Helper functions provide a human-readable interpretation of contigs and check that hotspot residues are present in input residues ranges to flag issues before launching the job. The number of designs (for inference.num_designs) is divided across GPUs evenly, rounding up if needed. The output TRB files containing metadata are ‘unpickled’ and converted to JSON files.

Table 3 – Parameters for RFdiffusion and fold generation

Parameter	Example Value	Description
rfd_mode (new)	‘monomer_denovo’	Pipeline mode. Choose from 'monomer_denovo', 'monomer_foldcond', 'monomer_motifscaff', 'monomer_partdiff', 'binder_denovo', 'binder_foldcond', 'binder_motifscaff', or 'binder_partialdiff',
rfd_num_designs	8	Number of designs to generate using RFdiffusion.
rfd_contigs	'[100-100/0 B101-221]'	Specifies the residue ranges to be diffused.
rfd_input_pdb	'./target.pdb'	Path to an input PDB file.
rfd_hotspots	'[A56,A115,A123]'	Specifies hotspot residues for binding.
rfd_scaffold_dir	'./binderscaffolds'	Path to directory containing scaffold files.
rfd_target_adj	'./target_adj.pt'	Block adjacency file for scaffold-guided target.
rfd_target_ss	'./target_ss.pt'	Secondary structure file for scaffold-guided target.
rfd_mask_loops	false	Whether to ignore loop regions when applying secondary structure constraints from scaffolds (default=true). Recommended to set to false for binder design as the scaffold sets have enough diversity
rfd_inpaint_seq	‘[A17-19/A143-145]’	Residues that will have their sequence masked when diffusing
rfd_length	100-120	Length of diffused chain when performing motif scaffolding
rfd_partial_diffusion_timesteps	20	Number of timesteps for partial diffusion.
rfd_ckpt_override	‘complex_base’	Override the diffusion model checkpoint used by RFdiffusion. Choose from: 'active_site', 'base', 'base_epoch8', 'complex_base', 'complex_beta', 'complex_fold_base', 'inpaint_seq', 'inpaint_seq_fold'. RFdiffusion will automatically select a checkpoint. Not recommended to override except for 'binder_denovo' mode i.e. 'complex_beta'
rfd_noise_scale	0.5	Scale of noise for RFdiffusion translations (C-alpha) and rotations (frames). 1 (default) is recommended for non-binder modes to increase diversity. 0-0.5 is recommended for binder modes to increase success rates.
rfd_extra_config	'contigmap.inpaint_str=[B165-178]'	Additional configuration parameters to pass to RFdiffusion that are not externalised in Nextflow.

ProteinMPNN FastRelax

We modified the python environment of ProteinMPNN FastRelax to use conda-forge instead of the Anaconda defaults channel and removed PyTorch and NVIDIA dependencies as we are using CPU-only functionality as recommended by Bennet *et al*[5]. Additionally, we switched to NumPy v2 and used OpenBLAS (Basic Linear Algebra Subprograms) over Intel MKL (Math Kernel Library) to resolve intermittent crashes when using FastRelax. We used the 'inpaint_seq' array from RFdiffusion to identify residues to retain during sequence design and add 'FIXED' remarks to the input PDB files to ProteinMPNN. To extend the functionality of the FastRelax protocol, we created a modified version of the dl_interface_design python script (dl_interface_design_multi.py). We created parameters to control the maximum number of cycles and convergence criteria to end cycles early, the number of sequences to generate between each relax cycle, and optional relaxation of the final design (see Table 4). We also generated a FastRelax protocol for monomer designs. When generating multiple sequences between relax cycles, the sequence with the lowest (best) score is automatically selected. When writing output PDB files, we also output a JSON file containing the design file name, sequence, and score.

Full-Atom MPNN (FAMPNN)

Integration of FAMPNN required fixes to weight loading on GPUs. We modified the source code to add a parameter to exclude cysteine residues when performing sequence design (fampnn_exclude_cys). As with ProteinMPNN, we map fixed residues from RFdiffusion and exclude these from sequence design in the input CSV for FAMPNN. Since RFdiffusion removes all side-chain atoms from target residues, and FAMPNN requires sidechains for design context, we use PyRosetta to add sidechain atoms to fixed residues before design. Additionally, we derived a summary score for sequence design using the predicted side-chain error (PSCE) assigned to each atom by FAMPNN. This average PSCE is calculated per residue but excludes backbone atoms (that have a PSCE=0) and optionally C-beta atoms (that consistently have a PSCE ~0.6), thereby weighting the score towards designed side-chain atoms. This score and the designed sequence are then output to a JSON metadata file.

Table 4 – Parameters for sequence design using ProteinMPNN (mpnn) and Full-Atom MPNN (fampnn).

Parameter	Default Value	Description
seqs_per_design	8	Number of sequences to generate per RFdiffusion design.
mpnn_omitAAs	'CX'	A string of all residue types (one letter case-insensitive) to avoid for design.
mpnn_temperature	0.1	Temperature for sequence sampling. Higher values increase sequence diversity. Recommended to lower significantly for binders to improve success (e.g., 0.0001).
mpnn_checkpoint_type	vanilla	Checkpoint type to use for ProteinMPNN. Choose from 'vanilla' or 'soluble'
mpnn_checkpoint_model	v_48_020	Checkpoint model to use for ProteinMPNN. Indicates noise level used during training for backbone atoms e.g. 'v_48_020' noised with 0.20Å Gaussian noise. Choose from 'v_48_002','v_48_010,'v_48_020','v_48_030'.
mpnn_backbone_noise	0	Standard deviation of Gaussian noise to add to backbone atoms. 0 = no change to backbone, 0.1-0.3 = allows mild perturbation of backbone.
mpnn_relax_max_cycles (previously relax_cycles)	0	Maximum number of fast relax cycles to run on each sequence. Set to zero to disable.
mpnn_relax_seqs_per_cycle (new)	5	Number of sequences to generate between relaxation steps. The sequence with the best score will be kept.
mpnn_relax_output (new)	false	Whether to run a relaxation cycle before saving output.
mpnn_relax_convergence_rmsd (new)	0.2	Design is considered converged if the C-alpha RMSD (Å) between cycles is <= this threshold.
mpnn_relax_convergence_score (new)	0.1	Design is considered converged if the improvement in score between cycles is <= this threshold.
mpnn_relax_convergence_max_cycles (new)	1	Design is considered converged if it meets both convergence criteria for n consecutive cycles.
mpnn_extra_config	null	Additional configuration parameters to pass to ProteinMPNN-FastRelax that are not externalised in Nextflow.
fampnn_fix_target_sidechains	false	Fix target side-chain positions when performing binder sequence design
fampnn_psce_threshold	0.3	Only keep sidechains below this PSCE threshold during design; null to keep all.
fampnn_temperature	0.1	Temperature for sequence sampling. Higher values increase sequence diversity.
fampnn_exclude_cys (new)	True	Exclude cysteine residues from sequence design.
fampnn_extra_config	null	Additional configuration parameters to pass to Full-Atom MPNN that are not externalised in Nextflow.

AlphaFold2 Initial Guess

For structure validation, we use the AlphaFold2 Initial Guess implementation from the dl-binder-design repository (https://github.com/nrbennet/dl_binder_design)[5]. Prior to containerisation we updated the deprecated PyRosetta installation method and resolved a deprecated import in the MMCIF parsing module that was causing install to fail. We observed that when the sequence length changed, AlphaFold2 generated new features so to improve efficiency we sort binders by size when batching to GPUs. As RFdiffusion scales poorly with target size for binder design tasks it is common practice to

truncate target structures before design, but this can lead to issues in structure prediction, where the context is missing. We implemented new functionality to optionally replace the target chain in the protein designs with an uncropped target structure before structure prediction. The metrics from AlphaFold2 are converted to a JSON format, with binder_rmsd and target_rmsd renamed to rmsd_binder_bndaln and rmsd_binder_tgtaln for clarity. We added a rmsd calculation for all chains (rmsd_overall) and the target chain only (rmsd_target), as well as reporting of the predicted aligned error for all chains (pae_overall).

Boltz-2

As Boltz-2 is a general purpose structure prediction program and not designed exclusively for protein design like AlphaFold2 Initial Guess, we constructed python scripts to generate input YAML files with target and design sequences. We also created a process to align predictions with input designs to calculate RMSD values. We calculate three RMSD values after alignment: overall RMSD of all C-alpha atoms, and for binder design, RMSD between the binder design and prediction, and RMSD between the target chains and prediction. To speed up structure prediction, we disable multiple sequence alignment and perform inference only. The metrics from Boltz2 are converted to a JSON format, with 'ipde', 'iplddt', and 'iptm' renamed to 'pde_interface', 'plddt_interface', and 'ptm_interface', respectively, for clarity.

Table 5 – Parameters for structure prediction using AlphaFold2 Initial Guess (af2) and Boltz-2 (boltz)

Parameter	Default Value	Description
uncropped_target_pdb (new)	null	Path to uncropped target PDB file to use in structure prediction
af2_initial_guess	true	Whether to use an initial guess for target chains in AlphaFold2 structure predictions.
af2_extra_config	null	Additional configuration parameters to pass to AlphaFold2 Initial Guess that are not externalised in Nextflow. e.g. '-recycle 3'
boltz_recycling_steps	3	The number of recycling steps to use for Boltz-2 prediction.
boltz_diffusion_samples	1	The number of diffusion samples to use for Boltz-2 prediction.
boltz_sampling_steps	200	The number of sampling steps to use for Boltz-2 prediction.
boltz_extra_config	null	Additional configuration parameters to pass to Boltz-2 that are not externalised in Nextflow.

Generation of scaffold templates for fold conditioning of binders

To assist users with performing fold conditioning of binders, we compiled structural templates of minibinders from published datasets[21,25,26] and generated ~23,000 PyTorch files directly compatible with RFdiffusion using a parallelised version of the 'make_secstruc_adj.py' script from the RFdiffusion GitHub (https://github.com/RosettaCommons/RFdiffusion) (Figure 4).

Benchmarking and efficiency

We evaluated parallel scaling efficiency of the ProteinDJ pipeline using either 1, 2, 4, and 8 NVIDIA A30 GPUs on our SLURM HPC system. All executions used the maximum amount of GPUs and RAM and used the benchmarking target PD-L1 (**Supplementary** Table 6) generating 500 designs and 8 sequences per design. Scaling metrics were extracted from Nextflow trace files, with runtime calculated from actual compute time (start to completion timestamps) excluding queue time. Parallel

efficiency was calculated as $(T_1/ (N \times T_n)) \times 100\%$, where T_1 = single-GPU runtime, N = GPU count, and T_n = N-GPU runtime. ProteinMPNN was used as the sequence design method and AF2 was used for structure prediction and validation.

To compare the efficiency of each software package, we generated 100 folds of *de novo* binders against PD-L1 with a sequence length of 50-100 residues using RFdiffusion and provided these to either ProteinMPNN, with and without a FastRelax cycle, or Full-Atom MPNN, generating 10 sequences per fold. All sequences were passed to either AlphaFold2 or Boltz-2 for prediction. ProteinMPNN was executed on nodes with 2.60GHz Intel Xeon E5-2690 CPUs and FAMPNN, AlphaFold2 and Boltz-2 utilised 4 NVIDIA A30 GPUs. The total runtime excluding queue time was divided by the number of designs.

Comparison of sequence design methods

RFdiffusion in ProteinDJ was used to generate 100 folds for each target protein using previously described hotspots and contigs (**Supplementary** Table 6). The noise scale for diffusion was set to zero for all runs. Using Bindsweeper, the folds were provided to ProteinMPNN or FAMPNN, followed by AlphaFold2 Initial Guess via the rfd_skip path in ProteinDJ. The sequence design temperature that was set to 0.0001 in ProteinMPNN and FAMPNN for consistency. When using FastRelax, only one cycle was performed with one sequence per cycle. A design was considered successful if it passed three filters after prediction using AlphaFold2 Initial Guess: af2_max_rmsd_binder_bndaln $\leq$ 1, af2_max_pae_interaction $\leq$ 10, and af2_min_plddt_total $\geq$ 80 (see Supplementary Table 2 for definitions).

Availability of code and data

ProteinDJ code and benchmarking data is available at https://github.com/PapenfussLab/proteindj. An overview of all software dependencies and availability can be found in Supplementary Table 1.

Acknowledgments

Development and evaluation of software was performed on the Walter and Eliza Hall Institute (WEHI) Milton HPC infrastructure. We would like to acknowledge the support and assistances of the WEHI Research Computing Platform. D.S. was supported by a WEHI Alan W Harris Honours Scholarship. J.M.H. was supported by an Australian National Health and Medical Research Council (NHMRC) EL1 Investigator Grant (2008096). A.T.P. was supported by an NHMRC Investigator Grant (2026643) and funding from the Lorenzo and Pamela Galli Medical Research Trust. This work was made possible through Victorian State Government Operational Support Program and the Australian Government NHMRC Independent Research Institutes Infrastructure Support Scheme to WEHI. A.P.T, I.S.L, and J.M.H. are members of the Australian Research Council Industrial Transformation Training Centre for Cryo-Electron Microscopy of Membrane Proteins for Drug Discovery (IC200100052). The contents of this published material are solely the responsibility of the individual authors and do not reflect the views of the NHMRC or funding partners.

Author contributions

Dylan Silke: Conceptualization (equal); investigation (supporting); software (equal); visualization (supporting); writing – original draft (equal); writing – review and editing (equal).

Julie Iskander: Software (supporting); writing - review and editing (equal).

Junqi Pan: Software (supporting); visualization (supporting); writing - review and editing (equal).

Andrew P. Thompson; Supervision (supporting); writing - review and editing (equal).

Anthony T. Papenfuss; Supervision (equal); funding acquisition (equal); writing - review and editing (equal).

Isabelle S. Lucet; Supervision (equal); funding acquisition (equal); writing - review and editing (equal).

Joshua M. Hardy; Conceptualization (equal); investigation (lead); software (equal); supervision (equal); visualization (lead); writing – original draft (equal); writing - review and editing (equal).

Competing interests

The authors declare no conflicts of interest.

Supplementary Material

Supplementary Table 1 - The containers and software dependencies of ProteinDJ.

Container	Description	Repository
af2	A modified version of AlphaFold2 (Initial Guess) as implemented in dl_binder_design. Used for structure prediction and design validation.	https://github.com/PapenfussLab/dl_binder_design.git
boltz2	Structure prediction design validation and filtering using Boltz-2 (inference only).	https://github.com/jwohlwend/boltz.git
dl_binder_design	Environment for ProteinMPNN FastRelax used for sequence design. Includes 'vanilla' and 'soluble' checkpoint models and implementation of Rosetta FastRelax Protocol	https://github.com/PapenfussLab/dl_binder_design.git
fampnn	Environment for Full-Atom MPNN, a sequence design tool.	https://github.com/PapenfussLab/fampnn.git
pyrosetta_tools	Python interface to Rosetta molecular modeling suite (PyRosetta) bundled with standard data analysis libraries (BioPython, NumPy, Pandas, Matplotlib) for processing pipeline results.	N/A
rfdiffusion	Contains RFdiffusion, the fold design software integral to the pipeline	https://github.com/PapenfussLab/RFdiffusion

Supplementary Table 2 – The filtering parameters available in ProteinDJ.

Parameter	Description
RFdiffusion	
rfd_min_helices	Minimum number of alpha-helices required.
rfd_max_helices	Maximum number of alpha-helices allowed.
rfd_min_strands	Minimum number of beta-strands required.
rfd_max_strands	Maximum number of beta-strands allowed.
rfd_min_ss	Minimum number of secondary structure elements (α-helices + β-strands).
rfd_max_ss	Maximum number of secondary structure elements (α-helices + β-strands).
rfd_min_rog	Minimum radius of gyration (Å).
rfd_max_rog	Maximum radius of gyration (Å).
ProteinMPNN	
mpnn_max_score	Maximum MPNN score (negative log likelihood).
FAMPNN	
fampnn_max_psce	Max predicted-side chain error (PSCE) score for designed side-chains.
AlphaFold2 Initial Guess	
af2_max_pae_interaction	Max predicted aligned error for interactions
af2_max_pae_overall	Max predicted aligned error for all chains
af2_max_pae_binder	Max predicted aligned error for binder
af2_max_pae_target	Max predicted aligned error for target
af2_min_plddt_overall	Max C-alpha RMSD of binder when binder chains are aligned
af2_min_plddt_binder	Max C-alpha RMSD of binder when target chains are aligned
af2_min_plddt_target	Max C-alpha RMSD of target when target chains are aligned
af2_max_rmsd_overall	Max C-alpha RMSD when all chains are aligned
af2_max_rmsd_binder_bndaln	Max C-alpha RMSD of binder when binder chains are aligned
af2_max_rmsd_binder_tgtaln	Max C-alpha RMSD of binder when target chains are aligned
af2_max_rmsd_target	Max C-alpha RMSD of target when target chains are aligned
Boltz-2	
boltz_max_overall_rmsd	Max C-alpha RMSD between all chains of Boltz-2 prediction and RFD design
boltz_max_binder_rmsd	Max C-alpha RMSD between binder chains of Boltz-2 prediction and RFD design.
boltz_max_target_rmsd	Max C-alpha RMSD between target chains of Boltz-2 prediction and RFD design.
boltz_min_conf_score	Minimum confidence score of the prediction
boltz_min_ptm	Minimum predicted template modelling (pTM) score of the prediction
boltz_min_ptm_interface	Minimum pTM-score of the prediction interface
boltz_min_plddt	Minimum pLDDT score of the prediction
boltz_min_plddt_interface	Minimum pLDDT score of the prediction interface
boltz_max_pde	Maximum predicted distance error of the prediction
boltz_max_pde_interface	Maximum predicted distance error of the prediction interface

Supplementary Table 3 – ProteinDJ outputs a wide array of additional metrics to assist researchers to evaluate potential designs. Some metrics are calculated by software in the pipeline (+), but there are additional metrics calculated using PyRosetta[18] and BioPython[19].

Metric name	Explanation
description	Unique filename of structure prediction
fold_id	Fold ID for independent execution of RFdiffusion
seq_id	Sequence ID for independent execution of ProteinMPNN
rfd_sampled_mask+	Contigs used by RFdiffusion to produce this fold (e.g., ['E6-155/0', '100-100'])
rfd_helices	Number of α-helices in RFdiffusion fold
rfd_strands	Number of β-strands in RFdiffusion fold
rfd_total_ss	Total secondary structures (helices + strands) in RFdiffusion fold
rfd_RoG	Radius of gyration (RoG) for RFdiffusion fold (Å)
rfd_time	Time (seconds) taken by RFdiffusion to produce a fold
fampnn_avg_psce	Average predicted sidechain error (PSCE) from FAMPNN
mpnn_score+	ProteinMPNN's average negative log likelihood
af2_pae_interaction+	AF2 PAE for binding interface residue pairs (Å)
af2_pae_overall	AF2 PAE for all residue positions (Å)
af2_pae_binder+	AF2 predicted aligned error (PAE) for binder residue positions (Å)
af2_pae_target+	AF2 PAE for target protein residue positions (Å)
af2_plddt_overall+	Average predicted Local Distance Difference Test (pLDDT) score for all residues
af2_plddt_binder+	Average pLDDT score for binder residues
af2_plddt_target+	Average pLDDT score for target residues
af2_rmsd_overall	C-alpha RMSD comparing RFdiffusion design and AF2 prediction (all chains).
af2_rmsd_binder_tgtaln+	C-alpha RMSD between RFD binder and AF2 binder when target chain is aligned (Å)
af2_rmsd_binder_bndaln+	C-alpha RMSD between RFD binder and AF2 binder when binder chain is aligned (Å)
af2_rmsd_target	C-alpha RMSD between RFD target and AF2 target when binder chain is aligned (Å)
af2_time+	Time (seconds) for AlphaFold2 prediction
boltz_overall_rmsd	C-alpha RMSD between all chains of Boltz-2 prediction and RFD design (Å)
boltz_binder_rmsd	C-alpha RMSD between binder chains of Boltz-2 prediction and RFD design (Å)
boltz_target_rmsd	C-alpha RMSD between target chains of Boltz-2 prediction and RFD design (Å)
boltz_conf_score+	Confidence score for Boltz-2 prediction
boltz_ptm+	The predicted template modelling (pTM) score for the Boltz-2 prediction
boltz_ptm_interface+	Interface pTM-score for Boltz-2 prediction
boltz_plddt+	pLDDT score for Boltz-2 prediction
boltz_plddt_interface+	Interface pLDDT score for Boltz-2 prediction of the complex
boltz_pde+	Predicted distance error (PDE) for Boltz-2 prediction
boltz_pde_interface+	Interface PDE for Boltz-2 prediction of the complex (Å)
pr_helices	Number of α-helices in predicted structure
pr_strands	Number of β-strands in predicted structure
pr_total_ss	Total secondary structures (helices + strands) in predicted structure
pr_RoG	RoG of predicted structure (Å)
pr_intface_BSA	Buried surface area (BSA) at binding interface ($Å^2$)
pr_intface_shpcomp	Interface shape complementarity (0-1, 1=optimal)
pr_intface_hbonds	Number of interface hydrogen bonds
pr_intface_deltaG	Solvation free energy gain (delta G) at interface (kcal/mol)
pr_intface_packstat	Interface packing quality (0-1, higher=better)

pr_TEM	Rosetta Total Energy Metric (TEM) score of design (lower=more stable)
pr_surfhphobics_%	Percentage of hydrophobic residues on design surface
seq_ext_coef	Extinction coefficient ($M^{-1}cm^{-1}$) at 280nm
seq_length	Number of amino acids in design
seq_MW	Molecular weight (MW) of design (Da)
seq_pI	Isoelectric point (pI) of design
sequence	Amino acid sequence of design

+Existing metrics from RFdiffusion, ProteinMPNN-Fast Relax, AlphaFold 2 Initial Guess and Boltz-2

Supplementary Table 4 – Parameters for RFdiffusion used for different modes. Parameters are ignored if they are not relevant to the design type or cause conflicts.

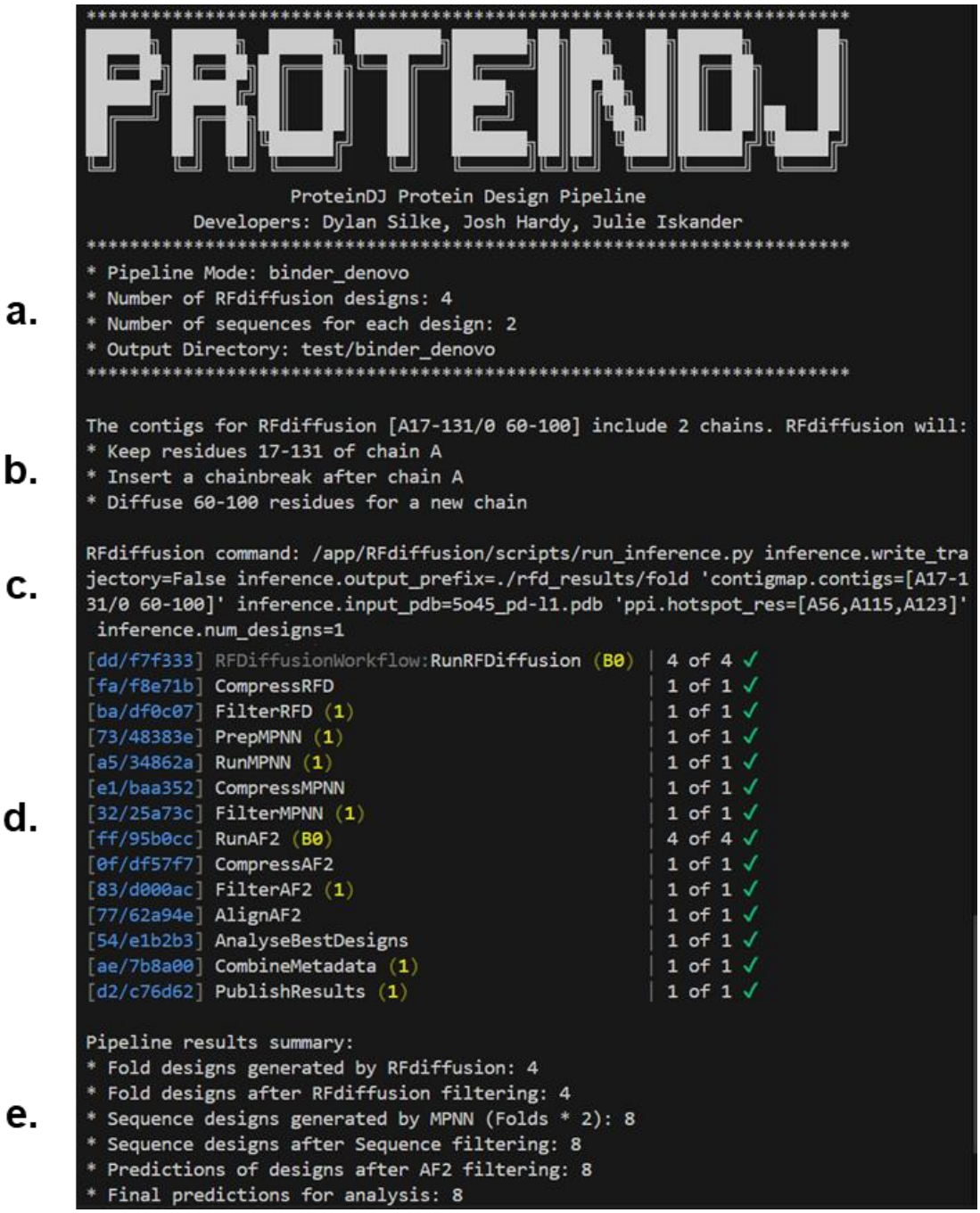
Parameter	**monomer denovo**	**monomer foldcond**	**monomer motifscaff**	**monomer partialdiff**	**binder denovo**	**binder foldcond**	**binder motifscaff**	**binder partialdiff**
rfd_contigs	**Required**	Ignored	**Required**	**Required**	**Required**	Ignored	**Required**	**Required**
rfd_input_pdb	Ignored	Ignored	**Required**	**Required**	**Required**	**Required**	**Required**	**Required**
rfd_hotspots	Ignored	Ignored	Ignored	Ignored	*Optional*	*Optional*	Ignored	Ignored
rfd_scaffold_dir	Ignored	**Required**	Ignored	Ignored	Ignored	**Required**	Ignored	Ignored
rfd_target_adj	Ignored	Ignored	Ignored	Ignored	Ignored	**Required**	Ignored	Ignored
rfd_target_ss	Ignored	Ignored	Ignored	Ignored	Ignored	**Required**	Ignored	Ignored
rfd_mask_loops	Ignored	*Optional*	Ignored	Ignored	Ignored	*Optional*	Ignored	Ignored
rfd_inpaint_seq	Ignored	Ignored	*Optional*	Ignored	Ignored	Ignored	*Optional*	Ignored
rfd_length	Ignored	Ignored	*Optional*	Ignored	Ignored	Ignored	*Optional*	Ignored
rfd_partial_diffusion_timesteps	Ignored	Ignored	Ignored	**Required**	Ignored	Ignored	Ignored	**Required**
rfd_ckpt_override	*Optional*	*Optional*	*Optional*	*Optional*	*Optional*	*Optional*	*Optional*	*Optional*
rfd_noise_scale	*Optional*	*Optional*	*Optional*	*Optional*	*Optional*	*Optional*	*Optional*	*Optional*

Supplementary Table 5 - Scaffold templates for fold conditioning

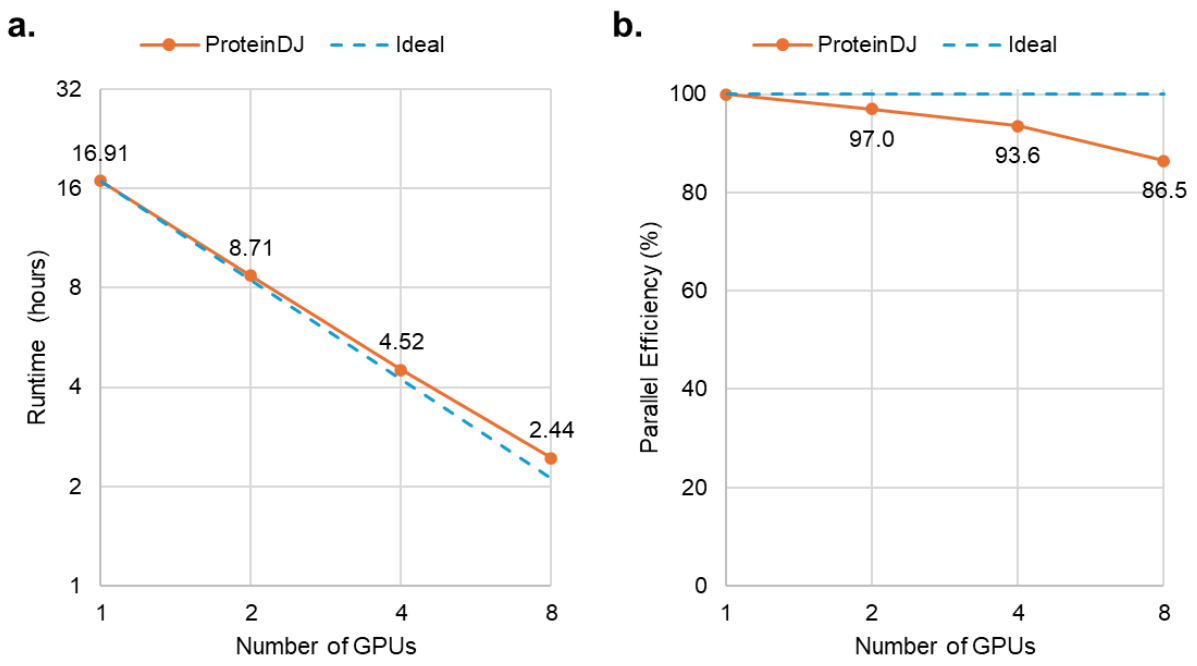
Prefix	Description	Count
HHH	"3-helical bundles"	17668
HHHH	"4-helical bundles"	3499
HEEHE	"Mixed alpha-helical (H) and beta-strand (E) scaffold. Artificial."	208
EHEEHE	"Mixed alpha-helical (H) and beta-strand (E) scaffold based on ferredoxin"	1555

Supplementary Table 6 – Benchmarking target structures. Structures were prepared including the same residues as used for RFdiffusion benchmarking in Watson et al.[5]. The contigs and hotspots provided to RFdiffusion are indicated below.

Protein (Acronym)	Biological function	PDB ID	RFdiffusion contigs	RFdiffusion hotspots
Influenza A H1 haemagglutinin (HA)	Viral surface protein for host cell attachment and entry.	5VLI	[A4-53/A79-83/A110-114/A261-325/0 B501-568/B580-670/0 50-100]	B521, B545, B552
Interleukin-7 receptor alpha (IL-7Rα)	Receptor for IL-7, important for lymphocyte development.	3DI3	[B17-209/0 50-100]	B58, B80, B139
Insulin Receptor (InsR)	Insulin receptor, regulates glucose uptake and metabolism.	4ZXB	[E6-155/0 50-100]	E64, E88, E96
Programmed cell death 1 ligand 1 (PD-L1)	Immune checkpoint, protein helps regulate immune responses.	5O45	[A17-131/0 50-100]	A56, A115, A123
Tropomyosin receptor kinase A (TrkA)	Receptor for nerve growth factor, promotes neuron survival and growth.	1WWW	[X282-382/0 50-100]	X294, X296, X333



Supplementary Figure 1 – Example execution of ProteinDJ pipeline for *de novo* binder design. a. Header with key parameters and output directory. **b.** Description of contigs provided and how these will be used in diffusion. **c.** The command that will be used for launching instances of RFdiffusion. **d.** Standard Nextflow process tracker that updates as each task completes. **e.** A summary of the run, highlighting success rates at each stage.



Supplementary Figure 2 – ProteinDJ multi-GPU efficiency and scaling. (a) Each data point represents the total wall-clock time excluding queue time for a complete ProteinDJ pipeline execution (4,000 designs) using the specified number of NVIDIA A30 GPUs. (b) Parallel efficiency of ProteinDJ calculated relative to single GPU execution. Each data point represents the parallel efficiency percentage calculated from wall-clock execution times.

References

1 Nada, H., Choi, Y. *et al.* New insights into protein-protein interaction modulators in drug discovery and therapeutic advance. *Signal Transduct Target Ther* **9**, 341, (2024).

2 The Royal Swedish Academy of Sciences. *The Nobel Prize in Chemistry 2024*, <https://www.nobelprize.org/prizes/chemistry/2024/press-release/> (August 27, 2025).

3 Watson, J. L., Juergens, D. *et al.* De novo design of protein structure and function with RFdiffusion. *Nature* **620**, 1089-1100, (2023).

4 Dauparas, J., Anishchenko, I. *et al.* Robust deep learning-based protein sequence design using ProteinMPNN. *Science* **378**, 49-56, (2022).

5 Bennett, N. R., Coventry, B. *et al.* Improving de novo protein binder design with deep learning. *Nat Commun* **14**, 2625, (2023).

6 Jumper, J., Evans, R. *et al.* Highly accurate protein structure prediction with AlphaFold. *Nature* **596**, 583-589, (2021).

7 Pacesa, M., Nickel, L. *et al.* One-shot design of functional protein binders with BindCraft. *Nature*, (2025).

8 Zambaldi, V., La, D. *et al.* De novo design of high-affinity protein binders with AlphaProteo. *arXiv*, (2024).

9 Passaro, S., Corso, G. *et al.* Boltz-2: Towards Accurate and Efficient Binding Affinity Prediction. *bioRxiv*, (2025).

10 Zheng, H., Lin, H. *et al.* AlphaFold3 in Drug Discovery: A Comprehensive Assessment of Capabilities, Limitations, and Applications. *bioRxiv*, 2025.2004.2007.647682, (2025).

11 Boitreaud, J., Dent, J. *et al.* Chai-1: Decoding the molecular interactions of life. *BioRxiv*, (2024).

12 Sappington, I., Toul, M. *et al.* Improved protein binder design using beta-pairing targeted RFdiffusion. *BioRxiv*, (2024).

13 Vazquez Torres, S., Benard Valle, M. *et al.* De novo designed proteins neutralize lethal snake venom toxins. *Nature* **639**, 225-231, (2025).

14 Di Tommaso, P., Chatzou, M. *et al.* Nextflow enables reproducible computational workflows. *Nat. Biotechnol.* **35**, 316-319, (2017).

15 Kurtzer, G. M., Sochat, V. & Bauer, M. W. Singularity: Scientific containers for mobility of compute. *PLoS One* **12**, e0177459, (2017).

16 Shuai, R. W., Widatalla, T., Huang, P.-S. & Hie, B. L. Sidechain conditioning and modeling for full-atom protein sequence design with FAMPNN. *BioRxiv*, (2025).

17 Goverde, C. A., Pacesa, M. *et al.* Computational design of soluble and functional membrane protein analogues. *Nature* **631**, 449-458, (2024).

18 Chaudhury, S., Lyskov, S. & Gray, J. J. PyRosetta: a script-based interface for implementing molecular modeling algorithms using Rosetta. *Bioinformatics* **26**, 689-691, (2010).

19 Cock, P. J., Antao, T. *et al.* Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* **25**, 1422-1423, (2009).

20 Goddard, T. D., Huang, C. C. *et al.* UCSF ChimeraX: Meeting modern challenges in visualization and analysis. *Protein Sci.* **27**, 14-25, (2018).

21 Cao, L., Coventry, B. *et al.* Design of protein-binding proteins from the target structure alone. *Nature* **605**, 551-560, (2022).
22 Overath, M. D., Rygaard, A. S. H. *et al.* Predicting Experimental Success in De Novo Binder Design: A Meta-Analysis of 3,766 Experimentally Characterised Binders. *BioRxiv*, (2025).
23 Gustafsson, O. J. R., Al Bkhetan, Z., Francis, R. & Manos, S. Enabling national step changes in bioinformatics through ABLeS, the Australian BioCommons Leadership Share. (2023).
24 Harris, C. R., Millman, K. J. *et al.* Array programming with NumPy. *Nature* **585**, 357-362, (2020).
25 Maguire, J. B., Haddox, H. K. *et al.* Perturbing the energy landscape for improved packing during computational protein design. *Proteins* **89**, 436-449, (2021).
26 Linsky, T. W., Noble, K. *et al.* Sampling of structure and sequence space of small protein folds. *Nat Commun* **13**, 7151, (2022).