Figure 1a: Reconstruction error for SMILES autoencoder based on TF-LSTM. We can observe that the model achieves low reconstruction errors (categorical cross entropy loss) w.r.t the validation set steeply (around ~200 epochs). However, the early stopping or optimal model
hyper-parameters are obtained for the TF-LSTM after 900 epochs where both training and validation loss are approximately 0.

Figure 1b. Reconstruction error for viral protein autoencoder based on CANN. We can observe that the model achieves low reconstruction errors (CCE) w.r.t the validation set steeply (around 20 epochs).
Figure 1c. The rate of convergence to optimal performance illustrated for the best randomization (out of 10) of the various end-to-end deep learning models.
Figure 2a: TF-LSTM Compound Autoencoder

Figure 2b: Protein Convolutional Neural Network Autoencoder
Figure 2c: Cross-validation performance of different machine learning methods w.r.t. R2 evaluation metric. Here "_LS_LS" stands for embedding representation learnt through TF-LSTM autoencoder for compounds and protein autoencoder for proteins and "_MFP_LS" corresponds to molecular fingerprint representation for compounds and embedding representation obtained through protein autoencoder. The mean R2 for GLM (0.537, 0.581), RF (0.660, 0.708), SVM (0.740, 0.808) and XGB (0.760, 0.825) methods are comparable to and have similar trends to performance of these models on the independent test set as depicted in Table 1.
| Model       | Optimal Configuration                                                                 |
|-------------|---------------------------------------------------------------------------------------|
| Random Forest | RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse', max_depth=8, max_features=0.7, max_leaf_nodes=None, max_samples=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=7, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=434, n_jobs=-1, oob_score=False, random_state=328, verbose=1, warm_start=False) |
| SVM         | SVR(C=2.1027036109989297, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.7433073841273525, kernel='rbf', max_iter=10000, shrinking=True, tol=0.001, verbose=False) |
| XGBoost     | XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bytree=1, gamma=0.0008026157228040275, importance_type='gain', learning_rate=0.05093845279011195, max_delta_step=0, max_depth=8, min_child_weight=4, missing=nan, n_estimators=378, n_jobs=-1, nthread=None, objective='reg:squarederror', random_state=0, reg_alpha=2.1941976179005755, reg_lambda=1.6232393459116288, scale_pos_weight=1, seed=None, silent=None, subsample=0.8233495190982153, verbosity=1) |
| CNN         | Seq2Func(  
|            | (protein_encoder): CNN_Encoder(  
|            | (embedding): Embedding(23, 64, padding_idx=1)  
|            | (convs): ModuleList(  
|            | (0): Conv1d(64, 128, kernel_size=(2,), stride=(1,))  
|            | (1): Conv1d(64, 128, kernel_size=(3,), stride=(1,))  
|            | (2): Conv1d(64, 128, kernel_size=(4,), stride=(1,))  
|            | (3): Conv1d(64, 128, kernel_size=(6,), stride=(1,))  
|            | (4): Conv1d(64, 128, kernel_size=(8,), stride=(1,))  
|            | (5): Conv1d(64, 128, kernel_size=(9,), stride=(1,))  
|            | (6): Conv1d(64, 128, kernel_size=(12,), stride=(1,))  
|            | (7): Conv1d(64, 128, kernel_size=(16,), stride=(1,))  
|            | )  
|            | (dropout): Dropout(p=0.1, inplace=False)  
|            | (batchnorm): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
|            | (fc): Linear(in_features=1024, out_features=256, bias=True)  
|            | )  
|            | (smiles_encoder): CNN_Encoder(  
|            | (embedding): Embedding(52, 64, padding_idx=1)  
|            | (convs): ModuleList(  
|            | (0): Conv1d(64, 128, kernel_size=(2,), stride=(1,))  
|            | (1): Conv1d(64, 128, kernel_size=(3,), stride=(1,))  
|            | (2): Conv1d(64, 128, kernel_size=(4,), stride=(1,))  
|            | )  
|            | ) |
| Conv1d(64, 128, kernel_size=(6,), stride=(1,)) |
| Conv1d(64, 128, kernel_size=(8,), stride=(1,)) |
| Conv1d(64, 128, kernel_size=(9,), stride=(1,)) |
| Conv1d(64, 128, kernel_size=(12,), stride=(1,)) |
| Conv1d(64, 128, kernel_size=(16,), stride=(1,)) |
| Dropout(p=0.1, inplace=False) |
| BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) |
| Linear(in_features=1204, out_features=256, bias=True) |
| Linear(in_features=512, out_features=128, bias=True) |
| Linear(in_features=128, out_features=1, bias=True) |
| Dropout(p=0.0, inplace=False) |

**LSTM**

Seq2Func(
  (protein_encoder): LSTM_Encoder(
    (embedding): Embedding(23, 64)
    (rnn): LSTM(64, 256, num_layers=2)
    (dropout): Dropout(p=0.0, inplace=False)
    (fc): Linear(in_features=512, out_features=128, bias=True)
  )
  (smiles_encoder): LSTM_Encoder(
    (embedding): Embedding(52, 64)
    (rnn): LSTM(64, 256, num_layers=2)
    (dropout): Dropout(p=0.0, inplace=False)
    (fc): Linear(in_features=512, out_features=128, bias=True)
  )
  (fc1): Linear(in_features=256, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=1, bias=True)
  (dropout): Dropout(p=0.0, inplace=False)
)

**CNN-LSTM**

Seq2Func(
  (protein_encoder): CNN_LSTM_Encoder(
    (embedding): Embedding(23, 64, padding_idx=1)
    (convs): ModuleList(
      (0): Conv1d(64, 64, kernel_size=(2,), stride=(1,))
      (1): Conv1d(64, 64, kernel_size=(3,), stride=(1,))
      (2): Conv1d(64, 64, kernel_size=(4,), stride=(1,))
      (3): Conv1d(64, 64, kernel_size=(6,), stride=(1,))
      (4): Conv1d(64, 64, kernel_size=(8,), stride=(1,))
      (5): Conv1d(64, 64, kernel_size=(9,), stride=(1,))
      (6): Conv1d(64, 64, kernel_size=(12,), stride=(1,))
      (7): Conv1d(64, 64, kernel_size=(16,), stride=(1,))
    )
    (rnn): LSTM(64, 128, num_layers=2)
    (dropout): Dropout(p=0.0, inplace=False)
  )
)
Table 1: Optimal configuration of the 3 traditional machine learning techniques and 3 end-to-end deep learning models utilized in our supervised learning framework for compound-virus activity prediction.

**Compound Autoencoder: TF-LSTM**

The goal of a compound autoencoder model (Kramer, 1991) is to learn the innate low dimensional representation $L_S^c$ from SMILES strings of compounds ($x^c$) in an unsupervised setting such that compounds with similar patterns tend to be closer in the low dimensional space. Our compound autoencoder framework consists of an encoder, a decoder, and a sequence to sequence (seq2seq) model which encapsulates the encoder and decoder and provides a way to interface with each. The encoder consists of a multi-layered LSTM (Gers et al., 1999) which overcomes limitations like vanishing gradients experienced by a traditional recurrent neural network (RNN) model (Dupond, 2019). The output of LSTM encoder can be represented as $(h, c) = \text{EncoderLSTM}(e(x^c))$. Here $e(x^c)$ represents the embedding representation for compound, $h$ and $c$ correspond to hidden state representations encapsulating sequential information.

The decoder component does a single step of decoding i.e. it outputs single ($\hat{y}^{cv}_t$) token per time-step $t$. Since, we are building a compound autoencoder model, $\hat{y}^{cv}_t = x^c_t$ i.e. the vector corresponding to the $t^{th}$ character in the compound representation $x^c$: The decoder can mathematically be depicted as $s^t=\text{DecoderLSTM}(x^c_t, (h, c))$. The hidden state $s^t$ obtained from
Decoder LSTM is passed through a linear layer $f$ to make a prediction for the next token in the target sequence i.e. $\hat{y}_{t+1} = f(s^t)$.

Our seq2seq method takes the source compound representation ($x_c$), target compound representation ($x_t$) and a teacher-forcing ratio. The teacher forcing ratio is used when training our model. When decoding, at each time-step we predict what the next token in the target sequence will be from the previous tokens $t+1$ decoded, $\hat{y}_{cv} = f(s^t)$. With probability $1 - \text{teacher forcing ratio}$, we will use the token that the model predicted as the next input to the model, even if it doesn’t match the actual next token in the sequence. The latent space representation $LS_c$ for a given compound is equivalent to the hidden state representation $h$ for our TF-LSTM model.

We trained this TF-LSTM model on $\approx 2.5$ million SMILES strings for small molecules. During the training phase, the teacher forcing ratio is set to 0.5 and during the test phase of our experiments, it is set to 0. Interestingly, 96.7% of the SMILES generated by our TF-LSTM model were valid small molecules (tested using RDKit (Landrum, 2013) package) and had a mean categorical cross-entropy (Goodfellow et al., 2016) error of 0.004. The convergence of the reconstruction error for our TF-LSTM model is depicted in Figure 1a. Figure 2a illustrates our TF-LSTM compound autoencoder model.

**Protein Autoencoder: CNN**

The goal of the viral protein autoencoder model is to learn a low dimensional representation $LS_v$ from the amino acid sequences of viral proteins $x_v$. We used a convolutional autoencoder neural network for this purpose. Our protein autoencoder framework consists of two main components: an encoder and a decoder as shown in Figure 2b. The autoencoder was trained in an unsupervised fashion to learn a low dimensional space ($LS_c$).

The encoder consists of multi-layered convolution and subsampling layers followed by a fully connected layer. The purpose of using the convolution layers is to extract features that preserve input neighborhood interactions and spatial locality, which is important to capture local protein structures and frequently occurring k-mers. The max-pooling layers are used for subsampling to obtain translation-invariant representations and reduce the number of convolution filters required resulting in a lesser number of trainable parameters. The max-pooling layers also perform regularization and help to generalize the learned latent space. The decoder consists of multi-layered deconvolution and upsampling layers preceded by a fully connected layer. These layers perform the inverse function of the encoder layers in the reverse order to generate the initial input.

We trained our autoencoder on 2,658,225 viral proteins. The mean categorical cross-entropy (Goodfellow et al., 2016) error for the autoencoder was 0.1. The convergence of the reconstruction error for the autoencoder is depicted in Supplementary Figure 1b.
a) GLM (SMILES) Model  
MSE=0.875  
MAE=0.665  
Pearson r=0.740  
R²=0.548

b) RF (SMILES) Model  
MSE=0.630  
MAE=0.559  
Pearson r=0.824  
R²=0.679

c) SVM (SMILES) Model  
MSE=0.479  
MAE=0.508  
Pearson r=0.869  
R²=0.755

d) XGB (SMILES) Model  
MSE=0.420  
MAE=0.452  
Pearson r=0.885  
R²=0.784

e) GLM (MFP) Model  
MSE=0.779  
MAE=0.648  
Pearson r=0.773  
R²=0.598

f) RF (MFP) Model  
MSE=0.551  
MAE=0.530  
Pearson r=0.849  
R²=0.720

g) SVM (MFP) Model  
MSE=0.359  
MAE=0.439  
Pearson r=0.904  
R²=0.817

h) XGB (MFP) Model  
MSE=0.327  
MAE=0.404  
Pearson r=0.912  
R²=0.831

i) CNN Model  
MSE=0.399  
MAE=0.452  
Pearson r=0.892  
R²=0.795
Figure 3: Comparison of predictive performance of the optimal version of each ML model for 4 evaluation metrics on the test set D_{test}. The best performance w.r.t. R2 is observed for the XGBoost (MFP) model. However, the CNN and GAT-CNN deep learning model's performance is highly competitive to the XGBoost (MFP) model.

Table 2: Main proteases of SARS-COV-2 virus targeted for inhibition by our data-driven compound repurposing framework.

| Uniprot Id | PDB Id | Protein Fragment | Sequence | L |
|------------|--------|------------------|----------|---|
| P0DTD1     | 6W02   | PL-PRO (NSP3)    | GEYNSFSOYLKLTONNYYNNDQEVE EAKKVKPTVVNANAYLHKQGGDIV AGAILNATNNAAMQVESDHYATNG PLKVQGSGCIVLDHNNLAKHQLHIGV PNYVNYGIGSLEGKLKSAVYENFNGNHEV LLAPLSAIGGASPPHRLVRCVDIV  | 170 |
| P0DTD1     | 5R7Y   | 3CL-PRO          | SQRFKMAFSSFQVGGMVQVTCG TTTLNGLWLDVXVCPHRIVCTSE DMNNPPYCCCLLRKSSHMLPDFAG NVQLRVHGSMQSNVYKLKVQSTAN PKTRPKYRVRGPGQTVSILACYN GSPSGVYGCGCMRNPFTFSGFSLNLG SCSGVSFGQDQVNCFCYMNHPINHEL PTOGWAQTDIEMTVQGVPQYQVOTA QAGQ7DTPTTVNLWLYYAAYINID RVKLRHRFFTTLTILNLVAMKYNVE PLTDQHQDGLPLSAGITQWLMDC ASLKKELLQGANGRTLIQSALEDE FTPFQDVVRCGQVTFQ  | 306 |
| P0DTC2     | 6MOJ   | Spike Protein    | TNLQFQGEFVNATRPAFABYVAYNKR RNSICYAZSYVLNYSASFSTFPKCYG VSPKTNOOCLFTNYAODSFRVDGG VRQAGTSGCDXKHDYKYPFSDMTQ CVKWAAMNNLSKGVQNYNLYRFLRKNHUPFPERIDESYDQAIDPC HNOEGFCYFQLLQSGYQGQPTGNG YQPYRVRVLSFELLHAPATCYG  | 229 |
List of Compounds and All predictions

The list of FDA approved and investigational compounds used in our work is available at: https://github.com/raghvendra5688/Drug-Repurposing/blob/master/data/COVID-19/all_verified_keys.list details on compound accumulation available in the README (in the data folder).

The predictions for all these compounds for the PL-PRO, 3CL-PRO and Spike protein of SARS-COV-2 are available at:

1. https://github.com/raghvendra5688/Drug-Repurposing/blob/master/results/PL_Pro_Top_Ranked_Compounds.csv
2. https://github.com/raghvendra5688/Drug-Repurposing/blob/master/results/3CL_Pro_Top_Ranked_Compounds.csv
3. https://github.com/raghvendra5688/Drug-Repurposing/blob/master/results/Spike_Pro_Top_Ranked_Compounds.csv

Molecular Docking Experiment Setup

In order to perform molecular docking, the structure coordinates of the viral proteins of SARS-CoV-2 were retrieved from PDB database (www.rcsb.org/pdb). We selected the following PDB structures for docking: a) PDB ID: 6W02 for PL-Pro; b) PDB ID: 5R7Y for 3CL-Pro; c) PDB ID: 6M0J for receptor-binding domain (RBD) of Spike protein.

We next removed heteroatoms and water molecules from the structure before performing the docking experiments. The selected compounds were retrieved from PubChem database and imported into OpenBabel (http://openbabel.org/) with PyRx (Dallakyan and Olson 2015) and subjected to energy minimisation. The energy minimisation was performed with the Universal Force Field (UFF) (Ogawa and Nakano 2010) using a conjugate gradient (Pytlak 2008) algorithm. The total number of steps was set to 200 and the number of steps for update set to 1. Structure-based docking simulations were performed using the tool AutoDock Vina (Trott and Olson 2010) compiled in PyRx. The docking simulation was run at an exhaustiveness of 8 and set to output only the lowest energy pose. We finally utilized UCSF Chimera (Pettersen et al. 2004) to visualize the docked poses.
References:

Dallakyan, Sargis, and Arthur J. Olson. 2015. “Small-Molecule Library Screening by Docking with PyRx.” *Methods in Molecular Biology* 1263: 243–50.

Ogawa, Tetsuji, and Tatsuya Nakano. 2010. “The Extended Universal Force Field (XUFF): Theory and Applications.” *Chem-Bio Informatics Journal*. https://doi.org/10.1273/cbij.10.111.

Pettersen, Eric F., Thomas D. Goddard, Conrad C. Huang, Gregory S. Couch, Daniel M. Greenblatt, Elaine C. Meng, and Thomas E. Ferrin. 2004. “UCSF Chimera—a Visualization System for Exploratory Research and Analysis.” *Journal of Computational Chemistry* 25 (13): 1605–12.

Pytlak, Radoslaw. 2008. *Conjugate Gradient Algorithms in Nonconvex Optimization*. Springer Science & Business Media.

Trott, Oleg, and Arthur J. Olson. 2010. “AutoDock Vina: Improving the Speed and Accuracy of Docking with a New Scoring Function, Efficient Optimization, and Multithreading.” *Journal of Computational Chemistry* 31 (2): 455–61.