# Variant effect prediction: Three models for classifying pathogenic mutations in human genes

Kelly Brock

Submitted as a final project in CSCI-S89, Introduction to Deep Learning, Summer 2022

Youtube Presentation Link: < https://www.youtube.com/watch?v=qcD5TAge05U>

GitHub Repo Link: <https://github.com/kpgbrock/CS89_VariantEffectPrediction>

## Abstract

Genetic disorders can involve changes to the coding region of our DNA - or in other words, the part of our genome that can be transcribed and translated to form proteins, which are the building blocks of cells. In particular, missense mutations (where a change in the DNA nucleotide(s) leads to a change in amino acids in the corresponding protein sequence) are of special interest, because they can have a more ambiguous effect on the encoded protein than other types of mutations like premature truncations. Although the scientific community has identified thousands of these single-amino acid changes that are thought to be pathogenic (contributing to disease), genetics studies can be time-consuming and statistically underpowered, particularly in the case of rare variants. Most genetic variants found in humans remain of unknown significance. To fill this gap, a growing wealth of computational methods aim to predict whether certain missense variants are either pathogenic or benign in the context of human disease. For this case study in biology, I built 3 separate neural net architectures that take as input increasingly more information about protein sequence and the missense mutation of interest, and output a classification for that missense mutant as either pathogenic or benign. A particular challenge was how to split our available data for training and testing, so I also experimented with different split methods as well. Across the three methods, I obtained a validation accuracy of up to 80%. I then used these architectures to generate predictions for all possible missense mutations for the KCNQ1 gene, which encodes a potassium channel implicated in congenital heart arrhythmia disorder called long QT syndrome.

# Table of Contents

## Introduction and Problem Statement

Goal: build a neural net classifier that can use minimal information to predict whether a coding genetic variant is pathogenic or not.

Roughly 1 in 10 people in the US are living with a rare disease (statistic from https://rarediseases.info.nih.gov/about), many with a strong genetic component that can be linked to nucleotide substitutions in single genes. Identifying these genetic variants, and correctly classifying them as pathogenic (disease-causing) or benign, can help inform both diagnosis and treatment. Next generation sequencing is enabling broader access to whole-exome and whole-genome sequences for patients and their families. However, many genetic variants identified in families are classified as being of uncertain significance, leading to diagnostic uncertainty and reduced potential for treatment. Computational predictors, including the recently published unsupervised method EVE [1] from my labmates, aim to fill this gap.

The current state of the art in the field of computational *variant effect prediction* - in other words, predicting whether a genetic variant is possibly contributory to a disease or not - requires information like protein structure models or evolutionary sequence alignments to make their calls. For this biology case study, I built three supervised neural net classifiers:

1) Model 1: The ‘aa_subs’ model
   Only input the original, *‘wildtype’* amino acid and what that amino acid mutates to, and feed into a fully connected neural net.

2) Model 2: The ‘window_seq_vep’ model
   Take a symmetric window of the protein sequence centered around our mutant position, encode it using biophysical characteristics of amino acids, and input this sequence along with our wildtype and mutant amino acid to a bi-directional GRU (RNN) model.

3) Model 3: The ‘full_length_alley2019’ model
   Encode the entire protein sequence using a published autoencoder method by Alley et al 2019[2], and feed these encodings along with our mutation into a fully connected neural net.

These models take as input a missense mutation (where one amino acid in the protein the gene encodes is changed), and two of the models additionally take in information about the surrounding protein sequence of the missense mutation. The output for all models is the predicted probability of the missense variant being pathogenic or not. My goal was to see how well a neural net can learn information about mutations from minimal information, and compare the performances for my models between each other and to more state of the art methods that incorporate more input information like sequence homology and structure.

## Methods and User Guide

### Details on data set

#### ClinVar database: labeled missense mutations

I used the ClinVar database[3] , found at <https://www.ncbi.nlm.nih.gov/clinvar/>, that lists genetic variants along with how clinicians at genetics centers have classified them on a scale from pathogenic to benign. In particular, I used a preprocessed version of this dataset from 2020 that was used for the EVE paper, found at:

Data/precursor_files/EVE_ClinVar_cleaned_annotated_GRCh38.csv

I am limiting my dataset to only variants that are labeled as either ‘benign’ or ‘pathogenic’ (leaving out those of ‘uncertain significance’) in my train, validate, and test sets. I then cross-referenced these genes with the Uniprot sequence database[4] to get the protein sequences. The full list of 20,388 canonical human protein sequences was downloaded from Uniprot (<uniprot.org>) to the following file:

Data/precursor_files/uniprot_canonical_human_proteins.tsv

I then matched the protein identifiers from the Uniprot data download to those in the ClinVar download. The list of protein sequences represented by my ClinVar dataset can be found in:

Data/precursor_files/ClinVar_Seqs.fa

All of the above data preprocessing was performed in the the following notebook:

**Code/clinvar_seq_preprocessing.ipynb**

This notebook was run on Google Colab using the same directory structure as uploaded on my GitHub repository. This notebook should be runnable on a standard Colab instance without GPU enabled. I used the processed data directly in downstream notebooks and all files are available in the top level of the ‘Data’ directory. However, if you wish to rerun this notebook, please change the ‘data_dir’ variable in the first cell to point to your installation.

In this same notebook, I also performed my train-validate-test split using multiple strategies. I would like this model to be generalizable to predict novel mutations on previously encountered sequences, and to be able to handle new sequences not encountered in training as well. In general, reducing leakage between train and test sets can be challenging for protein sequence modeling and is an ongoing area of research, because sequences that appear heavily diverged from each other can still share structure and function. However, it can be very difficult for models to learn representations for protein sequences that they have not encountered before.

In this case, because I’m not directly trying to predict structure or function and due to time constraints, I am being lenient with how I split compared to state of the art methods used in protein structure prediction, which has the potential to limit generalizability. Future work in this area could use Hidden Markov Model-based sequence alignment methods to enforce stricter cutoffs for these splits if generalizability is an issue. I used the CD-HIT webserver [5,6] to cluster sequences within a 15% sequence identity threshold. The webserver's primary output can be found in <Data/precursor_files/CD-HIT/1658330861.result/1658330861.fas.1.clstr.sorted>. The list of ClinVar variants, combined with their sequence information and which cluster they are in, is stored at:

Data/processed_clinvar_seqs_with_clusters.csv

I then randomly shuffled these clusters, and performed my splits for variants across sequences at approximately the 80-10-10% (train-validate-test) markers while making sure that clusters did not overlap the categories. The train, validate, and test sets are stored as 'csv' files in the top level of the data directory:

Cluster shuffled set:
Data/train_processed_clinvar_seqs_with_clusters_clustershuffle.csv
Data/validate_processed_clinvar_seqs_with_clusters_clustershuffle.csv
Data/test_processed_clinvar_seqs_with_clusters_clustershuffle.csv

The statistics for this split of the data set are as follows:

| | **# Benign** | **# Pathogenic** | **Fraction Pathogenic** | **# unique protein sequences** |
|---|---|---|---|---|
| **train** | 1930 | 3076 | 0.61 | 852 |
| **validate** | 128 | 268 | 0.68 | 64 |
| **test** | 255 | 391 | 0.61 | 128 |

*Table 1. Statistics for my first train/validate/test split, where I first cluster sequences based on 15% sequence identity threshold, and then shuffle the clusters.*

Note that in my entire ClinVar set, I only have 1,044 protein sequences (out of ~20K total for the human proteome) that have a variant labeled as either pathogenic or benign. For comparison, I also generated train-test splits where I did not index on similarities between input sequences, and just randomly shuffled the rows (each row corresponds to a particular missense mutation). In this case, we will see examples of the same sequence shared between train and test sets, but we won't be seeing the same mutation within that sequence. These files are found at:

Randomly shuffled set:
Data/train_processed_clinvar_seqs_with_clusters_random.csv
Data/validate_processed_clinvar_seqs_with_clusters_random.csv
Data/test_processed_clinvar_seqs_with_clusters_random.csv

| | **# Benign** | **# Pathogenic** | **Fraction Pathogenic** | **# unique protein sequences** |
|---|---|---|---|---|
| **train** | 1833 | 3007 | 0.62 | 960 |
| **validate** | 240 | 379 | 0.61 | 339 |
| **test** | 240 | 349 | 0.59 | 343 |

*Table 2. Statistics for train/validate/test split that randomly shuffled missense mutations without first clustering based on sequence identity.*

I will train using both of these splits, and compare between them in each section.

### Encoding weights from Alley et al 2019

For Model 3, I am also using the UniRep protein sequence encoding, published by Alley et al 2019[2]. This encoding was designed to provide a generalizable representation of all possible protein sequences, specifically for deep learning applications. I downloaded their model weights using the instructions on their GitHub repository, found at <https://github.com/churchlab/UniRep>. These downloaded weights for their 64-encoding set are available at:

Data/UniRep_Weights/64_weights

This model uses an older version of TensorFlow, so I made a specific notebook to run this code for all 20,388 canonical human protein sequences:

**Code/generate_full_length_encodings.ipynb**

This notebook can take on the order of 5 hours to run, so I stored the resulting dictionary mapping protein identifiers to their 64-entry encodings in the following file:

Data/protein_full_length_encodings.pickle

These encodings are referenced by their Uniprot identifier, so any canonical human protein should be runnable in the final demo notebook.

### Secondary data sources

#### *Amino acid properties*

For Model 2, I wanted to use a biophysical ‘encoding’ to input information about the protein sequence surrounding the missense variant. I am using four properties - Kyte-Doolittle hydrophobicity measure[7], molecular mass, relative abundance in protein sequences, and isoelectric point - to characterize different amino acids. These tables were obtained from <https://en.wikipedia.org/wiki/Amino_acid> and <https://www.chem.ucalgary.ca/courses/351/Carey5th/Ch27/ch27-1-4-2.html>.

### *BLOSUM62*

For visualizing my model predictions, I also brought in several outside sources of information to compare against. For Model 1, the goal is to predict the probability of a missense mutation being pathogenic, given just a starting amino acid and what that amino acid is mutating into. This problem definition is similar to the question of how much of a penalty should be taken for substituting one amino acid for another when aligning two sequences together. The BLOSUM62 matrix[8] is a widely used example for this use case. I downloaded BLOSUM62 from <https://www.ncbi.nlm.nih.gov/Class/FieldGuide/BLOSUM62.txt>, and saved it in:

Data/precursor_files/blosum62.csv

In this matrix, both row and column entries correspond to amino acids listed in the same order. Negative values indicate that the amino acid swap was seen less often than would occur by chance in natural sequence alignments, which intuitively corresponds to those mutations being disfavored in protein sequences. This file is processed in the notebook where I train the amino acid substitution model, Model 1.

### *gnomAD Allele Frequencies*

For my technology demonstration, I chose to showcase the KCNQ1 gene. For genetic variants, how frequently they are found in the population (the *allele frequency*) can give information about how likely they are to be pathogenic. In other words, more pathogenic variants tend to be much rarer in the population. I used the gnomAD database[9] to download KCNQ1 variant allele frequencies at the following link:

<https://gnomad.broadinstitute.org/gene/ENSG00000053918?dataset=gnomad_r2_1>

I restricted this set to just missense variants in the canonical transcript - in other weirds, in the accepted default sequence - and downloaded it as a csv file, which can be found at:

Data/precursor_data/gnomAD_KCNQ1_allele_frequencies_7-29-22.csv

## Installation and Configuration

### *Quick Start: Running Demo Notebook*

The code and all supporting data for this project were uploaded to a GitHub repository, found at <https://github.com/kpgbrock/CS89_VariantEffectPrediction>. I ran everything on Google Colab using standard libraries. The user can clone this repository to their own system, and run all notebooks there. All necessary datasets to reproduce the findings are included in this repository in the 'Data' directory.

**Important note: To run any of these model training or demo notebooks, the first cell in each notebook will have variables that need to be set to the user's particular configurations.**

For example, in all of the train and demo notebooks, there is a variable called ‘main_path’ set in the first cell. This should be set to the filepath of the top level directory for this project for the user. For example, I ran these notebooks on Google Colab from my Google Drive folder, so that path for my setup points to '/content/drive/MyDrive/DeepLearning_Summer2022/Final_Project/', which should contain both a ‘Code’ and a ‘Data’ directory. Please set this variable to point to your own cloned version of the repository.

I have created a demo Jupyter notebook to walk through how to load the three models I trained, and use them to predict the effects of missense mutations for a given protein:

**Code/demo_all_three_models.ipynb**

To run this notebook, load it either locally or in Google Colab. Set the ‘main_path’ variable in the first cell to point to your installation of the cloned GitHub repository. If running locally, also set the ‘using_google_colab’ variable to False.

This notebook can be used to run any valid human canonical Uniprot identifier - candidates are found in the ‘Entry’ column in the following file:

Data/precursor_files/uniprot_canonical_human_proteins.tsv

However, if you choose to run a different protein than the demonstration KCNQ1 protein product (P51787), please also set the ‘run_gnomad’ variable in the top cell to False. I only downloaded gnomAD data for KCNQ1, so the cells in the ‘Compare to gnomAD allele frequencies’ will not produce accurate results if attempted on a different protein.

After changing variables in the top cell, the notebook can be run normally to produce example heatmaps showcasing predictions across the three models for all possible missense mutations in the protein of interest.

One final note is that I defined classes, variables, functions, and other commonly used functions in the following file:

**Code/vep.py**

Please refer to this file if you want to see the ‘under the hood’ implementations.

## Implementation and Architecture

### Model 1: Only amino acid substitutions

The code for generating Model 1 can be found in:

**Code/model1_train_aa_subs_model.ipynb**

To rerun, change the ‘main_path’ variable in the top cell to point to your local installation. This notebook was run in Google Colab, so comment out the lines:

```
from google.colab import drive
drive.mount('/content/drive')
```

in the second cell if running locally. My goal for this model was to see how well I could predict pathogenicity, using just what the original (‘wildtype’) amino acid was, and the amino acid that it mutates to. There are 20 naturally occurring amino acids in human proteins, so I used a one-hot encoding for both the wildtype and mutant amino acid. Therefore, for each missense mutation, my input was a 40x1 vector containing both of these one hot encodings. For my train/test split, I only tried the clustered set, since I was not using any information about the surrounding protein sequence.

For the architecture, I used a fully connected neural network with 20 neurons in the first layer, 10 in the second layer, and 2 for the output layer. This is a binary classification problem so I only really needed one output neuron; however, I wanted my code to be flexible to accommodate an ‘uncertain’ designation as well, so I kept it as 2 outputs. On that note, I also used categorical_crossentropy as my loss metric as well.

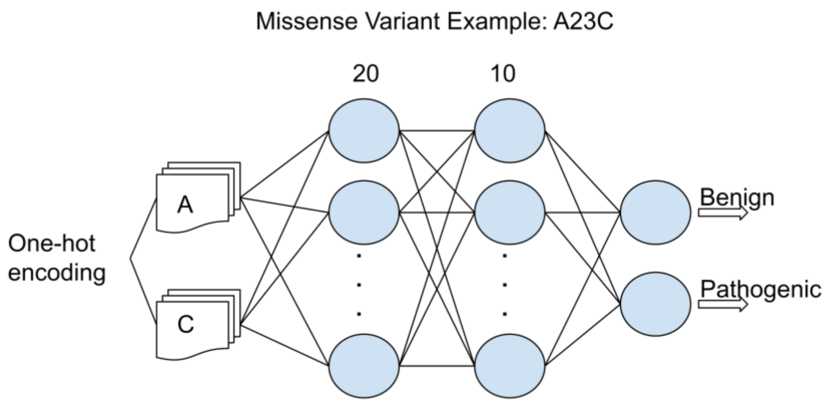


*Figure 1. Neural net architecture for Model 1.*

For my other hyperparameters, I used ‘rmsprop’ as my optimizer, a batch size of 512, and training time of 30 epochs.

I used a Python class called Model1, defined in Code/vep.py, to store and process results. This is a child class of the VEP_Model class, defined in the same file, and inherits some of its plotting functionalities from there.

### Model 2: RNN with missense mutant + sequence window

How pathogenic a particular missense variant is can depend on the protein sequence surrounding it. Therefore, I wanted to build a model that could look at a window of the protein sequence centered around the mutant position. The window length is a hyperparameter that can be changed. In the training notebook, looking at 6 values to the left and right of the amino acid of interest (resulting in a total window size of 13 counting the mutant position) produced stable results, although further hypertuning could result in an improved model. In terms of the RNN architecture, this is equivalent to looking at 13 timesteps. If my window went beyond the boundaries of the protein sequence, I padded with the gap character '-' and gave it zero values for all amino acid properties.

I chose to use a bi-directional RNN to tackle this problem, to make sure that I was fully accounting for amino acids both upstream and downstream of the mutant position. I also chose to use GRU as my RNN of choice. The code used to train this model is found in

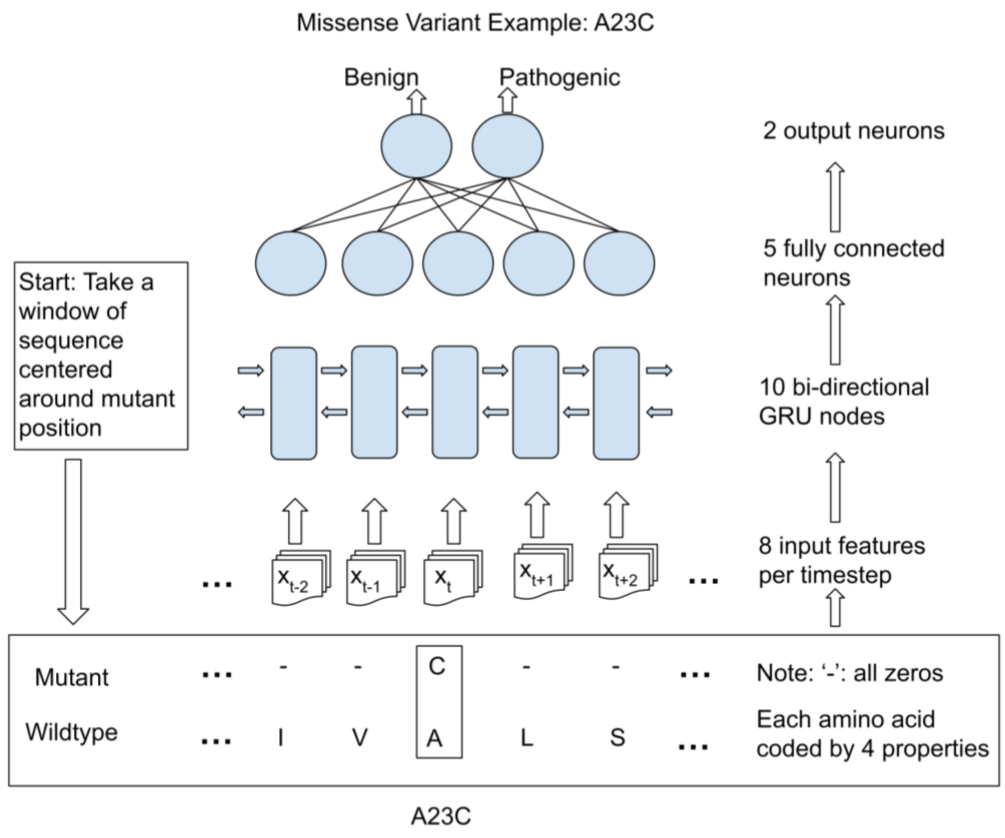
**Code/model2_train_window_seq_vep_model.ipynb**

For my input, I wanted to encode important values about each amino acid, so I used 8 features per position (timestep). The first 4 were the amino acid characteristics for the wildtype amino acid, as described in this writeup at

Details on data set -> Secondary data sources -> Amino acid properties

The last four features corresponded to the mutant amino acid. I either put in all zeros for the flanking amino acids (timesteps 1:(n-1) and (n+1):N where n is the position of the missense mutation and N is the length of the total window), or the 4 amino acid characteristic values for the mutant amino acid at position n. I used a MinMaxScaler from sklearn to scale the data.

For this model, I had a layer of 10 GRU nodes with the 'relu' activation function, followed by a fully connected layer of 5 neurons, and finished with two nodes for output. This model was particularly difficult to train, and I had ongoing problems with overfitting. I tried Dropout but ultimately did not include that layer; however, I did add an L2 regularizer with lambda of 0.00005 that helped. I trained for 200 epochs on both my clustered train/test split and my non-clustered ('random') split.



Missense Variant Example: A23C
Benign
Pathogenic
2 output neurons
5 fully connected neurons
Start: Take a window of sequence centered around mutant position
10 bi-directional GRU nodes
8 input features per timestep
$x_{t-2}$
$x_{t-1}$
$x_t$
$x_{t+1}$
$x_{t+2}$
Mutant
Wildtype
C
I
V
A
L
S
Note: '-': all zeros
Each amino acid coded by 4 properties
A23C

*Figure 2. Bi-directional GRU neural net architecture for Model 2.*

To handle data processing for this model, I also made a VEP_Model child class called Model2. The code is again in Code/vep.py - note that instantiating this class requires one input argument, the file path pointing to the csv file with amino acid properties,

Data/precursor_files/amino_acid_properties.csv

## Model 3: Missense mutant + full-length sequence encoding

Finally, I wanted to pass in information about the full protein sequence, in addition to the missense mutant of interest. Luckily, a 2019 paper by Alley et al[2] provides a method of encoding variable sequence lengths into a set of 64 encodings. I generated these encodings for all ~20K canonical human proteins using the:

Code/generate_full_length_encodings.ipynb

as described above and stored the results in:

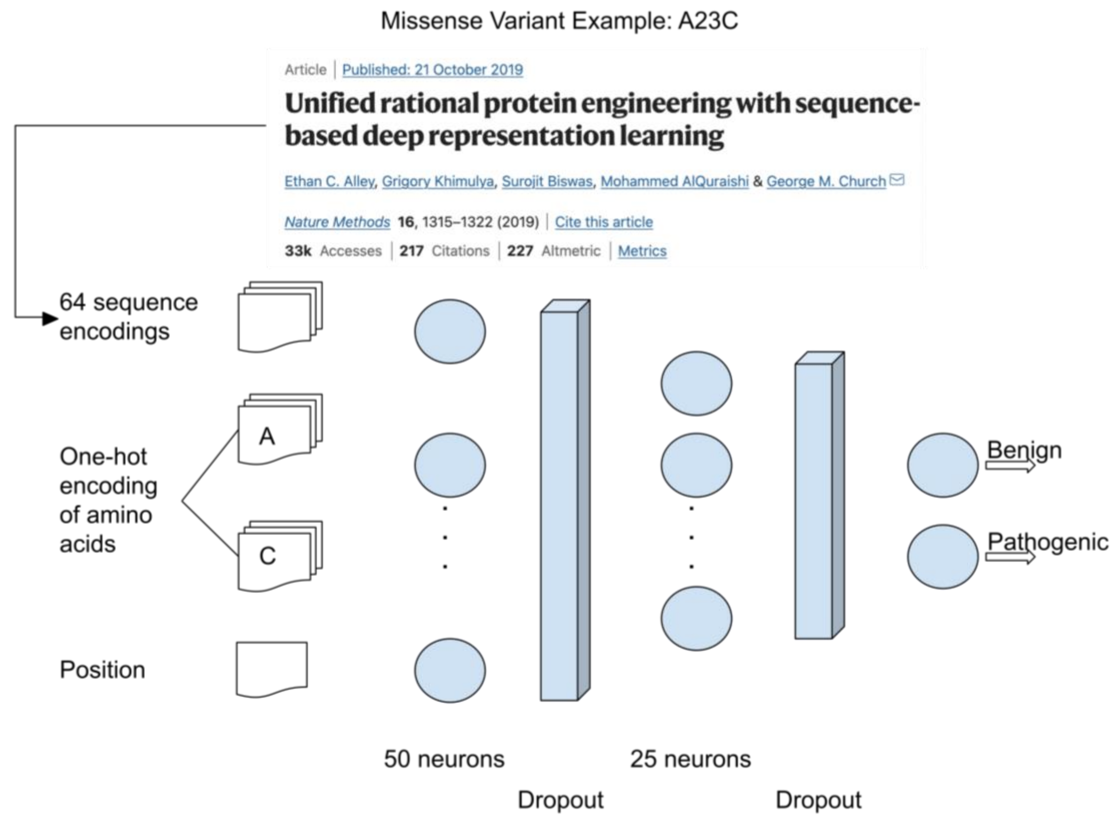
Data/Data/protein_full_length_encodings.pickle

Like Model 1 and Model 2, I also created a child class called Model3 to handle data processing and plotting. Each instance of this class must be instantiated with the file path pointing to the pickle file containing the dictionary of 64-node encodings.

The top-level code used to train this model is found in:

**Code/model3_train_full_length_alley2019_model.ipynb**

For my model architecture, I again used a fully connected neural network. The input was the 64 encodings for the specified protein, along with one-hot encodings for the wildtype and mutant amino acids as specified in Model 1. I also included 1 additional input, the position of the missense mutation normalized by the sequence length, to pass information about where the missense mutation was located in the protein. This resulted in 105 total features.



Missense Variant Example: A23C
Article | Published: 21 October 2019
Unified rational protein engineering with sequence-based deep representation learning
Ethan C. Alley, Grigory Khimulya, Surojit Biswas, Mohammed AlQuraishi & George M. Church
Nature Methods 16, 1315–1322 (2019) | Cite this article
33k Accesses | 217 Citations | 227 Altmetric | Metrics
64 sequence encodings
One-hot encoding of amino acids
A
C
Position
50 neurons
Dropout
25 neurons
Dropout
Benign
Pathogenic

*Figure 3. Neural network architecture for Model 3.*

The network architecture consisted of a layer of 50 neurons and L2 regularizers, a Dropout layer, a layer of 25 neurons with L2 regularizers, another Dropout layer, and a final output layer of 2 neurons. For all relevant layers, I used an L2 regularizer with lambda  0.025 and a dropout fraction of 0.4. For the clustered train/test split, I trained for 200 epochs, and for the randomized, non-clustered train/test split I trained for 800 epochs with a batch size of 512 for both.

## Results

### Model 1: Only amino acid substitutions

This model took in no sequence information, unlike the other two models. The training and validation loss and accuracy are plotted below as a function of epoch:

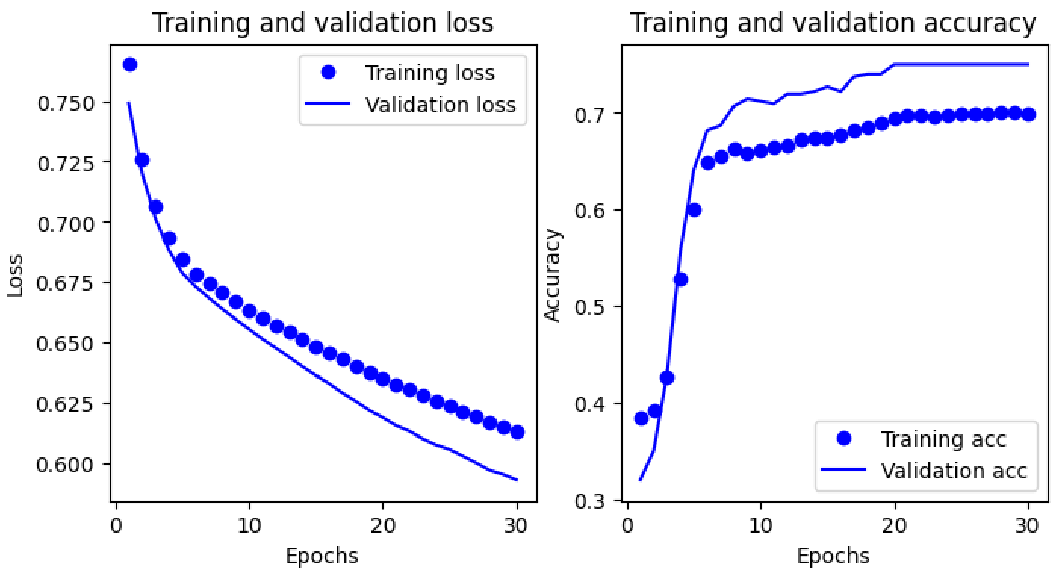


*Figure 4. Training and validation loss (left) and accuracy (right) for Model 1 as a function of epoch.*

The curves are relatively smooth, and the accuracy reaches a distinct plateau. The train and validate accuracies were 0.699 and 0.750 respectively, while the final test accuracy was 0.709. For this model, I also wanted to see how well it could do by chance - I shuffled the labels of the test set and calculated the resulting ‘accuracy’. The distribution of these randomized scores are shown in Figure 5, with the blue line indicating the actual test accuracy. From these results, our model is definitely learning something, judging by the increase in accuracy compared to what would be expected by chance.

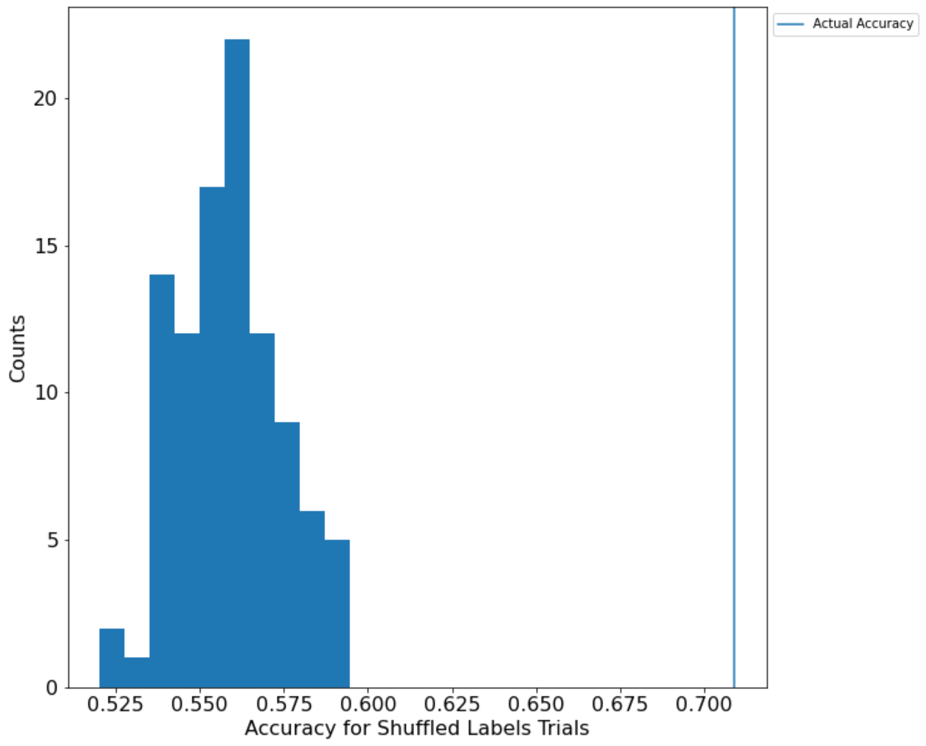


*Figure 5. Calculated accuracy for random shufflings of labels (blue bars), compared to the accuracy of the test set (blue line).*

For this model in particular, we also have another way to judge its performance. The BLOSUM62[8] matrix is a standard in the field of aligning different protein sequences together, and provides cost values for substituting one amino acid for another based on looking at observed alignments. In other words, the BLOSUM62 values can be used as a proxy for how much of an effect a particular amino acid substitution has, averaged across many different sequences. The more negative the value, the more of a penalty that particular substitution incurs. Intuitively, I might expect pathogenic amino acid substitutions to be more likely to have more negative BLOSUM62 scores. The comparison is shown in Figure 6.

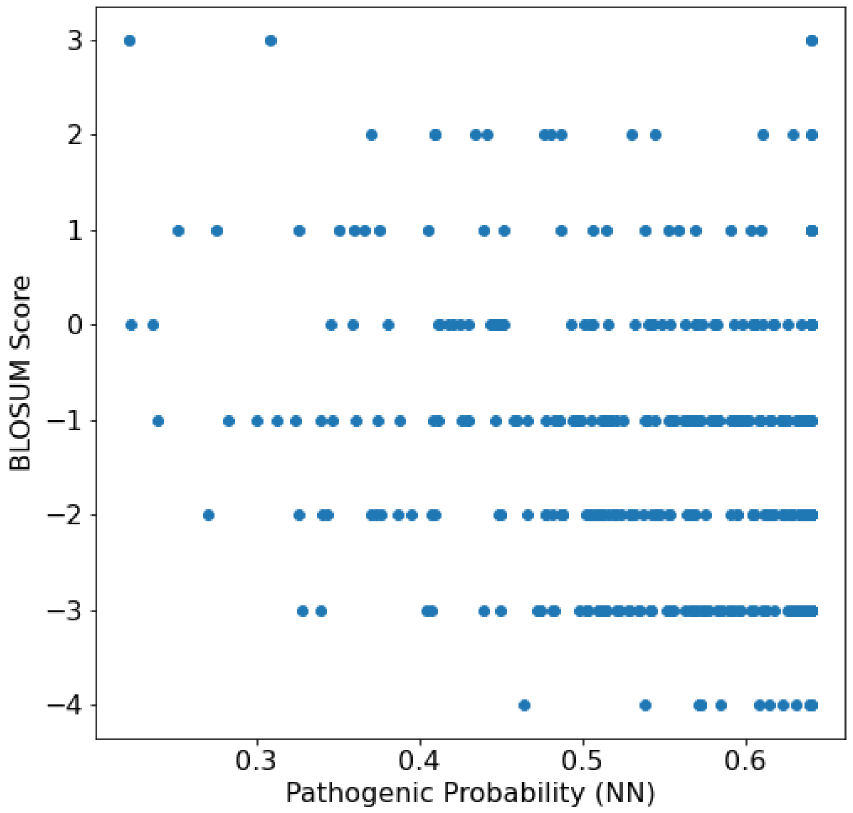


*Figure 6. Comparison of predicted pathogenicity probability (x-axis) to the transition score from the BLOSUM62 matrix (y-axis).*

The BLOSUM62 values are all integers, which is why we're seeing straight lines. Interestingly, the most penalized substitutions in BLOSUM62 (amino acid substitutions with value = -4) are almost all labeled as pathogenic in Model 1, whereas ⅔ of the least penalized substitutions are predicted as benign in our model. The overall Pearson correlation coefficient R is -0.28 (p-value of 2.2e-8). The fact that we do see a mild negative correlation, which matches the hypothesis that more negative BLOSUM62 scores correspond to a higher pathogenic probability, is reassuring.

### Model 2: RNN with missense mutant and sequence window

I now want to start incorporating information about the protein sequence in the area of the missense mutation, using a bi-directional GRU layer. For this section, I trained using two different methods of train/test split: the 'clustered' method guarantees that no sequences (or their corresponding annotated mutations) will be shared between the train, validate, and test sets, while the 'random' method only guarantees that we will not repeat mutations across our train, validate, and test sets. The first method could provide a better sense of how the model will perform on sequences that it has never seen before, while the second split method will provide a better measure of how the model will perform on novel mutations in a sequence that it has been exposed to previously.

The loss and accuracy curves are shown below for the clustered train/test split:

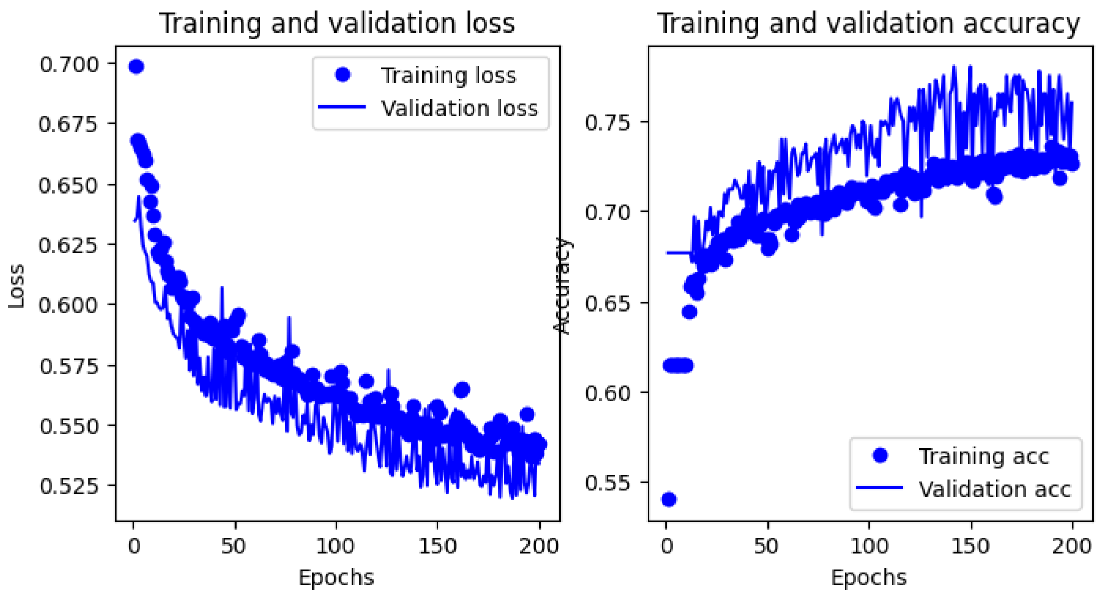


*Figure 7. Train and validate loss (left) and accuracy (right) for the 'clustered' test split strategy.*

This model did not train as cleanly as Model 1, and the loss and accuracy curves are a lot noisier as well. However, we did see train and validate accuracies improve from Model 1, with values of 0.726 and 0.760 respectively. The test accuracy was 0.672; this was a decrease compared to Model 1. Therefore, I want to also try training on the train/test split where we allow the model to see potential sequences in the train set (found in our 'random' split).

For this random train set, where we expect less generalization to unseen sequences but possibly better performance on sequences that have been trained on, we see final accuracies of 0.745 and 0.717:

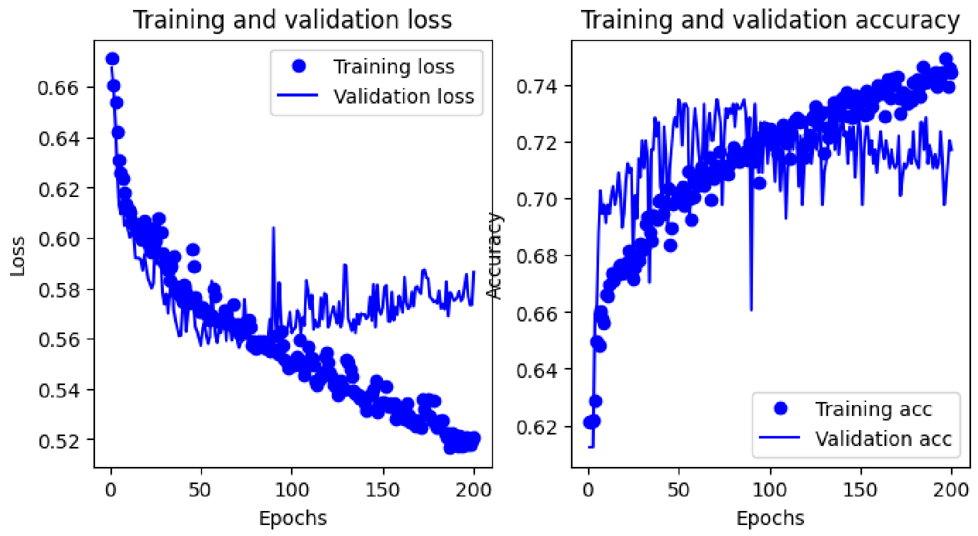


*Figure 8. Train and validate loss (left) and accuracy (right) for the 'random' test split strategy.*

Like with the other split strategy, our training is very noisy. The final test accuracy was 0.677, again representing a slight loss compared to the amino acid only model. I discuss my impressions about why it slightly underperformed what I expected in the Discussion section.

## Model 3: Missense mutant + full-length sequence encoding

This model was the one that used the full-length protein sequence by encoding it using a published method, along with information about the position and amino acids involved in the missense mutation. The training and validation loss and accuracies are shown in Figure 9 for my clustered test/train split:

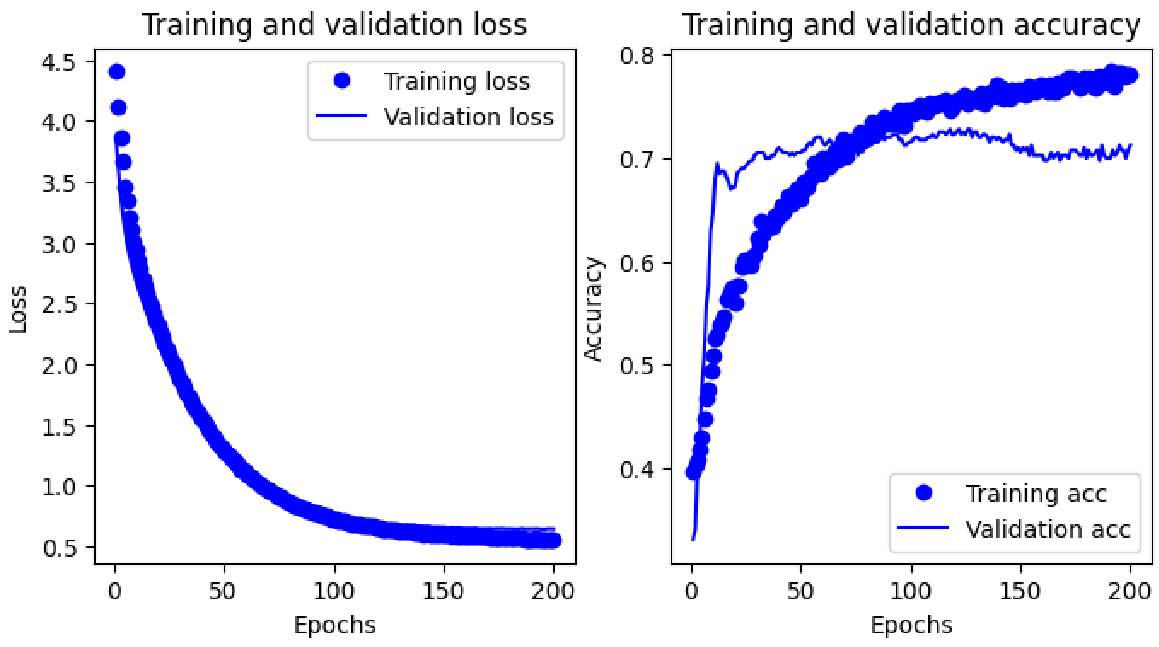


*Figure 9. Loss (left) and accuracy (right) for training and validation set for our clustering split, where we do not repeat sequences between train and test, for Model 3.*

These training curves were much less noisy than Model 2. However, they were also a bit subject to overfitting - if I kept training for more epochs, the train accuracy could get up to 90%, while the validate accuracy did not come close to that height. The train and validate accuracies were 0.781 and 0.712, while the test accuracy was 0.704. Although the train accuracy was higher than my other two models, these findings did not extend to the test set - likely because this model is particularly reliant on having seen the sequence before, probably even more so than Model 2 which uses small sequence windows as opposed to the full length.

My more lenient train/test split showcases this finding particularly well, and shows that for sequences the model has potentially been exposed to, it can predict unseen mutations in those sequences better than my versions of Models 1 and 2. The train and validation loss and accuracy curves are shown in Figure 10:

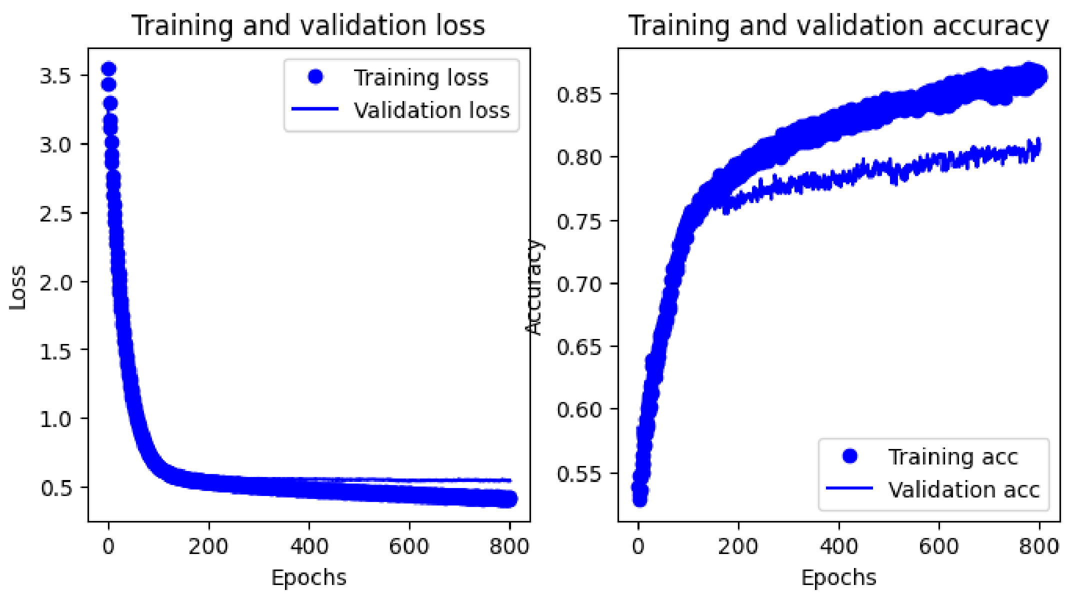


Figure 10. *Loss (left) and accuracy (right) for training and validation set for our random split, where we potentially repeat sequences between train and test, for Model 3.*

Final train accuracy was 0.863, final validation accuracy was 0.809, and final test accuracy was 0.805. These were by far our highest values across all models and train splits tested, and also represented our closest match between validation and test accuracies as well. The key point is that because we're using an encoding of the full length sequence, Model 3 is very sensitive to different sequence encodings and needs to have been exposed to the sequence in training in order to achieve these higher accuracy results. However, this means that it will likely not generalize well to sequences it has not been trained on.

## Case Study: Predicting KCNQ1 variants

To showcase how these models can be applied in a real-world setting, I did a case study by using these models to predict pathogenicity of all possible missense mutations on the KCNQ1 gene. Dozens of variants in this gene have been directly linked to long QT syndrome[10], a heart arrhythmia disorder where the heart takes longer than normal to recharge between beats. Untreated long QT syndrome can be asymptomatic; however, it can cause fainting and torsades de pointes, a potentially fatal heart rhythm. Long QT syndrome is most commonly inherited as an autosomal dominant trait, meaning that an affected parent has a 50% chance of passing it on to each child. If detected, long QT syndrome can be effectively treated with a class of drugs known as beta blockers; because of this, the American College of Medical Genetics includes KCNQ1 on their list of actionable genes that should be screened when sequencing is performed.

Although mutations in KCNQ1 are the most common cause of long QT syndrome, most of the variants in this gene reported to ClinVar are of uncertain significance (457 out of 617 total missense variants). This finding is not uncommon among genes of interest for human disease, and highlights why computational pathogenicity predictors can fill an important gap in knowledge. For KCNQ1, additional information about reported variants could provide both diagnosis and treatment modalities for affected families.

The sequence length of the protein encoded by the KCNQ1 gene is 676 amino acids long, meaning that there are 676 * 19 amino acids different to wildtype = 12,844 possible missense mutations. I predicted values for all of these for each model, and I'm highlighting the results below in the form of heat maps. These visualizations are formatted like typical protein mutation screens - the x-axis is the amino acid index (where in the sequence we're making predictions for), while the y-axis shows what amino acid we are mutating that position to. The color bar represents our returned probability of being in the pathogenic class, according to the model. Red colors correspond to predicted pathogenic, while blue colors are predicted benign.

The heat map for Model 1 is shown in Figure 11 below.

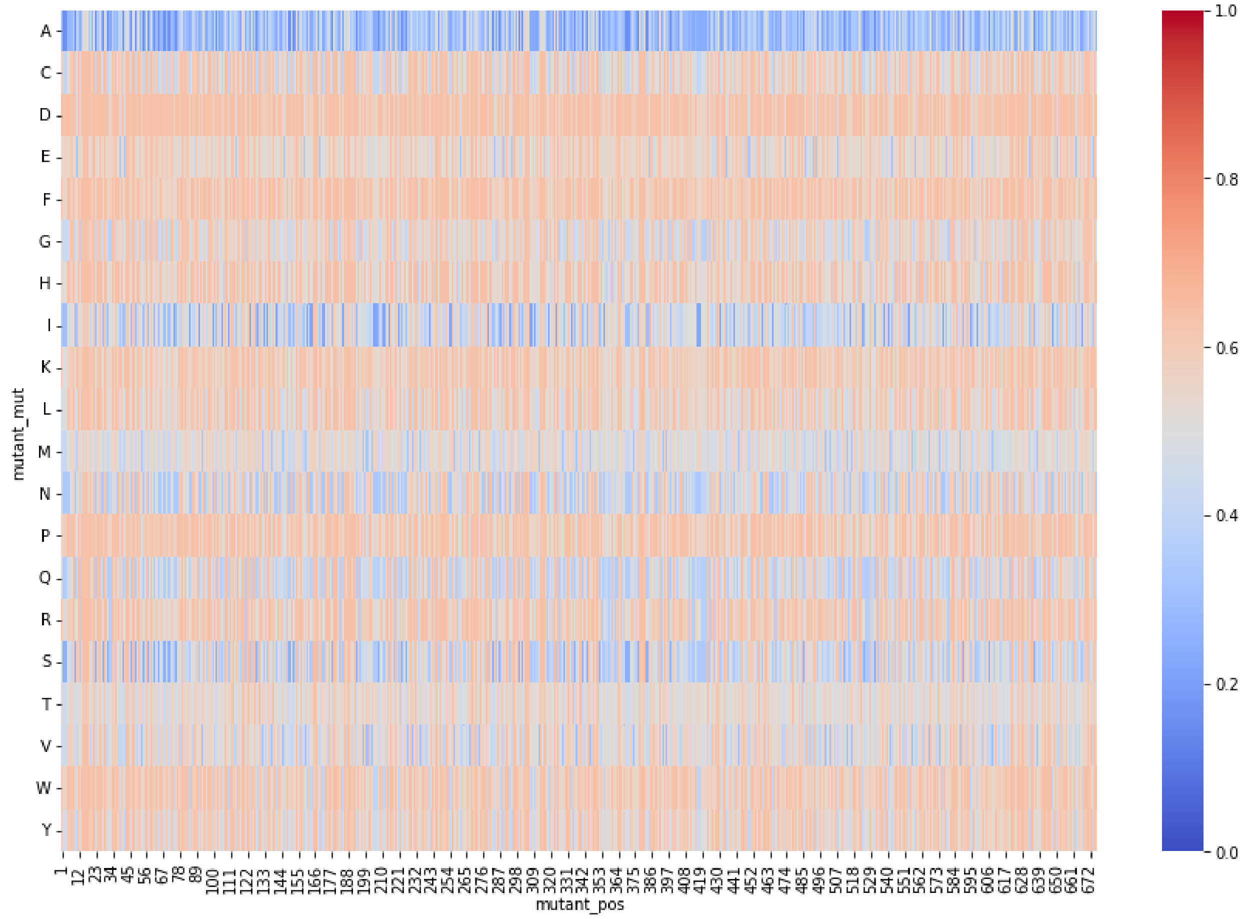


*Figure 11. Predicted pathogenicity (colorbar - red is more likely pathogenic, blue is more likely benign) for all possible positions (mutant_pos) and amino acid substitutions (mutant_mut) for KCNQ1 in Model 1.*

One thing that pops out right away is that most changes resulting in an alanine (single letter code A, the top row) are generally predicted to be benign. This matches our real world expectation that alanine is considered a fairly neutral mutation, to the point that an experimental technique called alanine scanning was developed to take advantage of this. On the other hand, the second row (mutations to cysteine, encoded as C) were on average less deleterious than I expected. Cysteines can form disulfide bonds with other cysteines, and adding in additional cysteine residues could change the physical properties of the protein. In the computational predictors I've worked with before, cysteine mutations tend to be marked as more deleterious on average than other amino acids, which makes this result somewhat surprising.

For Model 2, our RNN architecture with window subsequences, the heatmap for the model trained on the clustered sequences looks a bit different:

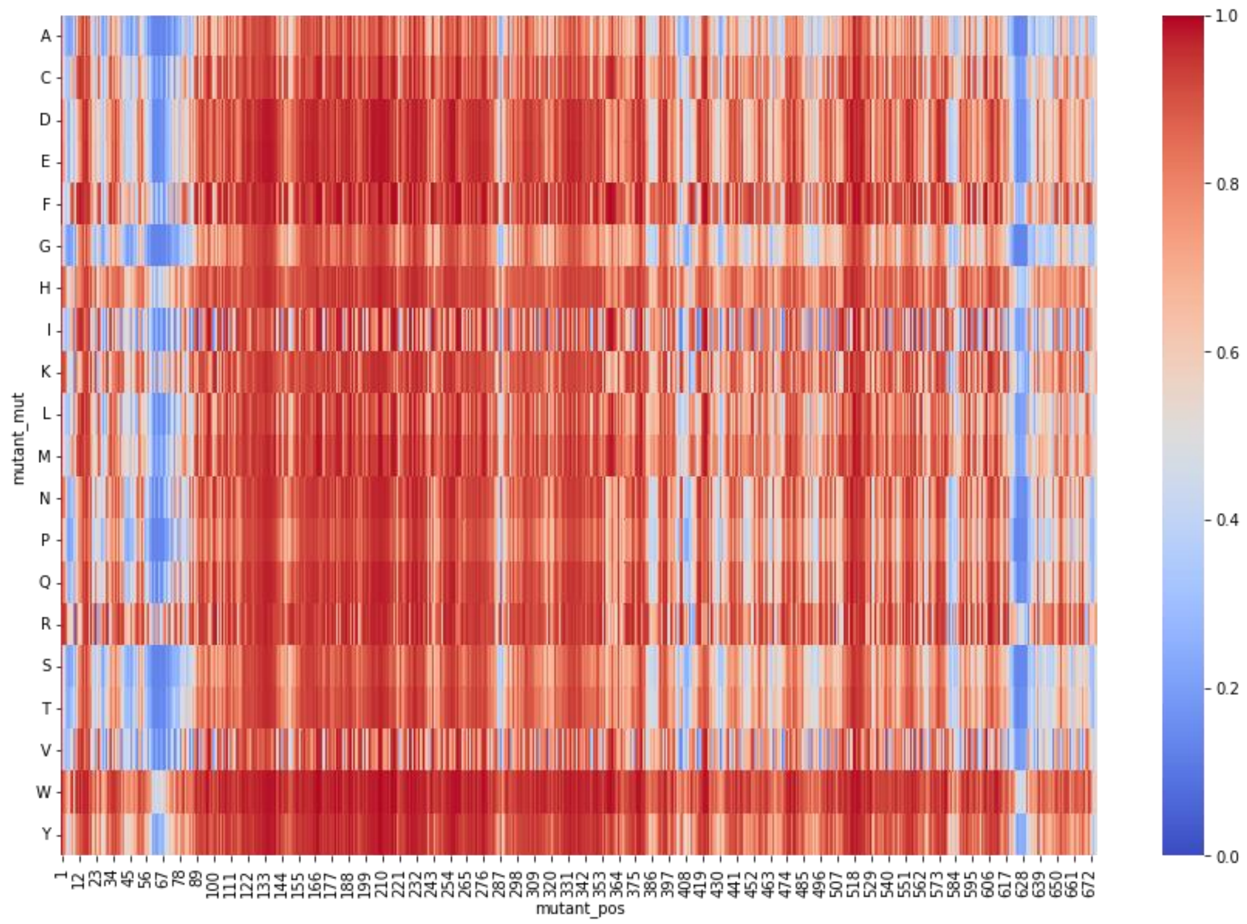


*Figure 12. Predicted pathogenicity (colorbar - red is more likely pathogenic, blue is more likely benign) for all possible positions (mutant_pos) and amino acid substitutions (mutant_mut) for KCNQ1 in Model 2.*

One thing to note is that this model tends to be a bit more confident in its predictions of pathogenicity, with probabilities near 1. Another interesting finding comes through when we consider the domains of this protein. Domains are regions of proteins that are thought to have evolved to have specific functions and tend to be modular units of protein sequences, and there are two annotated in KCNQ1 - an ion transporter and a voltage gated potassium channel. Qualitatively, it appears that a region from around residue 80 to 375 is particularly sensitive to mutations (in other words, there's a lot more red in this region). This is particularly interesting, because residues 122 to 359 form the transmembrane ion transporter region of the protein that spans the cellular membrane. The second domain in this protein, the voltage gated potassium channel, runs from residue 483 to 620. This roughly corresponds to a secondary vertical band of red. Of note, KCNQ1 was included in our train set, so the model had seen this sequence before. Although qualitative, these results indicate that this model is learning something about the protein sequence and using it to inform its variant effect predictions.

For Model 3, this time using the version trained on the random train/test split strategy, we see a similar banding, but the effect is more muted because the model overall seems less confident in its predictions:

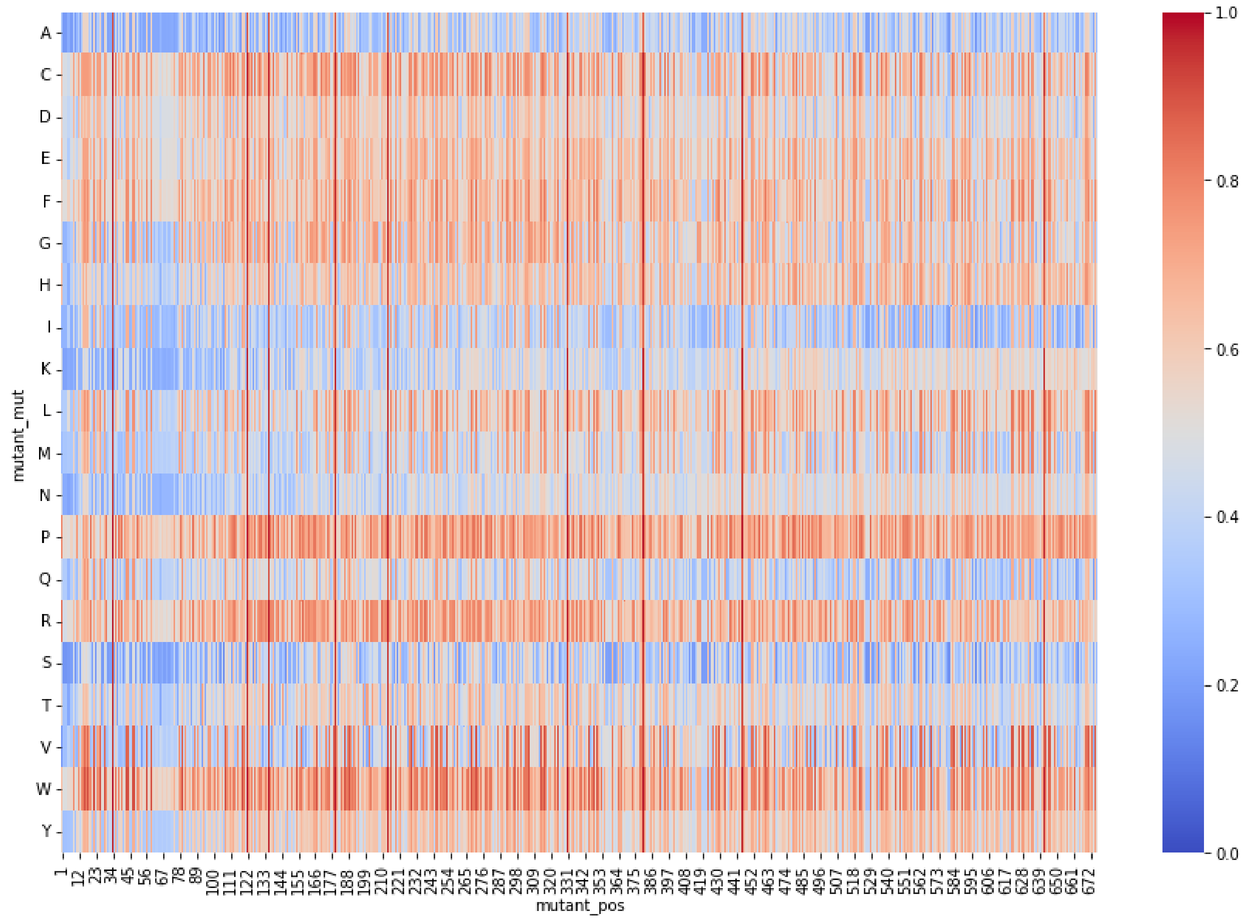


*Figure 13. Predicted pathogenicity (colorbar - red is more likely pathogenic, blue is more likely benign) for all possible positions (mutant_pos) and amino acid substitutions (mutant_mut) for KCNQ1 in Model 3.*

Our predicted mutations for Model 3 trend with the predictions for Model 2, as seen in the scatter plot shown in Figure 14:

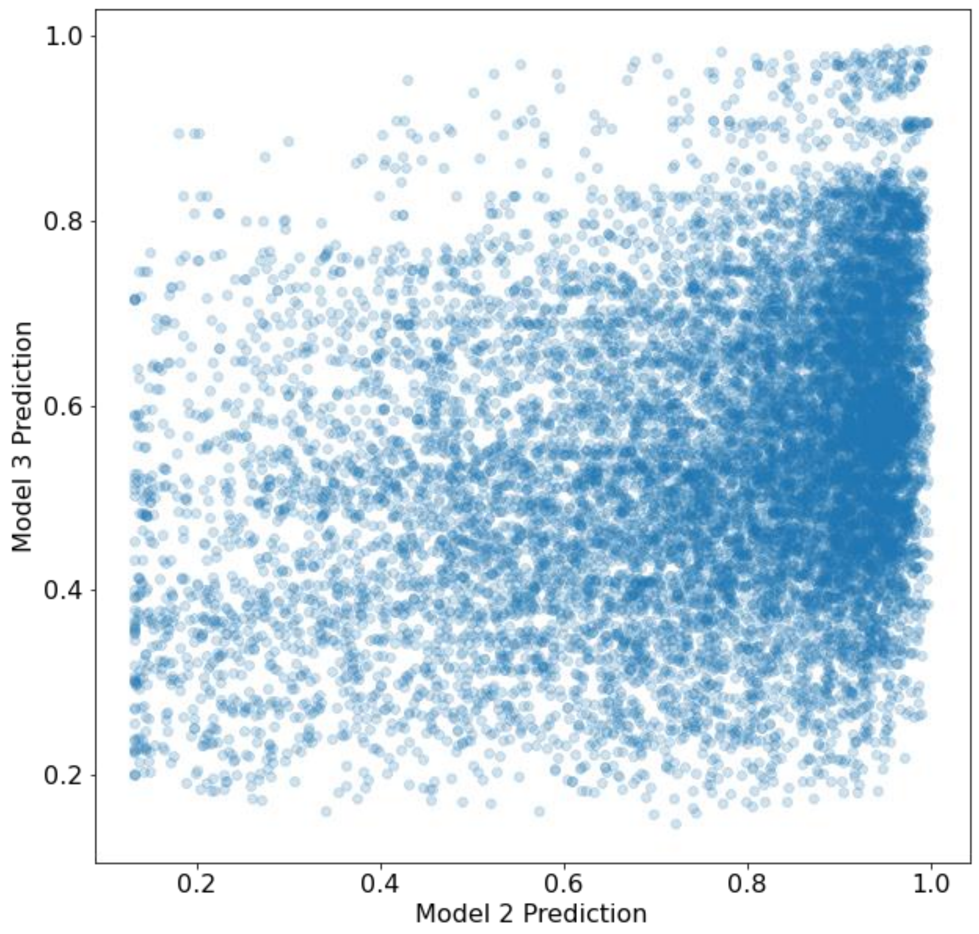


*Figure 14. Predicted probability of being pathogenic for Model 2 (x-axis) compared to Model 3 (y-axis).*

The Pearson correlation coefficient between the two models is 0.3, and the models agree (both above or both below 0.5) on classification for 62.4% of these predictions. Future work could include digging into where these models disagree, particularly the set of cases in the bottom right of the plot where Model 2 strongly predicts pathogenic but Model 3 predicts strongly benign. It is also worth noting that our training set had a small bias towards pathogenic mutations.

Some of this ability to pick out mutations as being particularly pathogenic if they are in the domains comes from the biased distribution of where our set of ClinVar variants are found in the sequence:

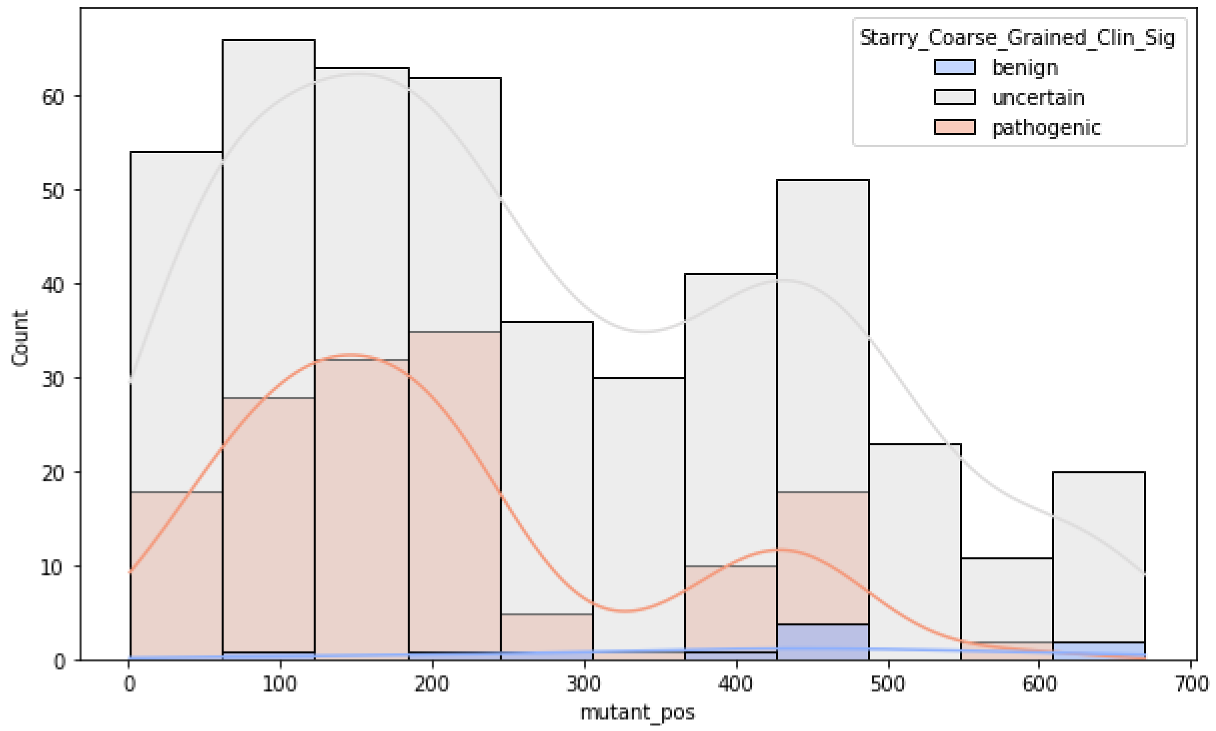


*Figure 15. Histograms of residue position of KCNQ1 variants reported in ClinVar, separated by annotation (benign, uncertain, or pathogenic).*

The pathogenic variants do fall in clusters within these regions, as do the variants of uncertain significance that the model was not trained on. However, we do have pathogenic variants at the beginning of the sequence outside of the first domain, and we have only variants of unknown significance in the region around residue 500 in the second domain. Therefore, while the model is learning something about the domain boundaries from the ClinVar data it sees, it seems to be learning additional information as well.

One final point of interest is to compare our missense variant effect predictions with how frequently those variants are found in the general population. This value is called the allele frequency. Generally, if the variant is rare - at the level of or below the frequency of the associated disease - then it is more likely to be pathogenic. (There are caveats here regarding heterozygosity versus homozygosity and the fact that dozens of mutations across multiple genes can cause long QT syndrome, but for these purposes I am using the disease prevalence as a rough order-of-magnitude measure). For this, we can use the gnomAD database as described in our "Details on data set -> Secondary data sources -> gnomAD Allele Frequencies'' section. Long QT syndrome, for reference, is estimated to affect on the order of 1 in 2,000 humans; mutations in KCNQ1 are the most common genetic contributor, although other related genes have also been implicated as well. Notably, only 72 mutations are shared between the 615 ClinVar variants and the 321 variants with reported allele frequencies, so our analysis of this set is heavily limited by the available data.

The scatter plots for allele frequencies versus predicted pathogenic probabilities for the three models described by the heatmaps are shown below:

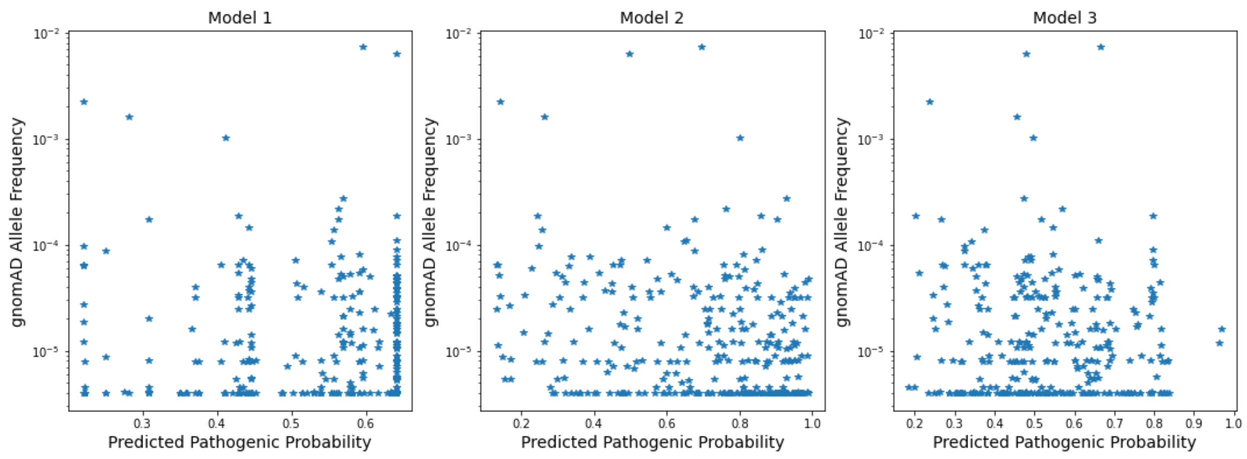


*Figure 16. Allele frequency from gnomAD compared to predicted pathogenic probability for the 3 models highlighted in this case study.*

We do not really see any correlation here for any of the models. However, it is worth noting that the vast majority of these allele frequencies are below the 1 in 2,000 disease prevalence cutoff that would be the maximum value for the variant to be associated with the disease. This finding can be visualized by looking at the distribution of allele frequencies for the 72 variants in common between gnomAD and ClinVar:

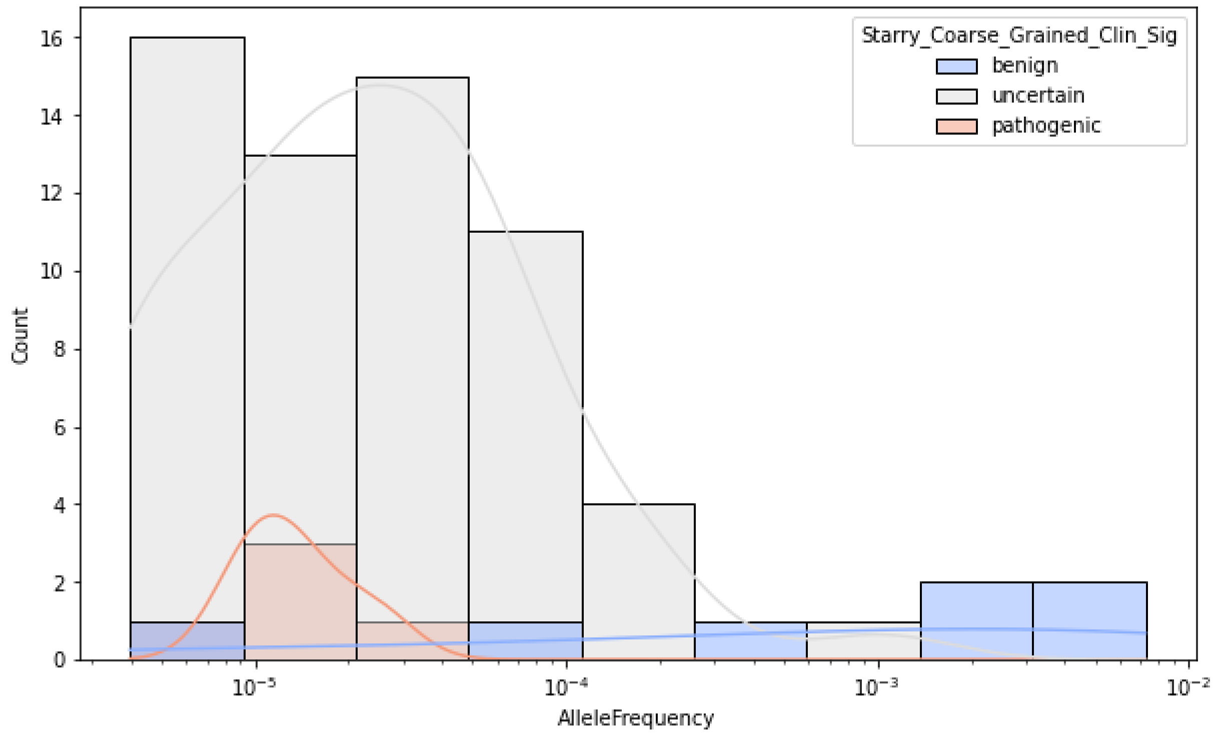


*Figure 17. Distributions of allele frequencies for KCNQ1 variants found in both ClinVar and gnomAD databases.*

From this plot, we see that benign mutations common to both databases occur most frequently (allele values > $10^{-3}$), while the handful of shared pathogenic mutations are all rare. However, we also find very rare variants (allele frequencies < $10^{-5}$) that have been annotated as benign as well. Therefore, it is hard to draw conclusions from comparing allele frequencies with predicted pathogenicity for this protein, due to the sparsity of data, overall prevalence of the disease, and because super rare variants can be benign with respect to the disease of interest.

## Discussion

### What worked (and what did not)

For Model 1, I was pleasantly surprised that the neural net produced reasonable results that were fairly stable across different parameter values. I was also happy to see that these values correlated with an industry standard for amino acid substitutions in a different use case. These results go to show that just looking at what kind of amino acid substitution is happening can provide useful information for pathogenicity prediction, and indeed this is one of the first things that clinical geneticists look at when interrogating a new variant.

Hyperparameter tuning for Model 2 was overall very challenging - this was the slowest model to run, and it also took quite a bit of fiddling with the regularizer and other hyperparameters to get the final model presented here. I think these parameters could be improved with further tuning, particularly of the window size variable. Furthermore, this model encoded amino acids based on 4 characteristics, as opposed to having 20 features in a one-hot encoding. It is possible that expanding how many features I use to represent amino acids could improve this model as well; I originally tried using just one feature - hydrophobicity - and saw much worse performance than this version where I used 4 values. Based on this, I think that this model has the potential to be improved substantially and could provide more flexibility than the other two models.

For Model 3, the most challenging part was trying to understand the paper describing the protein encodings that we used, and how to work with their published code. The earlier version of TensorFlow, which requires pre-defining a computational graph rather than using eager execution, was a particular sticking point. However, once that was running, this model was more stable compared to Model 2. It provided the overall best accuracy. The paper also provided larger dimensional encodings; it would be interesting to see how well a simple neural net like Model 3 trained on top of their larger encodings would perform.

### Lessons Learned

Deep learning is a powerful set of tools that can give valuable information about a wide range of applications, including pathogenicity prediction of missense variants. I also learned that training these models, particularly in the case of Model 2, is both an art and a science. Also in the context of Model 2, I learned how to think through what the optimal input would be for sequence processing. As an aside, another lesson was that I should try to make my code as abstract as possible from the beginning, instead of refactoring everything at the end. I feel much more comfortable using TensorFlow/Keras now, and I also gained a much better understanding of neural networks, their practical applications, and the scope of data needed through both this course and this final project. Thank you for an absolutely wonderful class!

## Works Cited

1. Frazer, J. *et al.* Disease variant prediction with deep generative models of evolutionary data. *Nature* **599**, 91–95 (2021).
2. Alley, E. C., Khimulya, G., Biswas, S., AlQuraishi, M. & Church, G. M. Unified rational protein engineering with sequence-based deep representation learning. *Nat. Methods* **16**, 1315–1322 (2019).
3. Landrum, M. J. *et al.* ClinVar: improving access to variant interpretations and supporting evidence. *Nucleic Acids Res.* **46**, D1062–D1067 (2018).
4. UniProt Consortium. UniProt: the universal protein knowledgebase in 2021. *Nucleic Acids Res.* **49**, D480–D489 (2021).
5. Fu, L., Niu, B., Zhu, Z., Wu, S. & Li, W. CD-HIT: accelerated for clustering the next-generation sequencing data. *Bioinformatics* **28**, 3150–3152 (2012).
6. Li, W. & Godzik, A. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics* vol. 22 1658–1659 (2006).
7. Kyte, J. & Doolittle, R. F. A simple method for displaying the hydropathic character of a protein. *J. Mol. Biol.* **157**, 105–132 (1982).
8. Henikoff, S. & Henikoff, J. G. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. U. S. A.* **89**, 10915–10919 (1992).
9. Karczewski, K. J. *et al.* The mutational constraint spectrum quantified from variation in 141,456 humans. *Nature* **581**, 434–443 (2020).
10. Moss, A. J. *et al.* Clinical aspects of type-1 long-QT syndrome by location, coding type, and biophysical function of mutations involving the KCNQ1 gene. *Circulation* **115**, 2481–2489 (2007).