Benchmarking AlphaFold2-RosettaRelax for Predicting the Impact of Indel Variants in DHFR

Student: Heleen Lanters – 0757624

Date: 24-05-2024

Bioinformatics profile internship as part of the master Molecular and Cellular Life Sciences at Utrecht University

Internship performed at the RNA and Computational Biology group of Amelie Stein at the University of Copenhagen.

Daily Supervisor: Marion Silverstini

Examiner: Amelie Stein

Examiner Utrecht University: Basten Snoek

Layman's summary

This project focused on using a computational method to predict how certain types of mutations, known as insertions and deletions (indels), affect proteins. Indels involve either adding or removing one or more amino acids (the building blocks of proteins), which can change the protein's structure and function. Understanding these changes is important because indels are common in the human genome and can lead to diseases. Additionally, knowing how indels affect proteins can help in designing new proteins for medical and industrial purposes. The computational method we used combines two advanced tools: AlphaFold2 (AF2) and RosettaRelax (RR). AlphaFold2 is a program that predicts the 3D structure of a protein from its amino acid sequence, while RosettaRelax refines these structures further and calculates how stable they are. We used the protein dihydrofolate reductase (DHFR) in our study because it is small, well-studied, and essential for cells. We created 374 DHFR variants, with 187 having a single amino acid added and 187 having one removed. Using AF2, we predicted the 3D structures of these variants. Then, we applied RR to refine these structures and calculate their stability scores. To check if our predictions were accurate, we compared them with experimental data obtained using a technique called CPOP. CPOP measures the actual folding stability of proteins in a lab, giving us a benchmark to compare our computational results against. Our findings showed that the AF2-RR method could predict the effects of indels with reasonable accuracy, making it a promising tool for further development. However, we found that the method struggled to accurately predict instability caused by indels in the middle of the DHFR protein. This suggests that improvements are needed, possibly by including more structural or functional data. Interestingly, we discovered that indels in secondary structures (like helices and sheets) were generally not tolerated, meaning they often made the protein unstable. In contrast, indels in less structured regions, such as loops or surface areas, were more tolerated. Finally, our study opens the door to using other advanced methods like INDELi-X, which combines information about the position of the indel in the protein with other data to make predictions. This method has shown high accuracy in other studies and could be a valuable tool for analysing DHFR and other proteins in the future. Overall, this study demonstrates the potential of combining AF2 and RR to understand how indels affect protein stability, offering insights that could advance both basic science and practical applications in protein engineering.

Abstract

In this study, we evaluated a computational method combining AlphaFold2 (AF2) and RosettaRelax (RR) to predict the impact of insertion and deletion (indel) mutations on the stability of dihydrofolate reductase (DHFR). We generated a series of DHFR variants, each containing a single amino acid insertion or deletion, and predicted their structural stability using AF2 to predict the fold and RR to refine the structures and calculate the change in free energy (ΔG). The predictions were compared with experimental data acquired with the CPOP system, to evaluate the performance of the method. The AF2 predictions included the pLDDT score to assess the confidence in the predicted structures, while the RR provided ΔG scores to estimate the stability of the variants. Additionally, we ran RR on DHFR complexed with methotrexate (MTX) to examine how the presence of this ligand affects protein stability. Our results indicate that the AF2-RR combined protocol is effective in predicting the stability changes due to indel mutations, but no significance improvement was found in accuracy when using MTX in the RR predictions. However, the method shows limitations in accurately predicting instability caused by mutations in the core region of the protein, though this might be specific to DHFR. We also observed that indels located within secondary structures such as helices or sheets were generally non-tolerated. This study highlights the utility of AF2 and RR in structural biology and provides insights into the stability of DHFR variants. Furthermore, the availability of the CPOP indel dataset allows for the testing of other computational methods, such as INDELi-X, which combines substitution scores with indel location to enhance prediction accuracy.

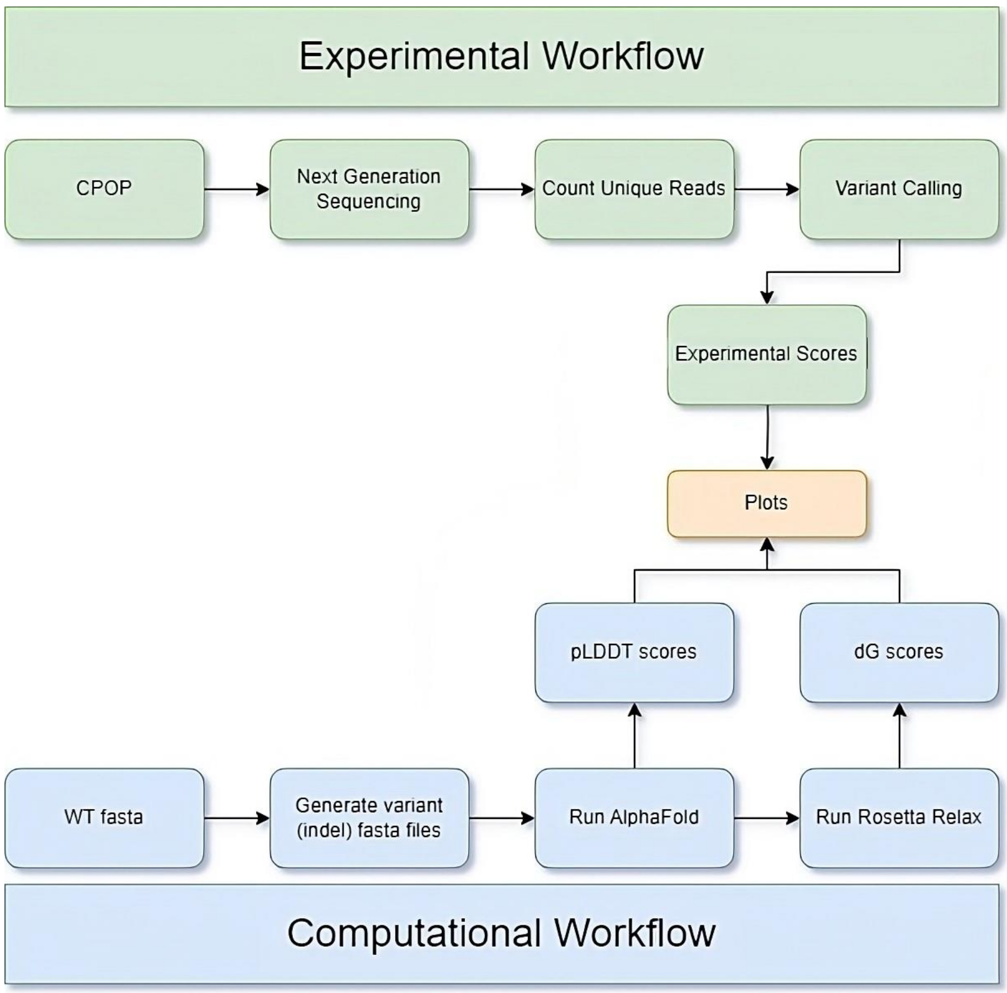

Figure 1: Graphical Abstract of the Experimental and Computational Workflow.

Table of contents

Layman's summary 1
Abstract 2
Introduction 3
Methods 7
Results & Discussion 8
Conclusion 14
Word of thanks 14
Bibliography 15
Supplements 16

Introduction

Mutations are fundamental alterations in the genetic material of an organism and lie at the heart of genetic diversity, evolutionary processes, and disease mechanisms [1] [2]. Understanding the nature, mechanisms, and consequences of mutations is essential for deciphering the workings of the genome [1] [2]. These DNA changes come in various forms, from single nucleotide substitutions to large-scale chromosomal rearrangements [1]. Among the diverse spectrum of genetic alterations, single nucleotide polymorphisms (SNPs) are the most common in the human genome [1] [3]. SNPs denote single base pair changes in the DNA sequence, and if they occur within the protein-coding region of a gene, they can lead to an amino acid (AA) change, also known as a missense mutation. The second most frequent mutation type in the human genome are insertions and deletions (indels), characterised by the addition or removal of one or more nucleotide bases within the DNA sequence [3]. Approximately 22% of mutations associated with diseases are attributed to indels [4]. These mutations can vary in size, ranging from a single base pair to larger sequences, and can occur in coding or non-coding regions [3]. Indels represent a distinct type of mutation compared to missense mutations [5]. While missense mutations only alter the side chain of a protein, indels modify the backbone of the protein, thereby changing its length [6]. Therefore indels can significantly impact protein structure and function, as they can lead to frameshift mutations, alter reading frames, and affect gene expression and protein functionality [5].

In this project, we focus on a specific subtype of indels: the insertion or deletion of three basepairs, resulting in the addition or removal of exactly one amino acid (AA) without causing a frameshift in the protein. These small single AA indels can have profound effects on protein structure and function. While indels are generally considered more deleterious than missense mutations, they may also offer novel functionalities that missense mutations cannot provide [5]. This potential for novel functionality makes indels particularly intriguing for industries seeking to optimise enzyme stability or activity. They present new opportunities for enhancing protein function that may not be achievable through missense mutations [6]. However, indels are poorly understood and inadequately studied, primarily due to a lack of systematic indel data sets [7]. A significant challenge is the scarcity of structural indel data, as indels often destabilise protein structures [7]. Experiments to study indels are time-consuming and expensive, underscoring the need for a computational method to model the effects of single AA indels. Such a method would be beneficial in advancing protein research [6] [7].

Currently there are several computational variant effect predictors (VEPs) available to model the effects of indels, however, none of them were specifically developed for indels [8]. The relatively poor performance of these VEPs on indels is shown by Topolska *et al*., who tested three methods [8]. Topolska *et al*. created an extensive experimental dataset of nine different protein domains, with 1 AA insertions and deletions for every position. The effect of each indel was quantified with abundance protein fragment complementation assay (aPCA). [8] This data was used to benchmark the performance of the VEPs. The first is Combined Annotation Dependent Depletion (CADD), where deleteriousness is measured by comparing annotations of fixed or nearly fixed derived alleles in humans with simulated variants. This method combines different genome annotations to evaluate potential human single nucleotide variant (SNV) or small insertion/deletion (indel) events, providing a thorough assessment of variant impact throughout the genome [9]. On the indel data from Topolska et al. CADD performed very poorly on insertions (Pearson's R = -0.07) and deletions (Pearson's R = -0.08) [8]. Protein Variation Effect Analyzer (PROVEAN) is a software tool designed to anticipate the effects of AA substitutions or indels on a protein's biological function [10]. By employing a rapid computational method for pairwise sequence alignment scores, PROVEAN has generated precomputed predictions for 20 single AA substitutions and a single AA deletion at each position across all protein sequences in both human and mouse genomes [10]. For Topolska et al. PROVEAN performed better on their insertions (Pearson's R = 0.43) and deletions (Pearson's R = 0.32) [8]. Best of the three tested VEPs was ESM1b, a language model adapted to predict all ~450 million possible human missense variant effects, which was recently adapted to predict indels [11]. Topolska *et al*. noted varying scores on their nine different protein domains for indels (Pearson's R = 0.55 average, ranging from: 0.19 – 0.61) and deletions (Pearson's R = 0.58 average, ranging from: 0.27 – 0.64) [8]. The ESM1b method performed best on average on the indel data set of Topolska *et al*., however the machine learning method is difficult to apply for a non-computational researcher and the scores were inconsistent over the 9 protein domains [11] [8]. All three methods, CADD, PROVEAN, and ESM1b, show there is room for improvement when modelling the effect of indels. One of the main reasons for this lacking performance in predicting one AA indels is the lack of data for models to be trained on [7]. Which is why researchers like Topolska *et al*. are producing large indel datasets, to develop computational methods to predict the effect of the indels [8] [7].

Another research group who developed their own indel data to develop indel VEPs is Woods *et al*. They aimed to explore the impact of deletion mutations on a small α-helical sterile alpha motif (SAM) domain [7]. The SAM domain consists of 72 AAs, including 5 AA repeats, therefore there would be no difference between either one of them in the protein sequence [7]. This resulted in 65 unique deletion variants, 17 of which were classified as soluble (neutral or beneficial) variants and the rest as insoluble (deleterious) variants by biomolecular NMR and circular dichroism [7]. All 17 soluble deletion variants of the SAM domain were located in either the N-terminus, C-terminus or loop regions of the protein, where all the residues were either surface exposed or flexible regions of the protein [7]. All other deletion variants resulted in insoluble protein [7]. Woods *et al*. benchmarked four computational methods commonly used in structural modelling: (1) *de novo* folding using Rosetta, (2) RosettaCM, (3) RosettaRelax, and (4) AlphaFold2 (AF2) combined with RosettaRelax (RR) on their SAM domain deletion dataset [7]. The AlphaFold2-RosettaRelax (AF2-RR) combined protocol was best able to separate the soluble from the insoluble deletion variants [7]. Not only did they test the AF2-RR protocol on their own SAM-domain, they also applied it to a GFP and a Rincin deletion variants dataset. Figure 2 shows the plots of the pLDDT and ΔΔG scores plotted against each other where the datapoints are coloured by the experimental scores. The SAM-domain only has a few points in the wrong quadrant, the GFP scores are worse, mostly due to the high pLDDT score of ~97,3, where almost all variants score lower than that whilst being acceptable deletion variants according to the experimental data. The same is true for the Rincin data, where ΔΔG shows good separation between the active (tolerated) and inactive (non-tolerated) deletion variants, but the pLDDT does not. Figure 2 shows promising results, but how well the AF2-RR method performs remains unclear from the plots.

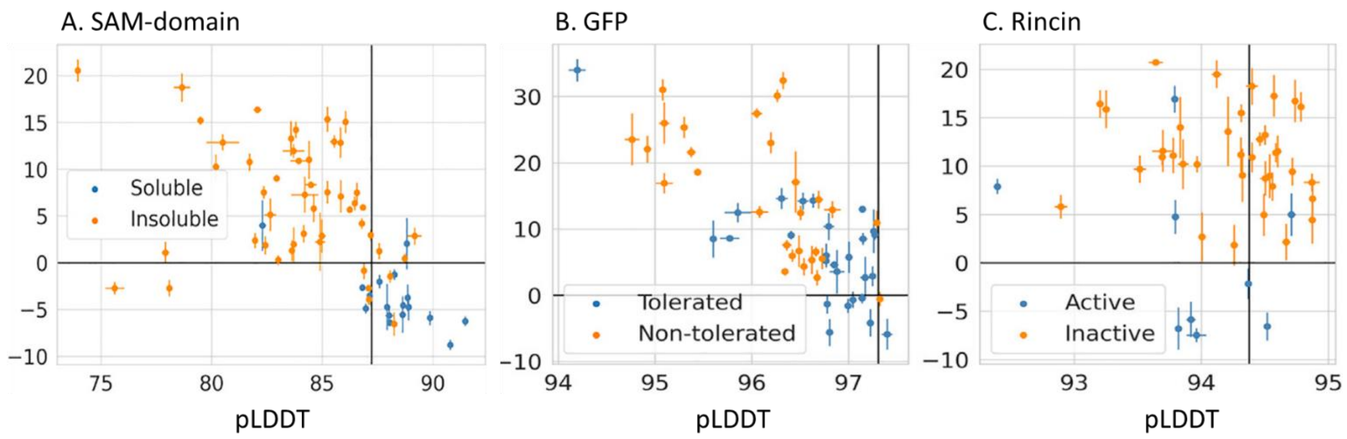

Figure 2: Woods plots of the deletion variants of the SAM-domain, GFP, and Rincin. RR ΔΔG score plotted against the AF2 pLDDT score. The black lines represent the wild type scores, meaning scores in the lower right corner are predicted as wild type like and scores in the top left corner are variants that are detrimental. The datapoints are coloured by the experimental scores, orange indicating a detrimental variant and blue a wild type-like variant [7].

AF2 is a deep learning model developed by DeepMind to predict protein structures from AA sequences [12]. Taking the AA input it searches databases to generate multiple sequence alignments (MSAs) and 3D templates [12]. The 3D templates are used together with a matrix of the input sequence to make a pair representation matrix [12]. These two matrices go into the Evoformer, a neural network, where both matrices are updated for 48 rounds [12]. The output is a refined MSA and a refined pair representation, which are used as input for the second neural network, the structure module, where the amino acids are translated and together with physical and chemical constraints, such as bonds and torsion angles, are used to produce an initial 3D protein structure [12]. The 3D structure together with the refined matrices are used as input for the Evoformer, starting the whole process over again, for three more rounds [12]. The pLDDT (predicted local-distance difference test) score provided by AF2 is a confidence measure for each residue's position within the predicted structure, with scores ranging from 0 to 100. A higher pLDDT score indicates higher confidence in the accuracy of the predicted position of that residue [12]. The average pLDDT score for every position per variant is one of the two metrics used by Woods *et al.* to predict the effect of a deletion [7]. The second metric, ΔΔG, comes from RR, computational protocol used to refine protein structures by iteratively optimizing them to achieve the most energetically favourable conformation [13]. RR improves the accuracy of predicted structures obtained from methods like AlphaFold2 by iteratively adjusting them towards the native state using van der Waals interactions, electrostatics, rotational energy, and hydrogen bonds, to name a few [13]. RosettaRelax outputs ΔG scores, which represent the Gibbs free energy of the folded protein [13]. This score reflects the stability of the protein, with lower (more negative) ΔG values indicating a more stable protein conformation [13]. To compare the stability of the indel variants to the wild type protein, ΔΔG scores were calculated [13]. ΔΔG is determined by subtracting the ΔG of the WILD TYPE protein from the ΔG of the variant [13]. This is important to be able to compare proteins. A ΔΔG score close to zero should be interpreted as the variant being as stable as the wild type [14]. A positive score means the variant is less stable than the wild type and, vice versa, a score lower than zero means the variant is more stable than the wild type [14].

In this project, we implemented the AlphaFold2-RosettaRelax method developed by Woods *et al*. to evaluate its performance using our dataset. We specifically focused on the protein dihydrofolate reductase (DHFR), a 186 AA protein, part of DNA synthesis and repair [15]. DHFR is well studied and understood essential enzyme which catalyses the NADPH dependent reduction of DHF to THF needed for purine and pyrimidine synthesis [15]. Furthermore, DHFR has an inhibitor, methotrexate (MTX), which binds to the active site, inhibiting DHFR activity but at the same stabilizing DHFR [15]. This gives the opportunity to explore a wider range of effects of the indels. For every position within the protein either a glycine 9G) was inserted or the AA was deleted. A G was chosen because it is a small neutral amino acid, enabling the study of the effect of change in protein length and not the effect of the side chain. The DHFR indel variants were evaluated with the circular permutated OPRTase (CPOP) system, a protein folding sensor (Figure 3) [16]. It makes use of the essential protein OPRTase, which de novo synthesizes the pyrimidine nucleotides like uracil [16]. The OPRTase is split in an N and C terminal domain with a short linker in between [16]. In the CPOP system the N and C terminal are switched around and the linker in between is lengthened and placed in an expression vector [16]. This has no impact on the function of the OPRTase, as tested in a yeast strain with a double knock-out of the two uracil producing genes, URA5 and URA10 [16]. A protein of interest, in this case DHFR, can be placed in the linker, if the protein of interest folds the N and C terminals of the OPRTase protein can still reach each other and form a normally functioning enzyme and therefore the double knock out yeast can grow [16]. If a protein does not fold, or does not fold properly the OPRTase is unable to form, therefore no yeast can grow [16]. All the DHFR indel variants are transformed into the CPOP system and grown in yeast. From there the DNA is extracted and sequenced with Next Generation sequencing. Per variant the reads were counted, followed by variant calling (is it an insertion or an deletion), from which the experimental scores are calculated including normalization.

In our experimental set up the CPOP experiments were performed in four different conditions: 30°C, °C + MTX, 37°C, and °C + MTX. To test the variants in a wider range of conditions. Furthermore adding MTX should stabilize DHFR, if a variant cannot be stabilized this might be due to a conformational change of the binding site. RR was run with and without taking MTX into account as a small molecule within the 3D protein structure. AF2 was run without the addition of MTX.

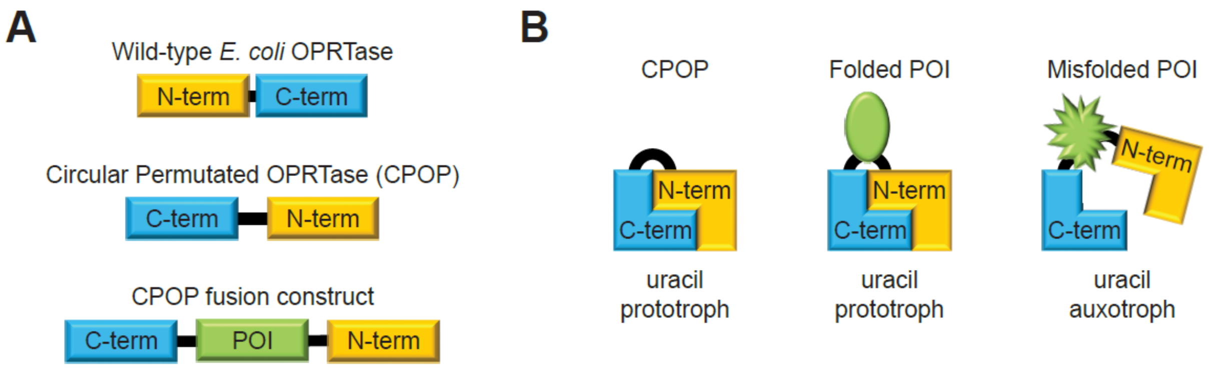

Figure 3: Schematic representation of the CPOP system.
A shows the wild type orientation of the N-terminal and C-terminal domains, followed by the switched orientation of the domains and the longer linker in the CPOP system, and finally how a protein of interested can be integrated in the linker. B shows the schematic representation of the folded state of an empty CPOP, followed by a folded CPOP with a correctly folded protein of interest in the linker, and finally a unfolded CPOP due to the unfolded protein of interest.
Image credits: Sven Larsen-Ledet

Methods

Generation of DHFR Indel Variants

The amino acid sequence of dihydrofolate reductase (DHFR) was obtained from the PDB entry 1U72 and saved in a FASTA file. For the insertion variants, a glycine (G) was inserted after every position in the DHFR sequence, resulting in one variant per insertion. Each variant was named based on the position of the amino acid after which the glycine was inserted (e.g., inserting a glycine after the 10th amino acid results in insertion variant 10). Similarly, for the deletion variants, each amino acid was individually deleted from the sequence, and the variant was named based on the position of the deleted amino acid (e.g., deleting the 10th amino acid results in deletion variant 10).

AlphaFold2 (AF2) Predictions

All indel variants were inputted into AlphaFold2 (version 2.2.4) with the following flags:

-t 2021-11-01: This sets the maximum template release date to November 1, 2021, only templates released up to this date are considered in the predictions.

-r false: This flag disables the final relaxation step, meaning the predicted models are not subjected to additional refinement post-prediction.

AF2 predicts five structures for each DHFR variant. Each position in every variant structure is assigned a pLDDT (predicted local-distance difference test) score, which represents the confidence in the accuracy of the predicted position of each residue. A mean pLDDT score is also given by AF2, an average of all the individual pLDDT position scores. The ΔpLDDT score was calculated by subtracting the wild type pLDDT score from the variant score, for easier comparison between variants. Besides the mean pLDDT a local average was also calculated around every indel position, using different window sizes. The windows at the beginning and end of the sequence were trimmed to still get a score for the variants at the edges of the sequence. The same windows were used to calculate the DHFR wild type per position score to subtract from the variant score for a ΔpLDDT per position score

RosettaRelax (RR) predictions

The output structures from AlphaFold2 were further refined using RosettaRelax (version 2021 Aug c7009b3) with the following flags:

-nstruct 10: Specifies that ten independent relaxation trajectories should be run on each input structure, resulting in 10 ΔG scores.

relax:constrain_relax_to_start_coords 1: Adds coordinate constraints to backbone heavy atoms based on the input structure, ensuring the relaxed structure remains close to the original predicted conformation.

To predict stability scores in the presence of methotrexate (MTX), a PDB file containing the DHFR and aligned MTX was used as input for RosettaRelax. A params file defining the properties of MTX as a new residue type was included. An additional flag was used to incorporate this params file into the relaxation process:

-extra_res_fa <params_file_path>: Includes the params file, ensuring that the properties of MTX are accurately represented in the computational predictions.

Per indel variant, 5 Alpha Fold structures were used as input for Rosetta Relax, producing 10 ΔG scores each, totalling 50 ΔG scores per variant. Following the Woods protocol only the top 3 ΔG scores were taken and averaged to ensure the capture of the global minimum. The same was done with the DHFR wild type scores. Subtracting the wild type ΔG score from the variant ΔG score resulted in the ΔΔG score via Equation 1 to correct for nucleotide length (+1 for insertions and -1 for deletions):

$$\Delta\Delta G = 187 \times \left(\frac{\Delta G_{indel}}{187 \pm 1}\right) - \Delta G_{wt}$$

Equation 1: calculating the ΔΔG score adjusting for the change in length of the indel variants.

ROC Curve Analysis

To assess the predictive performance of our computational methods, we utilised Receiver Operating Characteristic (ROC) curve analysis. A ROC curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. It is calculated by plotting the true positive rate (sensitivity) against the false positive rate (1-specificity) at various threshold settings. The area under the curve (AUC) provides a single metric to summarise the performance of the classifier, where an AUC of 1 indicates perfect prediction, 0.5 indicates no discriminative power.

For our analysis, the continuous CPOP data was converted to binary outcomes using a cutoff score of 0.5. Variants with scores equal to or below 0.5 were labelled as 0 (deleterious), while those with scores above 0.5 were labelled as 1 (beneficial/wild-type-like). This binary categorisation allowed us to generate ROC curves for each of the four CPOP conditions using three computational scores: pLDDT, ΔΔG, and ΔΔG + MTX. Separate ROC curves were created for both insertion and deletion variants. It is important to note that we used deduplicated scores for accuracy in benchmarking. For insertions, sequences resulting from inserting a glycine (G) before or after an already present G resulted in a duplication of the AA sequence and were therefore considered duplications and removed, leaving 174 unique variants after excluding 13 duplicates. Similarly, for deletions, sequences where the deletion of one of two or more consecutive identical amino acids resulted in duplications of the AA sequence and were excluded, leaving 176 unique variants after removing 11 duplicates.

This deduplication process ensures that our ROC curve analysis accurately reflects the predictive capability of our computational methods without bias introduced by duplicated sequences. The ROC curves and corresponding AUC values were used to evaluate and compare the performance of the computational scores under different CPOP conditions.

SSDraw: tool for plotting secondary protein stuctures

SSDraw google colab notebook [17] was used to obtain the secondary structure of DHFR. The AF2 wild type pdb was used to obtain the secondary structure figure.

Results & Discussion

In our study, we applied the AlphaFold2-RosettaRelax (AF2-RR) method developed by Woods *et al*. to evaluate its performance using our CPOP dataset of the DHFR indel variants. The computational scores were benchmarked against the experimental CPOP dataset under four different conditions. The distribution of scores is illustrated in Figure 4A-D through kernel density plots. A kernel density plot visualises data distribution over a continues interval, offering a smoothed version of a histogram. The four CPOP conditions display bimodal distributions, with one peak around or just below 1 (wild-type-like) and the other around 0 (detrimental). Notably, the distribution for CPOP at 37°C (Figure 4C) differs significantly, a data shift towards the detrimental end. This shift indicates that DHFR is more unstable at 37°C. However, upon adding MTX, the familiar bimodal peaks reappear, suggesting that MTX stabilizes the DHFR protein (Figure 4D). This stabilising effect of MTX is MTX is also evident when comparing the conditions at 30°C and 30°C with MTX, where the peak just below 1 becomes more prominent than the peak around 0 for both insertion and deletion variants (Figure 4B). In contrast, the dual distribution pattern observed in the CPOP conditions disappears in the computational scores (Figure 4E-G). The pLDDT scores predicted by AF2 (Figure 4E) show that all deletion scores are above

92, and the insertion scores are above 86, indicating high confidence in all variants by AF2. The majority of the RR ΔΔG scores cluster around or just above the DHFR wilt type score of 0, suggesting stability comparable to the wild type (Figure 4F). The RR scores with MTX are even higher, with both peaks for insertions and deletions falling below the wild type score of 0 (Figure 4G), clearly demonstrating that MTX stabilises DHFR. Critically, while the computational methods show a high confidence in the predicted structures and a general trend of stabilisation by MTX, they fail to capture the detailed instability observed experimentally at 37°C without MTX. This discrepancy highlights a potential limitation in the predictive power of the AF2-RR method under certain conditions, suggesting that further refinement and additional data may be necessary to improve its accuracy.

Next, we evaluated the distribution of the scores by plotting them in a heatmap and comparing them to the secondary structure of DHFR, as shown in Figure 5. Every column is a variant, every row a different score (pLDDT, ΔΔG, ΔΔG + MTX, and the four CPOP conditions). The computational scores align with the secondary structure, meaning a low pLDDT score and a high ΔΔG or ΔΔG + MTX score, often falls within a α-helix or β-sheet, confirming the findings of Woods *et al*. This is also true for the CPOP scores, however, they also have a low scoring block in the middle of the protein, from position ~47 to ~125 for both insertions and deletions and all four conditions. This middle block, where no indels are tolerated, is not detected by AF2 or RR. The intolerance to indels in this central region of the DHFR protein is intriguing and warrants further investigation. It may be specific to DHFR and related to its unique folding and functional mechanisms. This discrepancy highlights a potential area for improvement in our computational models, suggesting that additional structural or functional data might be necessary to enhance their predictive accuracy. If it is found that the middle of a protein generally does not accept indels, the AF2-RR method would benefit from an additional machine learning step to improve predictive power.

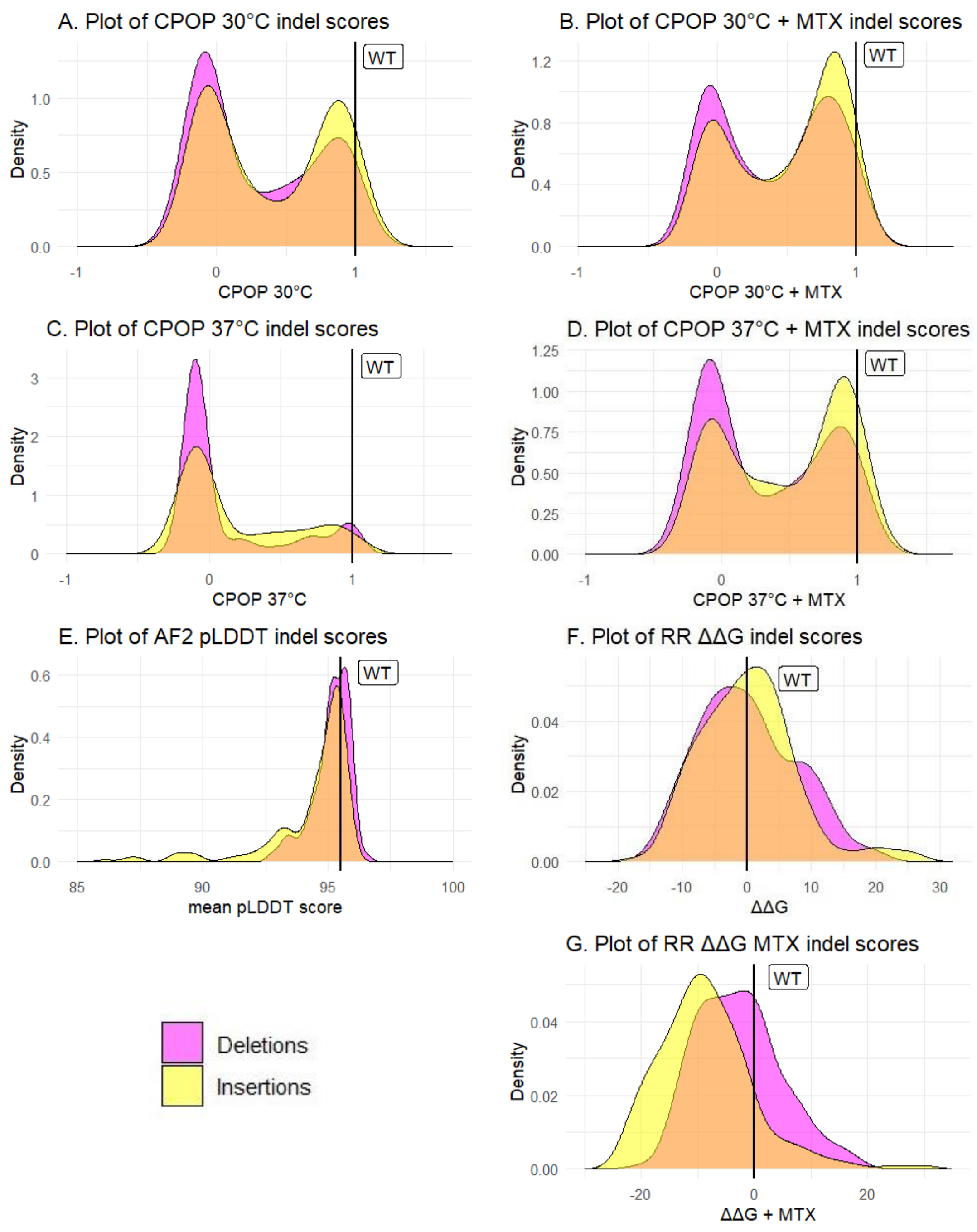

Figure 4: Kernel density distributions of the DHFR indel variant scores from CPOP, AlphaFold2, and RosettaRelax. A. CPOP 30°C scores: The distribution shows a clear separation between variants that are close to wild-type and those that are detrimental. B. CPOP 30°C + MTX scores: The stabilising effect of MTX on DHFR shifts more scores closer to 1, indicating increased stability. C. CPOP 37°C scores: At the higher temperature, most variants are detrimental, as indicated by the concentration of scores around 0. D. CPOP 37°C + MTX scores: MTX stabilises the variants, particularly the insertion variants, resulting in a distribution with more scores closer to 1. E. AlphaFold2 mean pLDDT scores: While insertion scores show a slight tail towards lower values, all scores remain high, indicating high confidence in the predicted structures. F. RosettaRelax ΔΔG scores: Similar to the pLDDT scores, the ΔΔG scores are generally close to WILD TYPE, indicating stability. G. RosettaRelax ΔΔG + MTX scores: Compared to the ΔΔG scores without MTX, these scores are shifted towards more negative values, reflecting increased stability with MTX.

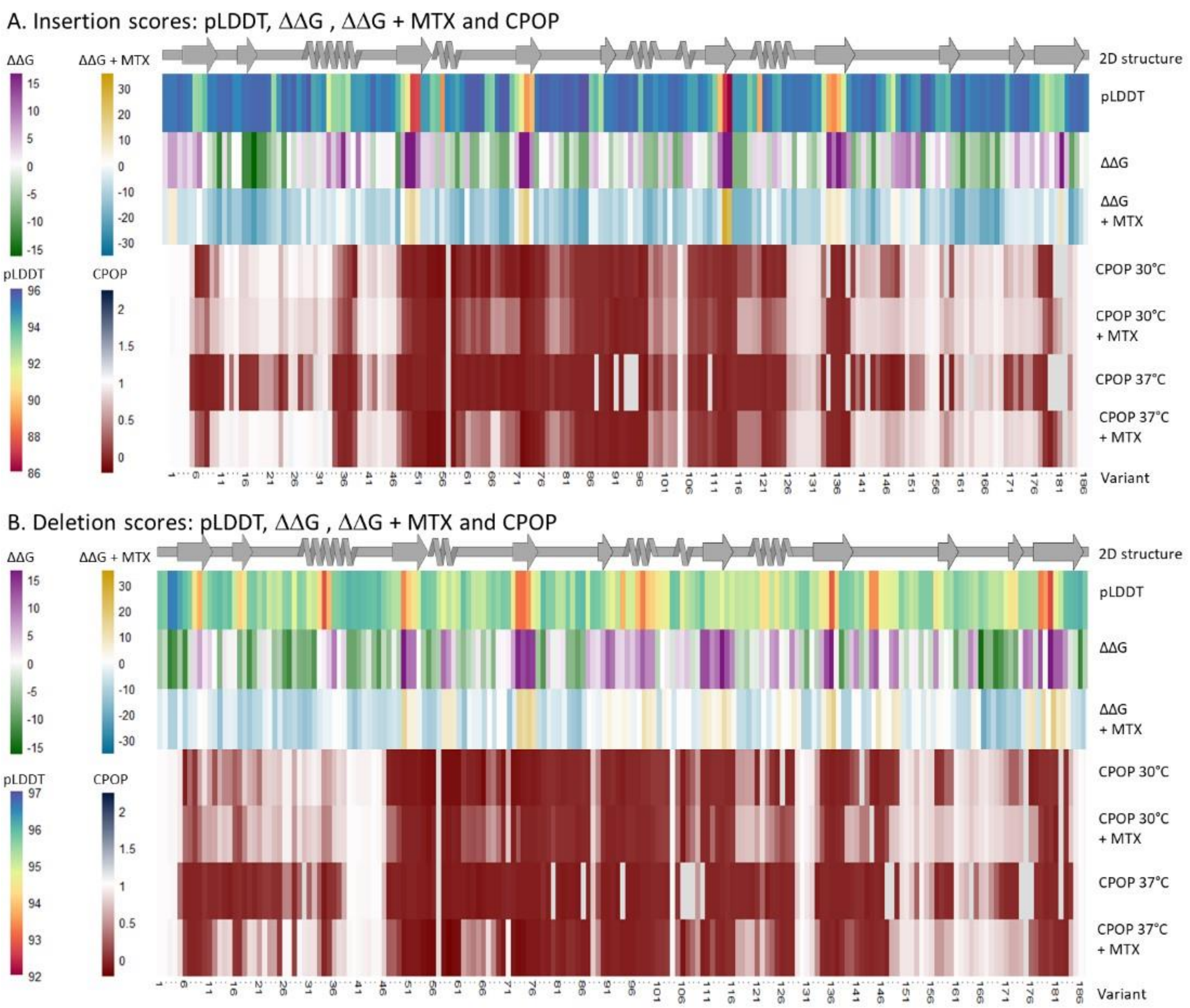

Figure 5: Secondary structure of DHFR and heatmaps of the indel scores: pLDDT, ΔΔG, ΔΔG + MTX, and CPOP. Variants are compared to the secondary structure of DHFR for the different scores available.
A. Insertion scores: Lower scores align with the secondary structure elements of DHFR, indicating regions where insertions are less tolerated. B. Deletion scores: Similar to insertion scores, lower scores align with the secondary structure elements, indicating regions where deletions are less tolerated.

A more detailed analysis of the pLDDT scores per amino acid position and per variant can provide deeper insights into the score composition. Figure 6 shows per position heatmaps for both insertion (A) and deletion (B) variants, where each row represents a variant and each column corresponds to an amino acid position in DHFR. The secondary structure of the protein is displayed on both axes. Several observations can be made from these per position heatmaps, which apply to both insertion and deletion variants:
1. Terminal Regions: The N-terminal and C-terminal amino acids consistently receive low confidence scores from AF2. This is due to these regions being exposed and flexible, lacking a defined structure.
2. Loops/Unstructured Regions: Similar to the terminal regions, loops and other unstructured regions are also assigned lower pLDDT scores, regardless of the indel position. This is indicated by the vertical red lines aligning with the unstructured regions of DHFR between helices and sheets. This observation

holds true for the wild-type DHFR AF prediction, visualised at position zero.
3. Secondary Structure Elements: In contrast, helices and sheets tend to have higher pLDDT scores. Variants score lower overall if the indel is within a helix or sheet, as indicated by the horizontal lines in the heatmap.
4. Indel Impact Zone: A diagonal red line of lower pLDDT scores indicates that AF2 is less confident in its prediction around and including the indel position, regardless of the indel's location.
5. Scattered Low Confidence Spots: Both insertion and deletion variants exhibit scattered spots of lower confidence scores across the heatmap. Initially, it was hypothesised that these spots might correspond to regions in close 3D proximity to the indel, as AF2 might be less confident in modelling amino acids in contact with the indel. However, this hypothesis was not supported by the data. The scattered spots may be specific to DHFR, and further evaluation of additional proteins is needed to draw more general conclusions.

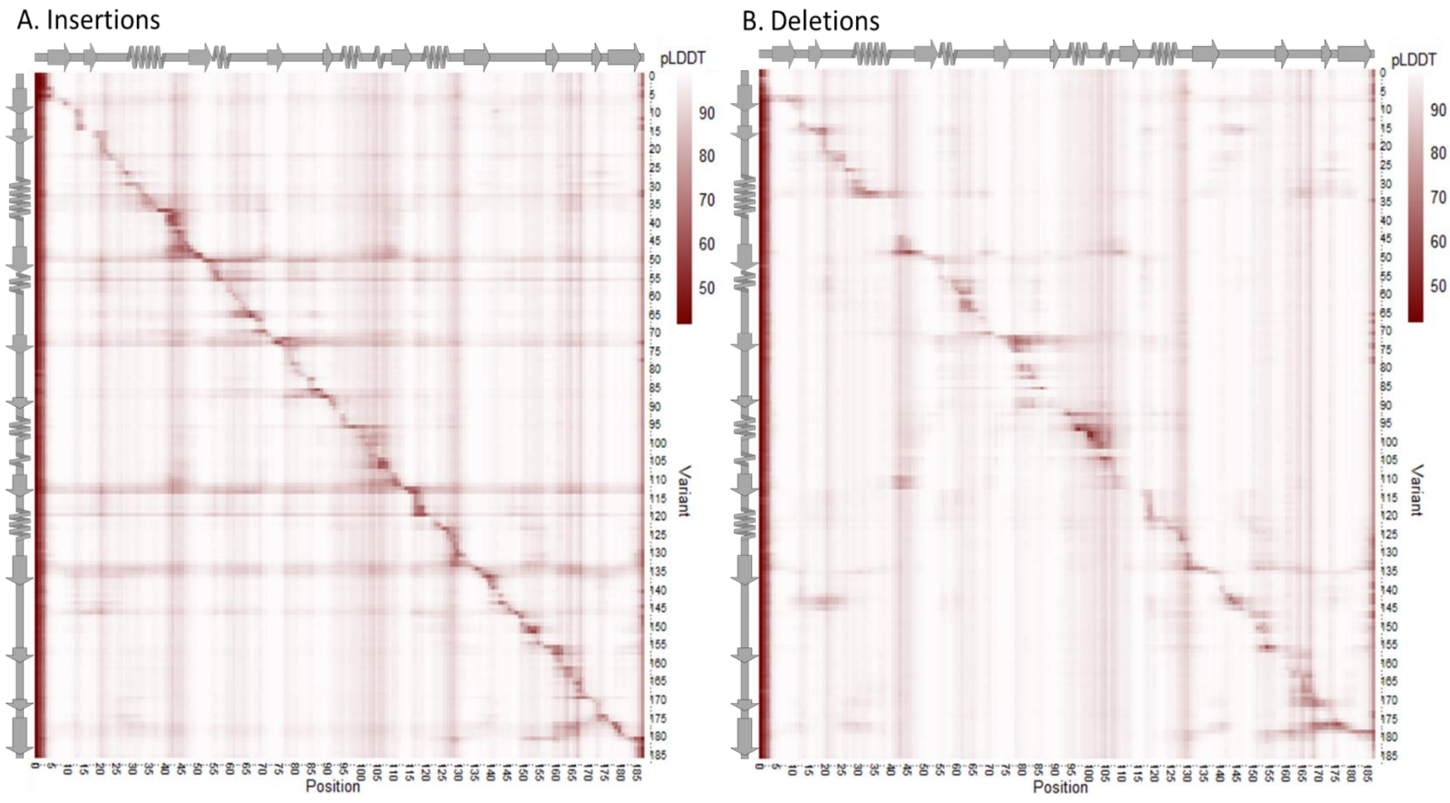

Figure 6: Heatmap of DHFR pLDDT scores per amino acid position for insertion (A) and deletion (B) variants plus DHFR secondary structure. Each row is a variant, starting with 0 which is the wild type DHFR prediction. Every column is a position within DHFR.

To further explore the relationship between the computational predictions and experimental data, we replicated the plots from Woods et al. by plotting the ΔpLDDT on the x-axis and the ΔΔG value on the y-axis, with colours representing the CPOP scores. The resulting four plots, corresponding to the different experimental conditions, reveal a correlation between the computational and experimental scores, similar to the findings of Woods *et al*. (Figure 7). This consistency suggests that the AF2-RR method is capable of producing reliable predictions that align with experimental observations under various conditions.

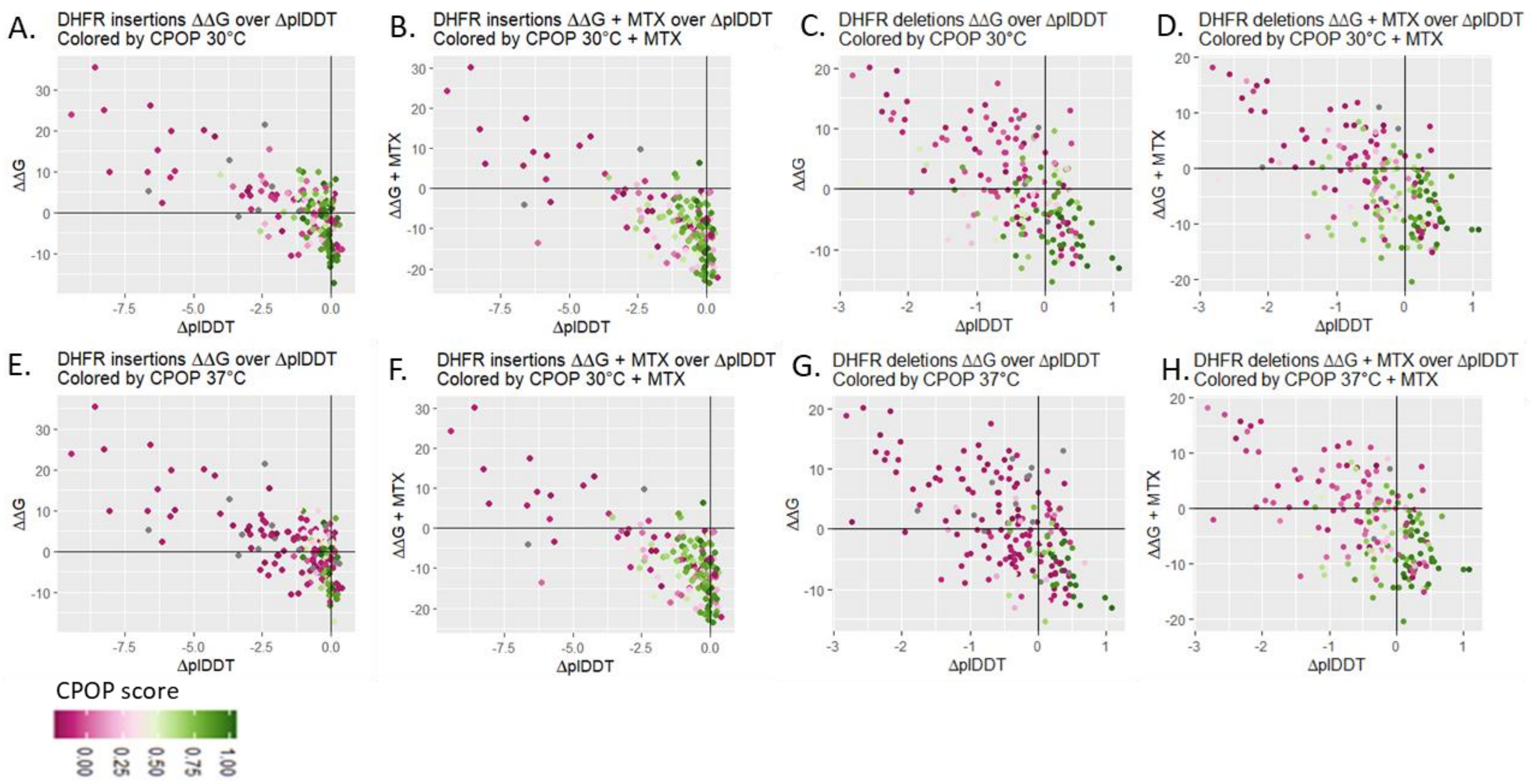

Figure 7: Woods plots of insertion and deletion variants and the four different CPOP conditions. The ΔΔG scores are plotted against the ΔpLDDT scores and coloured by the CPOP scores. Each plot is a different CPOP condition. The ΔΔG score with MTX was used in the plots with MTX in the CPOP data.

To further assess the accuracy of our predictions, we performed ROC curve analysis and plotted the area under the curve (AUC) to compare the computational scores with the CPOP scores. (Figure 8). In addition to the mean pLDDT score, we examined the predictive performance of a small average or window (w) around the indel position. The ROC curves reveal that the AUC increases with the size of the window. However, the AUC never surpasses the mean pLDDT score, except in the case of CPOP 37°C, where it is 0.1 higher. This observation can be rationalised by revisiting Figure 6, where it is evident that all positions around the indel exhibit lower scores, irrespective of the indel's location in the protein. Using a mean pLDDT score takes this into account, thereby enhancing predictive power. Comparing ΔΔG with and without MTX to predict the CPOP scores under corresponding conditions indicates that including MTX does not improve the AUC. Despite MTX's stabilising effect on DHFR, as seen in the CPOP data, the computational model's predictive performance does not significantly benefit from MTX inclusion. This discrepancy suggests that the stabilising influence of MTX, while evident experimentally, may not be fully captured by the computational model, highlighting a potential area for further refinement. In summary, our analysis demonstrates that the AF2-RR method developed by Woods et al. provides a reliable tool for predicting the stability of DHFR indel variants. The correlation between computational and experimental scores, as well as the trends observed in the ROC analysis, underscore the method's utility. However, there remain areas for improvement, particularly in capturing the effects of specific stabilising agents like MTX and in accurately predicting the intolerance to indels in certain protein regions. Future work should focus on integrating additional structural or functional data and exploring machine learning techniques to enhance the predictive accuracy of computational models for protein stability.

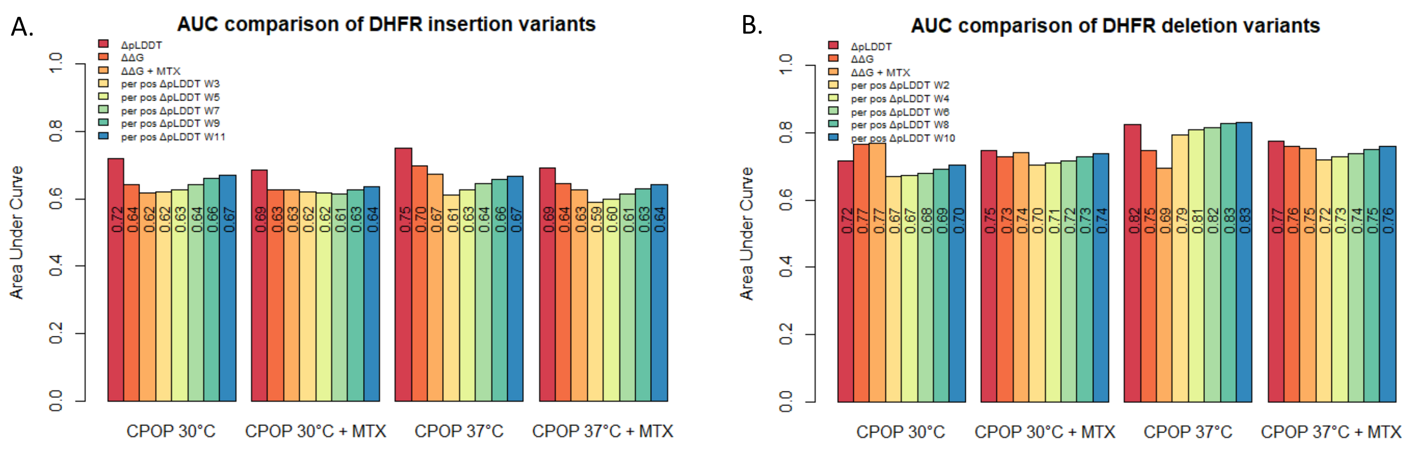

Figure 8: Area Under the Curve (AUC) from ROC Curve Analysis. This figure presents the AUC values derived from ROC curve analysis, comparing the predictive performance of AF2 and RR scores against the experimental CPOP scores under various conditions.

Conclusion

In this study, we applied the AlphaFold2-RosettaRelax (AF2-RR) method to predict the stability of DHFR indel variants. Our findings demonstrate that the AF2-RR method can effectively predict stability scores, with an AUC of 0.62-0.82. We observed a correlation between the computational predictions and the experimental CPOP scores, though the computational models did not capture the intolerance to indels in the middle segment of the DHFR protein. This discrepancy suggests a potential area for improvement in the models, possibly requiring additional structural or functional data. Future research should focus on expanding the dataset, incorporating more diverse protein structures, and refining computational methods to better predict the effects of indels. More data would also enable the possibility to combine the pLDDT and ΔΔG scores with machine learning for greater predictive power. Another route to explore is the method developed by the Topolska *et al*. group mentioned in the introduction [8]. We, like woods *et al*., found that most indels are not accepted within secondary structures and more so in unstructured regions, loops and surface exposed areas. The INDELi-X/D/G/E model of Topolska *et al*. takes the variant position within the protein into account by giving it a score based on whether it is positioned in a secondary structure, how long the structure is and what the neighbouring structures are. This metric is combined with average substitution score, either experimentally determined (INDELi-X) or computationally (DDMut: INDELi-D, GEMME: INDELi-G, ESM-1v: INDELi-E), for a very high predictive power. They tested their INDELi-E method on known indel variants from the ClinVar database, which resulted in an AUC of 0.92 for insertions and 0.86 for deletions. Making it a very promising method to explore with the DHFR data. More so, replacing the computational predicted average substitution score with a pLDDT score predicted by AF2 using a indel variant instead might improve the accuracy even more. Overall, this study highlights the potential and limitations of using AF2-RR for predicting the stability of protein indels, and suggests pathways for further improving these computational approaches.

Word of thanks

I would like to express my gratitude towards Amelie Stein for providing me with an internship position, the support during the project, and creative freedom. Marion, Aleks and Aitana, thank you so much for answering all my questions. More importantly for the moral support and fun and games. I really had a great time with everyone from the office and I will look back on this this abroad experience with a smile.

Bibliography

1. Durbin R. A map of human genome variation from population-scale sequencing. *Nature*. 2010;467(7319).

2. Loewe L, Hill WG. The population genetics of mutations: Good, bad and indifferent. *Phil Trans R Soc B*. 2010;365(1544):1153. doi: 10.1098/rstb.2009.0317.

3. Lin M, Whitmire S, Chen J, Farrel A, Shi X, Guo J. Effects of short indels on protein structure and function in human genomes. *Sci Rep*. 2017;7(1). doi: 10.1038/s41598-017-09287-x.

4. Stenson PD, Mort M, Ball EV, et al. The human gene mutation database: 2008 update. . 2009.

5. Larsen-Ledet S. Mind the gap. . 2023. doi: https://doi.org/10.1016/j.str.2023.05.005.

6. Miton CM, Tokuriki N. Insertions and deletions (indels): A missing piece of the protein engineering jigsaw. *Biochemistry*. 2022;62(2):148. doi: 10.1021/acs.biochem.2c00188.

7. Woods H, Schiano DL, Aguirre JI, et al. Computational modeling and prediction of deletion mutants. *Structure (London)*. 2023;31(6):713-723.e3. https://dx.doi.org/10.1016/j.str.2023.04.005. doi: 10.1016/j.str.2023.04.005.

8. Topolska M, Beltran A, Lehner B. Deep indel mutagenesis reveals the impact of amino acid insertions and deletions on protein stability and function. . 2023. doi: 10.1101/2023.10.06.561180.

9. Kircher M, Witten DM, Jain P, O&#39;roak BJ, Cooper GM, Shendure J. A general framework for estimating the relative pathogenicity of human genetic variants. *Nat Genet*. 2014;46(3):310. doi: 10.1038/ng.2892.

10. Choi Y, Sims GE, Murphy S, Miller JR, Chan AP. Predicting the functional effect of amino acid substitutions and indels. *PLoS ONE*. 2012;7(10). doi: 10.1371/journal.pone.0046688.

11. Brandes N, Goldman G, Wang CH, Ye CJ, Ntranos V. Genome-wide prediction of disease variant effects with a deep protein language model. *Nat Genet*. 2023;55(9):1512. doi: 10.1038/s41588-023-01465-0.

12. Jumper J, Evans R, Pritzel A, et al. Highly accurate protein structure prediction with AlphaFold. *Nature*. 2021;596(7873):583. doi: 10.1038/s41586-021-03819-2.

13. Conway P, Tyka MD, Dimaio F, Konerding DE, Baker D. Relaxation of backbone bond geometry improves protein energy landscape modeling. *Protein Science*. 2013;23(1):47. doi: 10.1002/pro.2389.

14. Alford RF, Leaver-Fay A, Jeliazkov JR, et al. The rosetta all-atom energy function for macromolecular modeling and design. *J. Chem. Theory Comput.* 2017;13(6):3031. doi: 10.1021/acs.jctc.7b00125.

15. Schnell JR, Dyson HJ, Wright PE. Structure, dynamics, and catalytic function of dihydrofolate reductase. *Annu. Rev. Biophys. Biomol. Struct.* 2004;33(1):119. doi: 10.1146/annurev.biophys.33.110502.133613.

16. Bjerre B, Nissen J, Madsen M, et al. Improving folding properties of computationally designed proteins. . 2019;32(3):145. doi: 10.1093/protein/gzz025.

17. Chen EA, Porter LL. SSDraw: Software for generating comparative protein secondary structure diagrams. . 2023. doi: 10.1101/2023.08.25.554905.

Supplements

Here all/most of the code can be found used to run AF2-RR and produce the figures. Most scripts are only shown in part due to the repetitiveness of the code.

Script 1: Python code to produce the insertion variants FASTA files used for AF2 input.
Script 2: Python code used to run AlphaFold2 in Computerome (supercomputer cluster).
Script 3: Python code used to fetch the AF2 output from Computerome (pLDDT scores).
Script 4: Python code to fetch the pLDDT score per amino acid position
Script 5: Python code to run RosettaRelax in DeiC (Danish e-infrastructure Cooperation).
Script 6: R code used to make Figure 4: distributions of the data.
Script 7: R code used to make Figure 5: heatmap, of all the scores.
Script 8: R code used to make Figure 6: heatmap of the per position pLDDT scores.
Script 9: R code used to make Figure 7: Woods plots.
Script 10: R code to calculate the average pLDDT scores with different window sizes.
Script 11: R code to make Figure 8: calculating the area under the curve.

Script 1: Python code to produce the insertion variants FASTA files used for AF2 input.

```
# aim of the script is to produce a .FASTA file of insertion variants
# deletion script written (and used) by Aitana Genzor
# with the wt sequence and all amino acid insertions on every position,
individually

# import
import os

# Global Variables
# P00374-1 human wt DHFR (can be replaced with any desired protein
sequence)
PROTEIN_NAME = "dhfr"
PROTEIN_AA_SEQUENCE =
"MVGSLNCIVAVSQNMGIGKNGDLPWPPLRNEFRYFQRMTTTSSVEGKQNLVIMGKKTWFSIPEKNRPLKGRINL
VLSRELKEPPQGAHFLSRSLDDALKLTEQPELANKVDMVWIVGGSSVYKEAMNHPGHLKLFVTRIMQDFESDTFF
PEIDLEKYKLLPEYPGVLSDVQEEKGIKYKFEVYEKND"
AMINO_ACIDS = "G"  # all amino acids: "ARNDCEQGHILKMFPSTWYV"
FILE_PATH = './dhfr_fasta_insert_files'
FASTA_NAME = ">sp|P00374|DYR_HUMAN|DHFR"
ZEROS = "{:0" + str(len(str(len(PROTEIN_AA_SEQUENCE)))) + "d}" # to add
leading zero's to the names {:03d}

# make folder to hold all other folders with the .FASTA files
os.mkdir(FILE_PATH)
# make protein wt folder
os.mkdir(FILE_PATH + "/" + PROTEIN_NAME)
# make protein .FASTA file and place in said folder
# step 1: make the path to the protein file
fasta_path = FILE_PATH + "/" + PROTEIN_NAME + "/" + PROTEIN_NAME + ".FASTA"
# step 2: write the fasta file, fill it and close it
save_fasta = open(fasta_path,  "w+")
save_fasta.write(FASTA_NAME + "\n" + PROTEIN_AA_SEQUENCE)
save_fasta.close()

# for all the insertion sequences
current_aa_seq = ""
protein_seq_library = []
name = ""
filename = ""
for pos in range(1,len(PROTEIN_AA_SEQUENCE) + 1):
    for aa in AMINO_ACIDS:
        current_aa_seq = str(PROTEIN_AA_SEQUENCE[:pos] + aa +
PROTEIN_AA_SEQUENCE[pos:len(PROTEIN_AA_SEQUENCE)])
        if current_aa_seq in protein_seq_library:
            print("insert aa sequence already present, FASTA file not made,
nr:" + str(pos))
        else:
            # add FASTA to the library
            protein_seq_library.append(current_aa_seq)
            # make the FASTA sequence name:
            name = FASTA_NAME + "_" + aa + "_inserted_at_position_" +
ZEROS.format(pos)
            # filename
            filename = PROTEIN_NAME + "_" + aa.lower() +
"_inserted_at_position_" + ZEROS.format(pos)
            # make protein varient folder
            os.mkdir(FILE_PATH + "/" + filename)
            # make protein .FASTA file and place in said folder
            # step 1: make the path to the protein file
            fasta_path = FILE_PATH + "/" + filename + "/" + filename +
".FASTA"
            # step 2: write the fasta file, fill it and close it
            save_fasta = open(fasta_path, "w+")
            save_fasta.write(FASTA_NAME + "|" + name + "\n" +
current_aa_seq)
            save_fasta.close()
```

Script 2: Python code used to run AlphaFold2 in Computerome (supercomputer cluster).

Only showing insertions due to the repetitiveness of the code.

```
# aim of the script is to run alphafold in computerome on the fasta files
that are each in a likewise named folder
```

```python
# import
import os
import subprocess

# global variables
EMAIL = "h.lanters@students.uu.nl" #to notify when the job is done
NODES = 1
GPUS = 1
PPN = 40
MEMORY = 100
WALLTIME = 4 #hours
MAIN_FOLDER_DIRECTORY = "./dhfr_fasta_insert_files"
#MAIN_FOLDER_DIRECTORY =
"/home/projects/ku_00122/people/hellan/dhfr_fasta_insert_files"
START = 0
STOP = 10 # how many jobs you want to submit, max 188

# List all subdirectories in the main folder
subdirectories = [d for d in os.listdir(MAIN_FOLDER_DIRECTORY) if
os.path.isdir(os.path.join(MAIN_FOLDER_DIRECTORY, d))]

# Iterate over each subdirectory
for folder in subdirectories[START:STOP]:
    # Store the folder name in a variable
    folder
    # make the .sh script with the job details:
    script_content = f"""#!/bin/bash
#PBS -W group_list=ku_00122 -A ku_00122
#PBS -N {folder}
#PBS -e {folder}/slurm.err
#PBS -o {folder}/slurm.out
#PBS -m e
#PBS -M {EMAIL}
#PBS -l nodes={NODES}:gpus={GPUS}:ppn={PPN},mem={MEMORY}gb
##PBS -l nodes=1:thinnode:ppn=40,mem=100gb
#PBS -l walltime={WALLTIME}:00:00

module load cuda/toolkit/11.3.0 cudnn/11.3-8.2.1.32 tensorrt/7.2.1.6
anaconda3/2023.09-0 perl/5.36.1 openmpi hhsuite/3.3.0 hmmer/3.3.2
kalign/3.3.1
module load tools alphafold/2.2.4

dir_AF=/services/tools/alphafold/2.2.4
dir_db=/home/databases/alphafold
dir_F={MAIN_FOLDER_DIRECTORY}

#source $dir_F/env.sh || exit 1

bash $dir_AF/run_alphafold.sh -f $dir_F/{folder}/{folder}.FASTA -t 2021-11-
01 -d $dir_db -o $dir_F/{folder}/af_output -r false
"""
    # save the bash script
    print(folder)
    with open(MAIN_FOLDER_DIRECTORY + "/" + folder + "/" + folder + ".sh",
"w") as file:
        file.write(script_content)

    # how to run the qsub submission script from the python script
    subprocess.run(['qsub', MAIN_FOLDER_DIRECTORY + "/" + folder + "/" +
folder + ".sh"])
```

Script 3: Python code used to fetch the AF2 output from Computerome (pLDDT scores).

Only showing insertions due to repetitiveness of the code.

```
# This will be a script to process the JSON file output from AlphaFold.
# This script will collect all the ranking_debug.json data.
# Each file has 5 AF predicted scores, for the 5 predicted structures per
variant
# The scores are already the average per predicted structure.
# The scores will be saved in a matrix:
# Each row is a variant, each column an AF score for that variant

# To run the script in Computerome, numpy and pandas had to be installed,
which I did with the following commands:
## Create a Virtual Environment
#python3 -m venv myenv
## Activate the virtual environment
#source myenv/bin/activate
## install packages
#pip install numpy
#pip install pandas
## run the script
#python3 process_json.py
## how to exit the virtual environment
#deactivate
## copy the file to the login node so you can email it to yourself
#cp /home/projects/ku_00122/people/hellan/dhfr_af_average_plddt_scores.csv
/home/people/hellan/Documents/dhfr_af_average_plddt_scores.csv

# PACKAGES
import os
import json
import numpy as np
import pandas as pd

# GLOBAL VARIABLES
# This is the amount of variants.
# Find out how many there are by running this command in Computerome in the
directory with all the variant files:
# ls -1 | wc -l
NR_OF_ROWS = 175
NR_OF_COLS = 5  # Because we have 5 AF scores (one per prediction)
MAIN_FOLDER_DIRECTORY = "./dhfr_fasta_insert_files"
#MAIN_FOLDER_DIRECTORY =
"/home/projects/ku_00122/people/hellan/dhfr_fasta_insert_files"
#OUTPUT_CSV_PATH =
"home/people/hellan/Documents/dhfr_af_average_plddt_scores.csv"
#OUTPUT_CSV_PATH =
"/home/projects/ku_00122/people/hellan/dhfr_af_average_plddt_scores.csv"
OUTPUT_CSV_PATH = "./dhfr_af_average_plddt_scores.csv" # The people
directory did not work, I copied the files there
START = 0
STOP = NR_OF_ROWS

# Function to process a JSON file and update the matrix
def process_json(json_path, matrix, row_names):
    with open(json_path, 'r') as json_file:
        data = json.load(json_file)
    # Extract plddt scores and update the matrix
    plddt_scores = [data['plddts'][model] for model in data['order']]
    matrix.append(plddt_scores)

    # Extract row name from the directory name
    row_name = os.path.basename(os.path.dirname(json_path))
    row_names.append(row_name)

# List all subdirectories in the main folder
subdirectories = [d for d in os.listdir(MAIN_FOLDER_DIRECTORY) if
os.path.isdir(os.path.join(MAIN_FOLDER_DIRECTORY, d))]

# Initialize an empty matrix
matrix = []
row_names = []
col_names = [f"plddt_score_{i+1}" for i in range(NR_OF_COLS)]

# Iterate over each subdirectory
for folder in subdirectories[START:STOP]:
    # Create the path to the json file
    json_file_path = os.path.join(MAIN_FOLDER_DIRECTORY, folder,
"af_output", folder, "ranking_debug.json")
    # Extract the plddt data
    process_json(json_file_path, matrix, row_names)

matrix = np.array(matrix)

# Create a DataFrame with row and column names
df = pd.DataFrame(matrix, index=row_names, columns=col_names)

# Print the final DataFrame
#print(df)

# Save the DataFrame as a CSV file
df.to_csv(OUTPUT_CSV_PATH)

print(f"CSV file saved to: {OUTPUT_CSV_PATH}")
```

Script 4: Python code to fetch the pLDDT score per amino acid position.

Only showing deletions due to repetitiveness of the code

```
# script to extract the per position pLDDT score (currently set for
deletions)

import os
import pandas as pd
import re

# Directory containing PDB files
directory =
"C:/Users/hllee/Documents/UU/Copenhagen/Data/DHFR/dhfr_pdbs_del"

# Initialize a list to store plddt scores for each residue
plddt_series_list = []

# Iterate over each PDB file in the directory
for filename in os.listdir(directory):
    if filename.endswith(".pdb"):
        filepath = os.path.join(directory, filename)
        plddt_scores = {}
        with open(filepath, "r") as file:
            # Iterate over each line in the PDB file
            for line in file:
                if line.startswith("ATOM"):
                    # Extract residue number (column 6) and plddt score
(column 11)
                    residue_number = int(line[22:26].strip())
                    plddt_score = float(line[61:68])
                    # Store plddt score for each residue only once
                    plddt_scores[residue_number] = plddt_score
        # Extract part of the filename to use as row name
        rn = filename.split("_")[-2]  # Extracts the xxx_x part from the
filename
        row_name = re.sub('\D', '', rn)
        # Append plddt scores for the current file to the list
        plddt_series_list.append(pd.Series(plddt_scores, name=row_name))

# Concatenate all Series into a DataFrame
plddt_df = pd.concat(plddt_series_list, axis=1)

# Transpose the DataFrame to have files as rows and residues as columns
plddt_df = plddt_df.transpose()

# Reset index to make filenames a column
plddt_df.reset_index(inplace=True)
plddt_df.rename(columns={"index": "position"}, inplace=True)

# sort the position row
plddt_df['position'] = pd.to_numeric(plddt_df['position'])
plddt_df_sort = plddt_df.sort_values(by=['position'], na_position='first')

# Print the DataFrame
print(plddt_df_sort)

plddt_df_sort.to_csv(r'C:\Users\hllee\Documents\UU\Copenhagen\Data\DHFR\dhf
r_plddt_perpos_dup_all_del.csv', index = False)
```

Script 5: Python code to run RosettaRelax in DeiC (Danish e-infrastructure Cooperation).

```
# Script to Run Rosetta Relax (adapted for deletions)

import os
import subprocess

run_dir = '/groups/sbinlab/heleen/rundir'
input_dir = '/groups/sbinlab/heleen/dhfr_pdbs'
#pdb_list = os.listdir(input_dir)
pdb_list = ["dhfr_0", "dhfr_1"]
output_dir = '/groups/sbinlab/rosetta_relax_dhfr_insertions_ddg'
path_to_ROSETTA =
'/sbinlab/software/Rosetta_2021_Aug_c7009b3/source/bin/relax.default.linuxg
ccrelease'
partition = 'sbinlab'

for pdb in pdb_list:
  pdb_path = os.path.join(input_dir, pdb)
path_to_bash = os.path.join(run_dir, (pdb + '.sbatch'))

with open(path_to_bash, 'w') as fp:
  fp.write(f'''#!/bin/sh
#SBATCH --job-name=Woods_relax
#SBATCH --time=60:00:00
#SBATCH --mem 5000
#SBATCH --partition={partition}

''')

fp.write((f'/sbinlab/software/Rosetta_2021_Aug_c7009b3/source/bin/relax.default.linuxgccrelease '
            f'-s {pdb_path} '
            f'-relax:constrain_relax_to_start_coords 1 '
            f'-nstruct 10 '
            f'-out:file:silent {output_dir}_{pdb.split(".")[0]}.out '
            f'-out:file:scorefile {output_dir}_{pdb.split(".")[0]}.sc >
{output_dir}_{pdb.split(".")[0]}.log'))
subprocess.call(f'sbatch {path_to_bash}', shell=True, cwd=run_dir)
```

Script 6: R code used to make Figure 4: distributions of the data.

Only one condition shown (CPOP 30°C) due to the repetitiveness of the code

```
##############################################################
####################### CPOP 30°C #######################
wt <- 1

# Create the combined plot
combined_plot <- ggplot() +
  geom_density(data = del, aes(x = cpop_30, fill = "Deletions"), alpha =
0.5) +
  geom_density(data = ins, aes(x = cpop_30, fill = "Insertions"), alpha =
0.5) +
  scale_fill_manual(values = c("Deletions" = "magenta", "Insertions" =
"yellow")) +
  labs(title = "A. Plot of CPOP 30°C indel scores",
       x = "CPOP 30°C",
       y = "Density") +
  theme_minimal() +
  theme(legend.title = element_blank()) +
  xlim(-1, 1.7) +
  geom_vline(xintercept = wt, linetype = "solid", color = "black", size =
.8) +
  annotate("label", x = 1.2, y = 1.28, label = "WT" , color = "black",
alpha = 0.3)


# Print the combined plot
print(combined_plot)
```

Script 7: R code used to make Figure 5: heatmap, of all the scores.

Secondary DHFR structure was obtained using SSDraw and put together in PowerPoint.

```
# libraries

library(pheatmap)
library(RColorBrewer)
library(viridis)
library(reshape2)
library(grid)
library(gridExtra)
# load the data
```

```r
ins <-
read.csv("~/UU/Copenhagen/Data/DHFR/dhfr_cpop_plddt_dplddt_ddg_ddgmtx_windo
ws_dup_ins.csv")
del <-
read.csv("~/UU/Copenhagen/Data/DHFR/dhfr_cpop_plddt_dplddt_ddg_ddgmtx_windo
ws_dup_del.csv")

# "settings" for the heatmaps
# nr of colors (same as number of breaks)
nrcolors <- 100

# add column with new numbers, starting at 1 skipping 5
# Create a vector with the values for the skip_pos column
skip_values <- rep(".", nrow(ins))
skip_values[seq(1, nrow(ins), by = 5)] <- seq(1, nrow(ins), by = 5)

# Assign the skip_pos column to the ins dataframe
ins$skip_pos <- skip_values


# CPOP insertions heatmap
colrwb <- colorRampPalette(c("#720000", "white", "#001540"))
cmin <- -0.3
cmax <- 2.3
pheatmap(t(ins[,2:5]), cluster_rows = F, cluster_cols = F,
         color = colrwb(nrcolors),
         breaks = seq(from = cmin, to = cmax, length.out = nrcolors),
         labels_row = c("", "", "", ""), # removing the rownames makes it
easier to align
         labels_col = ins$skip_pos)

# CPOP deletions heatmap
pheatmap(t(del[,2:5]), cluster_rows = F, cluster_cols = F,
         color = colrwb(nrcolors),
         breaks = seq(from = cmin, to = cmax, length.out = nrcolors),
         labels_row = c("", "", "", ""), # removing the rownames makes it
easier to align
         labels_col = ins$skip_pos)

# CPOP insertions heatmap only CPOP 30
colrwb <- colorRampPalette(c("#720000", "white", "#001540"))
cmin <- -0.234
cmax <- 2.234
pheatmap(t(ins$score_30), cluster_rows = F, cluster_cols = F,
         color = colrwb(nrcolors),
         breaks = seq(from = cmin, to = cmax, length.out = nrcolors),
         labels_col = ins$skip_pos)

# CPOP deletions heatmap
pheatmap(t(del$score_30), cluster_rows = F, cluster_cols = F,
         color = colrwb(nrcolors),
         breaks = seq(from = cmin, to = cmax, length.out = nrcolors),
         labels_col = ins$skip_pos)


# heatmap pLDDT insertions
colrwb <- colorRampPalette(brewer.pal(10, "Spectral"))
cmin <- 86 # OWN SCALE
cmax <- 96
pheatmap(t(ins$plddt), cluster_rows = F, cluster_cols = F,
         color = colrwb(nrcolors),
         breaks = seq(from = cmin, to = cmax, length.out = nrcolors))

# and deletions OWN SCALE
cmin <- 92
cmax <- 97
pheatmap(t(del$plddt), cluster_rows = F, cluster_cols = F,
         color = colrwb(nrcolors),
         breaks = seq(from = cmin, to = cmax, length.out = nrcolors))


# heatmap ddG insertions
colrwb <- colorRampPalette(c("darkgreen", "white", "#7c1881"))
# truncated because the extremes are not important
cmin <- -16.5
cmax <- 16.5

pheatmap(t(ins$ddg), cluster_rows = F, cluster_cols = F,
         color = colrwb(nrcolors),
         breaks = seq(from = cmin, to = cmax, length.out = nrcolors))
# and deletions
pheatmap(t(del$ddg), cluster_rows = F, cluster_cols = F,
         color = colrwb(nrcolors),
         breaks = seq(from = cmin, to = cmax, length.out = nrcolors))


# heatmap ddG + MTX insertions
colrwb <- colorRampPalette(c("#026e97", "white", "#cc9b00"))
cmin <- -35.5
cmax <- 35.5
pheatmap(t(ins$ddg_mtx), cluster_rows = F, cluster_cols = F,
         color = colrwb(nrcolors),
         breaks = seq(from = cmin, to = cmax, length.out = nrcolors))
# and deletions
pheatmap(t(del$ddg_mtx), cluster_rows = F, cluster_cols = F,
         color = colrwb(nrcolors),
         breaks = seq(from = cmin, to = cmax, length.out = nrcolors))
```

Script 8: R code used to make Figure 6: heatmap of the per position pLDDT scores.

Only insertions shown due to receptiveness of the code.

```
### heatmap of per position pLDDT score INSERTIONS
ppins <-
read.csv("~/UU/Copenhagen/Data/DHFR/dhfr_perpos_plddt_dup_all_av_ins.csv")
colnames(ppins) <- sub("^X", "", colnames(ppins))
ppins <- ppins[-1,] # exclude wt

# Define color range and breaks
colrwb <- colorRampPalette(c("#720000", "white"))
ccol <- c(colrwb(100))
cmin <- min(apply(ppins[,-1], 1, min, na.rm = T))
cmax <- max(apply(ppins[,-1], 1, max, na.rm = T))

# add column with new numbers, starting at 1 skipping 5
# Create a vector with the values for the skip_pos column
skip_values <- rep(".", nrow(ppins))
indices <- seq(0, nrow(ppins) - 1, by = 5)
skip_values[indices + 1] <- indices
```

```
# Assign the skip_pos column to the df
ppins$skip_pos_row <- skip_values

# make the heatmap
hm_pp_ins <- pheatmap(ppins[,-c(1, 190)], cluster_rows = F, cluster_cols =
F,
                      color = ccol,
                      breaks = seq(from = cmin, to = cmax, length.out =
100) ,
                      main = "Heatmap of DHFR per postion pLDDT score of
the insertion variants",
                      labels_row = ppins$skip_pos_row, fontsize_row = 7,
                      labels_col = ppins$skip_pos, fontsize_col = 7)
```

Script 9: R code used to make Figure 7: Woods plots.

Only insertions shown due to repetitiveness of the code.

```
# scripts to make all the plots with the complete CPOP data

library(ggplot2)
library(gridExtra)
library(pROC)

ins <-
read.csv("~/UU/Copenhagen/Data/DHFR/dhfr_cpop_dplddt_ddg_ddgmtx_dup_ins.csv
")
del <-
read.csv("~/UU/Copenhagen/Data/DHFR/dhfr_cpop_dplddt_ddg_ddgmtx_dup_del.csv
")

###########################################################################
########## INSERTIONS
plot_ins_30 <- ggplot(ins, aes(x = dplddt, y = ddg, color = cpop_30)) +
  geom_point() +
  scale_color_gradientn(colours = c("#8E1652", "#C5247D", "#DE77AE",
"#F1B6DA", "#FDE0EF",
                                    "#E6F5D0", "#B8E186", "#7FBC41",
"#4D9220", "#286419"),
                        name = "CPOP \n30°C") +
  geom_vline(xintercept = 0) + geom_hline(yintercept = 0) +
  ggtitle("DHFR insertions ΔΔG over ΔplDDT \nColored by CPOP 30°C") +
  labs(x= "ΔplDDT", y="ΔΔG")  +
  theme(plot.title = element_text(size = 12))

plot_ins_30_mtx <- ggplot(ins, aes(x = dplddt, y = ddg_mtx, color =
cpop_30_mtx)) +
  geom_point() +
  scale_color_gradientn(colours = c("#8E1652", "#C5247D", "#DE77AE",
"#F1B6DA", "#FDE0EF",
                                    "#E6F5D0", "#B8E186", "#7FBC41",
"#4D9220", "#286419"),
                        name = "CPOP \n30°C\n+ MTX") +
  geom_vline(xintercept = 0) + geom_hline(yintercept = 0) +
  ggtitle("DHFR insertions ΔΔG + MTX over ΔplDDT \nColored by CPOP 30°C +
MTX") +
  labs(x= "ΔplDDT", y="ΔΔG + MTX")  +
  theme(plot.title = element_text(size = 12))

plot_ins_37 <- ggplot(ins, aes(x = dplddt, y = ddg, color = cpop_37)) +
  geom_point() +
  scale_color_gradientn(colours = c("#8E1652", "#C5247D", "#DE77AE",
"#F1B6DA", "#FDE0EF",
                                    "#E6F5D0", "#B8E186", "#7FBC41",
"#4D9220", "#286419"),
                        name = "CPOP \n37°C") +
  geom_vline(xintercept = 0) + geom_hline(yintercept = 0) +
  ggtitle("DHFR insertions ΔΔG over ΔplDDT \nColored by CPOP 37°C") +
  labs(x= "ΔplDDT", y="ΔΔG")  +
  theme(plot.title = element_text(size = 12))

plot_ins_37_mtx <- ggplot(ins, aes(x = dplddt, y = ddg_mtx, color =
cpop_37_mtx)) +
  geom_point() +
  scale_color_gradientn(colours = c("#8E1652", "#C5247D", "#DE77AE",
"#F1B6DA", "#FDE0EF",
                                    "#E6F5D0", "#B8E186", "#7FBC41",
"#4D9220", "#286419"),
                        name = "CPOP \n37°C\n+ MTX") +
  geom_vline(xintercept = 0) + geom_hline(yintercept = 0) +
  ggtitle("DHFR insertions ΔΔG + MTX over ΔplDDT \nColored by CPOP 37°C +
MTX") +
  labs(x= "ΔplDDT", y="ΔΔG + MTX")  +
  theme(plot.title = element_text(size = 12))

grid.arrange(plot_ins_30, plot_ins_30_mtx, plot_ins_37, plot_ins_37_mtx,
ncol = 2
```

Script 10: R code to calculate the average pLDDT scores with different window sizes.

Only insertions shown due to repetitiveness of the code, data used for the ROC curves.

```
# script to calculate the per position plddt scores with different window
sizes
# insertions

# load the data
ppins <-
read.csv("~/UU/Copenhagen/Data/DHFR/dhfr_plddt_perpos_dup_ins.csv",
check.names = F)
ppins_wt <- as.matrix(ppins[1,])
ppins <- as.matrix(ppins[-1,])

ppins_av <- matrix(nrow = nrow(ppins), ncol = 6)

for (i in 1:nrow(ppins)) {
  colnr <- which(colnames(ppins) == ppins[i+1, 1])
  ppins_av[i, 1] <- ppins[i, 1]
  ppins_av[i, 2] <- mean(c(ppins[i, colnr-1], ppins[i, colnr  ], ppins[i,
colnr+1]))
  ppins_av[i, 3] <- mean(c(ppins[i, colnr-2], ppins[i, colnr-1], ppins[i,
colnr  ], ppins[i, colnr+1], ppins[i, colnr+2]))
  ppins_av[i, 4] <- mean(c(ppins[i, colnr-3], ppins[i, colnr-2], ppins[i,
colnr-1], ppins[i, colnr  ], ppins[i, colnr+1], ppins[i, colnr+2], ppins[i,
colnr+3]))
  ppins_av[i, 5] <- mean(c(ppins[i, colnr-4], ppins[i, colnr-3], ppins[i,
colnr-2], ppins[i, colnr-1], ppins[i, colnr  ], ppins[i, colnr+1], ppins[i,
colnr+2], ppins[i, colnr+3], ppins[i, colnr+4]))
  ppins_av[i, 6] <- mean(c(ppins[i, colnr-5], ppins[i, colnr-4], ppins[i,
colnr-3], ppins[i, colnr-2], ppins[i, colnr-1], ppins[i, colnr  ], ppins[i,
```

```r
colnr+1], ppins[i, colnr+2], ppins[i, colnr+3], ppins[i, colnr+4], ppins[i,
colnr+5]))
}
colnames(ppins_av) <- c("position", "w_3", "w_5", "w_7", "w_9", "w_11")

# fix the averages at the beginning and end
# if the window is bigger then there are numbers I will take less numbers
############# pos 1
# w5
ppins_av[1,3] <- mean(ppins[1, 2:5])
# w7
ppins_av[1,4] <- mean(ppins[1, 2:6])
# w9
ppins_av[1,5] <- mean(ppins[1, 2:7])
# w11
ppins_av[1,6] <- mean(ppins[1, 2:8])

############# pos 2
# w7
ppins_av[2,4] <- mean(ppins[2, 2:7])
# w9
ppins_av[2,5] <- mean(ppins[2, 2:8])
# w11
ppins_av[2,6] <- mean(ppins[2, 2:9])

############# pos 3
# w9
ppins_av[3,5] <- mean(ppins[3, 2:9])
# w11
ppins_av[3,6] <- mean(ppins[3, 2:10])

############# pos 4
# w11
ppins_av[4,6] <- mean(ppins[4, 2:11])


# now the same for the end values

#################### pos 183
# w 11
ppins_av[183, 6] <- mean(ppins[183, 180:189])

#################### pos 184
# w 11
ppins_av[184, 6] <- mean(ppins[184, 181:189])
# w 9
ppins_av[184, 5] <- mean(ppins[184, 182:189])
# w 7
ppins_av[184, 4] <- mean(ppins[184, 183:189])
# w 5
ppins_av[184, 3] <- mean(ppins[184, 184:188])
# w 3
ppins_av[184, 2] <- mean(ppins[184, 185:187])
# pos
ppins_av[184, 1] <- 184

#################### pos 185
# w 11
ppins_av[185, 6] <- mean(ppins[185, 182:189])
# w 9
ppins_av[185, 5] <- mean(ppins[185, 183:189])
```

```r
# w 7
ppins_av[185, 4] <- mean(ppins[185, 184:189])
# w 5
ppins_av[185, 3] <- mean(ppins[185, 185:189])
# w 3
ppins_av[185, 2] <- mean(ppins[185, 186:188])
# pos
ppins_av[185, 1] <- 185

#################### pos 186
# w 11
ppins_av[186, 6] <- mean(ppins[186, 183:189])
# w 9
ppins_av[186, 5] <- mean(ppins[186, 184:189])
# w 7
ppins_av[186, 4] <- mean(ppins[186, 185:189])
# w 5
ppins_av[186, 3] <- mean(ppins[186, 186:189])
# w 3
ppins_av[186, 2] <- mean(ppins[186, 187:189])
# pos
ppins_av[186, 1] <- 186

#################### pos 187
# w 11
ppins_av[187, 6] <- mean(ppins[187, 184:189])
# w 9
ppins_av[187, 5] <- mean(ppins[187, 185:189])
# w 7
ppins_av[187, 4] <- mean(ppins[187, 186:189])
# w 5
ppins_av[187, 3] <- mean(ppins[187, 187:189])
# w 3
ppins_av[187, 2] <- mean(ppins[187, 188:189])
# pos
ppins_av[187, 1] <- 187

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

ppins_d <- ppins_av - ppins_wt_av
ppins_d[,1] <- ppins_av[,1]

write.csv(ppins_av,
"~/UU/Copenhagen/Data/DHFR/dhfr_perpos_plddt_dup_windows_ins.csv",
row.names = F,
          quote = F)
write.csv(ppins_wt_av,
"~/UU/Copenhagen/Data/DHFR/dhfr_perpos_plddt_dup_windows_wt_ins.csv",
row.names = F,
          quote = F)
write.csv(ppins_d,
"~/UU/Copenhagen/Data/DHFR/dhfr_perpos_dplddt_dup_windows_ins.csv",
row.names = F,
          quote = F)
```

Script 11: R code to make Figure 8: calculating the area under the curve.

For the ROC plots only pLDDT insertions are shown due to the repetitiveness of the code. Note how the deduplicated data was used.

```r
# script to make ROC plots for the CPOP ins and dels to get the AUC

library(ggplot2)
library(pROC)
library(gridExtra)
library(RColorBrewer)
library(tidyverse)

ins <-
read.csv("~/UU/Copenhagen/Data/DHFR/dhfr_cpop_plddt_dplddt_ddg_ddgmtx_windo
ws_dedup_ins.csv")
del <-
read.csv("~/UU/Copenhagen/Data/DHFR/dhfr_cpop_plddt_dplddt_ddg_ddgmtx_windo
ws_dedup_del.csv")

# cut off for cpop data 0.5
# above 0.5 soluble, below and equal to 0.5 deleterious

calculate_cutoff <- function(df, cutoff_nr, colnr, new_colname){
  for (i in 1:nrow(df)) {
    if (is.na(df[i, colnr])) {
      df[i, new_colname] <- NA
    } else if (df[i, colnr] > cutoff_nr) {
      df[i, new_colname] <- 1
    } else {
      df[i, new_colname] <- 0
    }
  }
  return(df)
}

# add the cut-off score as a new column to the dataset
ins <- calculate_cutoff(df = ins, cutoff_nr =  0.5, colnr =  2,
"cutoff_30")
ins <- calculate_cutoff(df = ins, cutoff_nr =  0.5, colnr =  3,
"cutoff_30_mtx")
ins <- calculate_cutoff(df = ins, cutoff_nr =  0.5, colnr =  4,
"cutoff_37")
ins <- calculate_cutoff(df = ins, cutoff_nr =  0.5, colnr =  5,
"cutoff_37_mtx")

del <- calculate_cutoff(df = del, cutoff_nr =  0.5, colnr =  2,
"cutoff_30")
del <- calculate_cutoff(df = del, cutoff_nr =  0.5, colnr =  3,
"cutoff_30_mtx")
del <- calculate_cutoff(df = del, cutoff_nr =  0.5, colnr =  4,
"cutoff_37")
del <- calculate_cutoff(df = del, cutoff_nr =  0.5, colnr =  5,
"cutoff_37_mtx")


# roc plots for plddt and ins
rocobj1 <- roc(ins$cutoff_30, ins$dplddt)
auc1 <- round(auc(ins$cutoff_30, ins$dplddt), 4)
roc_ins_dplddt_30 <- ggroc(rocobj1, colour = 'steelblue', size = 2) +
  ggtitle(paste0('ROC Curve, DHFR insertions, CPOP 30°C, ΔpLDDT ', '(AUC =
', auc1, ')'))

rocobj2 <- roc(ins$cutoff_30_mtx, ins$dplddt)
auc2 <- round(auc(ins$cutoff_30_mtx, ins$dplddt), 4)
roc_ins_dplddt_30_mtx <- ggroc(rocobj2, colour = 'steelblue', size = 2) +
  ggtitle(paste0('ROC Curve, DHFR insertions, CPOP 30°C MTX, ΔpLDDT ',
'(AUC = ', auc2, ')'))

rocobj3 <- roc(ins$cutoff_37, ins$dplddt)
auc3 <- round(auc(ins$cutoff_37, ins$dplddt), 4)
roc_ins_dplddt_37 <- ggroc(rocobj3, colour = 'steelblue', size = 2) +
  ggtitle(paste0('ROC Curve, DHFR insertions, CPOP 37°C, ΔpLDDT ', '(AUC =
', auc3, ')'))

rocobj4 <- roc(ins$cutoff_37_mtx, ins$dplddt)
auc4 <- round(auc(ins$cutoff_37_mtx, ins$dplddt), 4)
roc_ins_dplddt_37_mtx <- ggroc(rocobj4, colour = 'steelblue', size = 2) +
  ggtitle(paste0('ROC Curve, DHFR insertions, CPOP 37°C  MTX, ΔpLDDT ',
'(AUC = ', auc4, ')'))

grid.arrange(roc_ins_dplddt_30, roc_ins_dplddt_30_mtx, roc_ins_dplddt_37,
roc_ins_dplddt_37_mtx, ncol = 2)

############################################################################
################################### AUC ###################################
############################################################################
# Collect the AUC scores and put them into a df
# Insertions
auc_ins_dplddt <- c(auc1, auc2, auc3, auc4) # pLDDT
auc_ins_ddg <- c(auc5, auc6, auc7, auc8) # ddG
auc_ins_ddg_mtx <- c(auc5m, auc6m, auc7m, auc8m) # ddG + MTX
auc_ins_ppdplddt_w3 <- c(auc31, auc32, auc33, auc34) # window 3
auc_ins_ppdplddt_w5 <- c(auc51, auc52, auc53, auc54) # window 5
auc_ins_ppdplddt_w7 <- c(auc71, auc72, auc73, auc74) # window 7
auc_ins_ppdplddt_w9 <- c(auc91, auc92, auc93, auc94) # window 9
auc_ins_ppdplddt_w11 <- c(auc111, auc112, auc113, auc114) # window 11
# combine
auc_insertions <- rbind(auc_ins_dplddt, auc_ins_ddg, auc_ins_ddg_mtx,
auc_ins_ppdplddt_w3, auc_ins_ppdplddt_w5,
                        auc_ins_ppdplddt_w7, auc_ins_ppdplddt_w9,
auc_ins_ppdplddt_w11)
colnames(auc_insertions) <- c("CPOP 30°C", "CPOP 30°C + MTX", "CPOP 37°C",
"CPOP 37°C + MTX")

# deletions
auc_del_dplddt <- c(auc9, auc10, auc11, auc12) # pLDDT
auc_del_ddg <- c(auc13, auc14, auc15, auc16) # ddG
auc_del_ddg_mtx <- c(auc13m, auc14m, auc15m, auc16m) # ddG + MTX
auc_del_ppdplddt_w2 <- c(auc21, auc22, auc23, auc24) # window 2
auc_del_ppdplddt_w4 <- c(auc41, auc42, auc43, auc44) # window 4
auc_del_ppdplddt_w6 <- c(auc61, auc62, auc63, auc64) # window 6
auc_del_ppdplddt_w8 <- c(auc81, auc82, auc83, auc84) # window 8
auc_del_ppdplddt_w10 <- c(auc101, auc102, auc103, auc104) # window 10
# combine
auc_deletions <- rbind(auc_del_dplddt, auc_del_ddg, auc_del_ddg_mtx,
auc_del_ppdplddt_w2, auc_del_ppdplddt_w4,
                        auc_del_ppdplddt_w6, auc_del_ppdplddt_w8,
auc_del_ppdplddt_w10)
colnames(auc_deletions) <- c("CPOP 30°C", "CPOP 30°C + MTX", "CPOP 37°C",
"CPOP 37°C + MTX")


# barplot ins
bp_ins <- barplot(auc_insertions, beside = T,
        col = brewer.pal(nrow(auc_insertions), "Spectral"),
        ylim = c(0, 1), ylab = "Area Under Curve",
        main = "AUC comparison of DHFR insertion variants")
text(x = bp_ins, y = 0.54, labels = format(round(auc_insertions, digits =
2), nsmall = 1), srt = 90, cex = 0.8)
legend('topleft', fill=brewer.pal(nrow(auc_insertions), "Spectral"),
       legend = c("ΔpLDDT", "ΔΔG", "ΔΔG + MTX", "per pos ΔpLDDT W3", "per
pos ΔpLDDT W5", "per pos ΔpLDDT W7",
                  "per pos ΔpLDDT W9", "per pos ΔpLDDT W11"), horiz = F,
inset = c(0, -0.1), xpd = T, cex = 0.625,
       bty = "n")

# barplot del
bp_del <- barplot(auc_deletions, beside = T,
            col = brewer.pal(nrow(auc_deletions), "Spectral"),
            ylim = c(0, 1), ylab = "Area Under Curve",
            main = "AUC comparison of DHFR deletion variants")
text(x = bp_del, y = 0.54, labels = format(round(auc_deletions, digits =
2), nsmall = 1), srt = 90, cex = 0.8)
legend('topleft', fill=brewer.pal(nrow(auc_deletions), "Spectral"),
       legend = c("ΔpLDDT", "ΔΔG", "ΔΔG + MTX", "per pos ΔpLDDT W2", "per
pos ΔpLDDT W4", "per pos ΔpLDDT W6",
                  "per pos ΔpLDDT W8", "per pos ΔpLDDT W10"), horiz = F,
inset = c(0, -0.1), xpd = T, cex = 0.625,
       bty = "n")
```