EFFICIENT DRUG DISCOVERY WITH LSTM-BASED MODELS: INSIGHTS FROM SARS-COV-2 VARIANTS

MONCHI ESTEVEZ

Abstract. The rapid evolution of SARS-CoV-2 variants underscored the need for accelerated drug discovery methods. This study demonstrates the use of recurrent neural networks (RNNs) with Long Short-Term Memory (LSTM) units to generate novel pharmaceutical compounds capable of inhibiting SARS-CoV-2 through protein binding, using variants (Alpha, Beta, Gamma, Delta) as reference points. Three LSTM-based RNN models were developed, trained on a dataset of 2,572,812 preprocessed SMILES (Simplified Molecular-Input Line-Entry System) sequences from the ChEMBL 29 and MOSES databases, and fine-tuned against these variants. The models, differing in dropout regularization parameters, were evaluated for validity, originality, and uniqueness of generated molecules, with performance assessed via simulated protein binding affinity scores using PyRx. Results demonstrate that Model 3, with the lowest dropout values (0.2 and 0.4), outperformed others, achieving a 98.0% validity rate, 94.1% originality, and 97.9% uniqueness, and generating molecules with high binding affinities (e.g., -17.40 kcal/mol). These findings highlight the efficacy of LSTM-RNNs in automating and optimizing drug discovery, potentially offering a scalable, efficient alternative to traditional methods. Further laboratory validation is recommended to translate these computational results into practical therapeutic applications.

1. Introduction. The development of pharmaceutical drugs has historically been a slow and resource-intensive process, yet the twentieth century marked a significant shift, often regarded as a "golden era" for drug discovery due to advancements in technology, particularly in computer science and data science. Traditional methods, while foundational, have been augmented by tools like high-throughput screening, which accelerate the identification of potential therapeutic compounds. Nevertheless, the chemical space remains vast—estimated to encompass between 10^{90} and 10^{100} drug-like molecules—posing a significant barrier to exhaustive exploration with conventional approaches [7]. The COVID-19 pandemic exemplified this challenge, highlighting the urgent need for rapid development of treatments to address the SARS-CoV-2 virus and its evolving variants. Despite swift progress in vaccine development, screening expansive chemical libraries for effective antiviral drugs continues to demand substantial time and funding, necessitating more efficient methodologies.

Machine learning offers a promising avenue for enhancing drug discovery by enabling algorithms to infer patterns from data and adapt to new inputs without explicit programming. Within this field, artificial neural networks (ANNs) emulate biological neural systems through interconnected nodes, akin to axons and dendrites, processing information across input, hidden, and output layers [6]. Unlike traditional ANNs, which propagate data unidirectionally, recurrent neural networks (RNNs) incorporate feedback loops, allowing information to cycle between nodes within and across layers. This architecture is particularly suited for sequential data, with node weights optimized through backpropagation, a process that adjusts connections by computing the gradient of a loss function. However, standard RNNs face limitations due to the vanishing gradient problem, where gradients diminish over long sequences, impeding learning across distant layers [6]. Long Short-Term Memory (LSTM) units address this by introducing gating mechanisms—input, forget, and output gates—that regulate information flow, preserving critical patterns and mitigating gradient decay.

These computational techniques hold significant potential for drug discovery, particularly in navigating large molecular databases and generating novel chemical structures. RNNs, enhanced by LSTM units, excel at processing sequential representations, such as those encoded in the Simplified Molecular-Input Line-Entry System (SMILES) format, making them ideal for modeling molecular sequences. The scale of chemical space and the pressing need for rapid therapeutic solutions, as demonstrated by the SARS-CoV-2 outbreak, underscore the value of such approaches. This study investigates the application of LSTM-based RNNs to generate pharmaceutical compounds designed to inhibit SARS-CoV-2 through protein binding, using its variants (Alpha, Beta, Gamma, Delta) as reference points. We present the design, training, and evaluation of three LSTM-RNN models, each with distinct architectural configurations, to assess their ability to produce viable drug candidates. By leveraging extensive chemical datasets and fine-tuning against SARS-CoV-2 variant data, this research demonstrates how these models can streamline the identification of molecules with high binding affinity, providing an efficient alternative to traditional drug discovery methods.

2. Data Collection. The datasets utilized in this study were divided into training and testing sets to facilitate model development and evaluation. The training set, which forms the basis for learning underlying patterns and features, was constructed from two comprehensive chemical databases: ChEMBL 29 [4] and the Molecular Sets (MOSES) dataset [5]. These repositories provide molecular structures in the Simplified Molecular-Input Line-Entry System (SMILES) format, a text-based representation of chemical compounds suitable for computational processing. Collectively, the raw data from these sources

comprised 4,251,849 unique SMILES strings. The training set enables the models to map base principles, while the testing set, consisting of unseen data, serves to evaluate generalization performance. Model efficacy was assessed through simulated protein binding scores, calculated for each generated molecule against the SARS-CoV-2 variants, with higher binding accuracy indicating greater potential for viral inhibition [2].

The SARS-CoV-2 variants utilized as reference points in this study—Alpha (B.1.1.7), Beta (B.1.351), Gamma (P.1), and Delta (B.1.617.2)—were selected due to their distinct spike protein mutations, which influence binding affinity to molecular inhibitors [1]. These variants, identified during the COVID-19 pandemic, exhibit variations in their receptor-binding domains, providing diverse structural targets for evaluation. Data for these variants, including their protein sequences, were sourced from public repositories (GISAID) and processed into a format compatible with binding affinity simulations.

To ensure data quality, preprocessing and filtering were applied to the raw SMILES dataset. This step eliminated duplicates, null entries, and extraneous chemical entities such as salts and acids, while retaining only relevant molecular information. Each SMILES string was tokenized into arrays of characters, converting the sequences into a format compatible with Python-based processing. Token length was restricted to a range of 32 to 128 characters to balance complexity and computational feasibility, resulting in a refined dataset of 2,572,812 SMILES strings for training. A simplified version of this preprocessing script is shown in Figure 2.1.

```
from rdkit import Chem, RDLogger, Preprocessor  # Import RDKit library

def preprocessing(input, output, **kwargs):
  with open(input, 'r') as file: # Open file with SMILES
    file_smiles = [row.rstrip() for row in file] # Read SMILES

  to_do_smiles = [Preprocessor(smile) for smile in file_smiles] # Normalize
  output_smiles = []  # Store cleaned SMILES
  if kwargs['clean']:
    for one_smile in to_do_smiles:  # Iterate through SMILES
      try:
        token_smile = Preprocessor.tokenize(one_smile)  # Tokenize SMILES
        if 32 <= len(token_smile) <= 128:  # Filter by length
          output_smiles.append(one_smile)  # Add valid SMILES
```

FIG. 2.1. *Simplified SMILES preprocessing script implemented using RDKit.*

3. Model Architecture. The models developed in this study leverage recurrent neural networks (RNNs) enhanced with Long Short-Term Memory (LSTM) units to generate novel molecular sequences in SMILES format. Traditional RNNs, while effective for sequential data, suffer from the vanishing gradient problem, where gradients diminish across long sequences, limiting the contribution of distant layers to parameter updates [6]. To address this, LSTM units incorporate input, forget, and output gates that regulate information flow, allowing critical gradients and patterns to persist across extended sequences. This architecture is particularly suited for learning the probability distribution of SMILES tokens, enabling the models to predict the next token in a sequence and ultimately generate complete molecular structures from scratch. In this context, the models aim to maximize the likelihood of selecting optimal tokens tailored to inhibit SARS-CoV-2 through protein binding.

Input SMILES sequences, tokenized during preprocessing, are encoded as one-hot vectors, where each token is represented by a binary vector with a single non-zero entry corresponding to its position in the vocabulary. This encoding process is illustrated for a benzene molecule in Figure 3.1, showing its SMILES representation and corresponding one-hot encoded form.

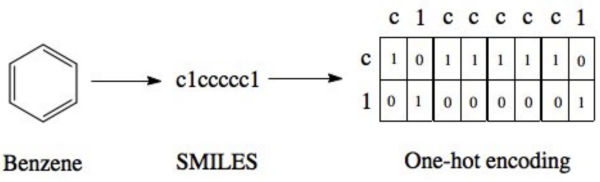

FIG. 3.1. *The SMILES representation (e.g., `c1ccccc1`) and one-hot encoding for a benzene molecule.*

A simplified script implementing this one-hot encoding process is provided in Figure 3.2, demonstrating how SMILES sequences are transformed into a format suitable for LSTM processing.

```
from rdkit import Chem
import numpy as np

SMILES_CHARS = ['#', '%', '(', ..., 'A', 'B', 'C']  # Partial vocabulary
index = dict((w, i) for i, w in enumerate(SMILES_CHARS))

def encoding(smiles, max_len=128):
  smiles = Chem.MolToSmiles(Chem.MolFromSmiles(smiles))
  hot_encoded = np.zeros((max_len, len(SMILES_CHARS)))
  for i, w in enumerate(smiles):
    hot_encoded[i, index[w]] = 1 # Mark index of integer with 1 in array

  return hot_encoded
```

FIG. 3.2. *Simplified one-hot encoding script for SMILES sequences in Python 3.7.11.*

This encoding facilitates processing by the LSTM layers, which output a list of potential next tokens. To convert outputs into a probability distribution, a softmax activation function is applied, defined as:

$$P(y_i) = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_j)}, \tag{3.1}$$

where y_i is the score for the i-th token and n is the vocabulary size. This normalization ensures that the model assigns probabilities to each possible next token, selecting the one with the highest value. The loss function, used to evaluate and update model weights, is categorical cross-entropy, suitable for multi-class classification with one-hot encoded targets. It is expressed as:

$$\text{Loss} = -\sum_{i=1}^{n} y_i \cdot \log \hat{y}_i, \tag{3.2}$$

where y_i is the true label (1 or 0) and $\hat{y}_i$ is the predicted probability for the i-th token. This function quantifies the difference between predicted and actual distributions, guiding weight adjustments via backpropagation.

Three distinct LSTM-based RNN models were constructed, each following a one-to-many sequence architecture, producing multiple output tokens from a single input sequence. Each model comprises two LSTM layers, regularized with dropout to mitigate overfitting by probabilistically excluding connections during training, followed by a dense layer and the softmax activation function. The dense layer performs matrix-vector multiplication on the outputs of the LSTM layers, ensuring dimensional compatibility with the vocabulary size. All models maintain a hidden state size of 256 units for standardization, but differ in dropout values to explore their impact on performance. Model 1 employs dropout rates of 0.6 and 0.8 for the first and second LSTM layers, respectively; Model 2 uses 0.4 and 0.6; and Model 3 uses 0.2 and 0.4. These configurations allow the models to balance regularization and learning capacity.

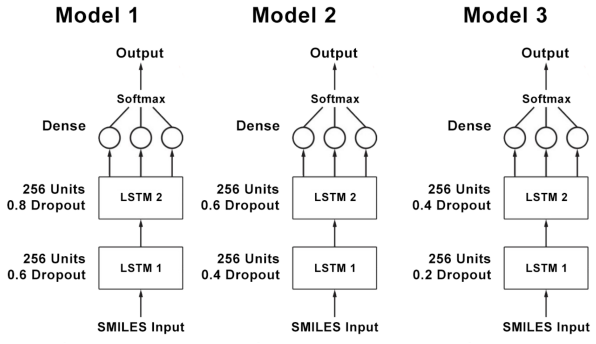

FIG. 3.3. *Representation of the three LSTM models with a one-to-many sequence architecture.*

The overall process involves one-hot encoding of the input SMILES, which is fed into the LSTM layers to produce a list of predicted possible next tokens in the sequence. The softmax activation function evaluates this list, computing a probability value for each token, and selects the token with the highest probability as the best fit for the sequence. At the end of the learning process, the loss function is averaged over all target tokens, and based on this loss, the model adjusts its learning parameters to minimize the loss value, thereby increasing the likelihood of selecting the optimal token. An example of this process, applied to predicting tokens for a benzene molecule with the SMILES sequence `c1ccccc1` (missing the final token), is depicted in Figure 3.4.

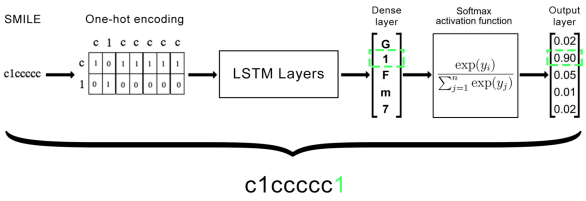

FIG. 3.4. *Example of the overall model process for learning to predict tokens for a benzene molecule (`c1ccccc1`), where the sequence is incomplete (missing the final token), and the model correctly selects the next token.*

4. Model Creation and Training. The development and training of the LSTM-based RNN models were implemented using TensorFlow 1.2 and Keras 2.0, selected for their robust support of deep learning architectures and efficient handling of sequential data. Molecular processing and SMILES validation were facilitated by the RDKit cheminformatics library, which provided tools for parsing, validating, and manipulating chemical structures critical to this study. This process encompassed model creation, training on the preprocessed SMILES dataset, and fine-tuning to generate molecules optimized for SARS-CoV-2 inhibition. The workflow was divided into distinct stages—configuration and data loading, model building and training, and molecule generation with subsequent evaluation—to ensure a systematic approach to optimizing the models' generative capabilities. This structured methodology allowed for iterative refinement and scalability, aligning with the study's objective of accelerating drug discovery through computational means.

Model creation began with defining configuration settings for each of the three models, specifying parameters such as the number of epochs, batch size, optimization strategy, and learning rate schedules. These settings were tailored to balance computational efficiency with model convergence, critical for handling the large-scale SMILES dataset. The training dataset of 2,572,812 SMILES strings, sourced from ChEMBL 29 and MOSES databases, was divided into batches of 512, yielding approximately 5,025 iterations per epoch. This batch size was chosen to optimize GPU memory usage while maintaining gradient stability during backpropagation. An arbitrary limit of 42 epochs was initially set to provide sufficient training cycles, though a dynamic stopping criterion was implemented, halting training when the validation loss stabilized—typically around 26 epochs, as detailed later—to prevent overfitting and conserve computational resources. A sample configuration file for Model 1, written in JSON format, is shown in Figure 4.1, illustrating key parameters including the Adam optimizer, selected for its adaptive learning rate properties [3], and loss monitoring via validation loss to track model generalization.

```
{
  "exp_name": "LSTM_Model_1",
  "data_filename": "./data/training_smiles.smi",
  "num_epochs": 42,
  "optimizer": "adam",
  "batch_size": 512,
  "checkpoint_monitor": "val_loss",
  "checkpoint_save_best_only": false,
  "checkpoint_save_weights_only": true,
  "smiles_max_length": 128,
  "finetune_epochs": 25,
  "finetune_batch_size": 1
}
```

FIG. 4.1. *Shortened configuration file (JSON) for Model 1, defining training and fine-tuning parameters.*

Data loading was managed by a custom script that supported three modes: training, testing, and fine-tuning. This script, shown in Figure 4.2 for Model 1, imported the training dataset in training mode, utilized unseen data for validation in testing mode, and incorporated SARS-CoV-2 variant data in fine-tuning mode to adapt the models to specific protein binding targets. The script tokenized and one-hot encoded the SMILES sequences, preparing them as input arrays for the LSTM layers.

```
class DataLoader(Sequence):
  def __init__(self, config, data_type='train'):
    self.config = config
    self.data_type = data_type
    assert self.data_type in ['train', 'test', 'finetune']
    if self.data_type == 'train':
      self.smiles = self.load(self.config.data_filename)
    elif self.data_type == 'finetune':
      self.smiles = self.load(self.config.finetune_data_filename)
    else:
      pass

    self.tokenized_smiles = self.tokenize(self.smiles)

  def load(self, data_filename):
    length = self.config.data_length
    with open(data_filename) as f:
      smiles = [s.rstrip() for s in f]
    return smiles

  def tokenize(self, smiles): # Tokenization and encoding
    tokenized_smiles = [self.encoding(molecule) for molecule in smiles]
    ...
    return tokenized_smiles
```

FIG. 4.2. *Shortened data loader script for Model 1 in Python 3.7.11, supporting training, testing, and fine-tuning modes.*

Model building was automated using a TensorFlow script that constructed each model's architecture, initialized weights with a random normal distribution, and saved checkpoints for later use. The script for Model 1, presented in Figure 4.3, defined the two LSTM layers with their respective dropout rates (0.6 and 0.8), followed by the dense and softmax layers, and compiled the model with the categorical cross-entropy loss and Adam optimizer.

```
class Model1(object):
  def build_model(self):
    st = encoding()
    n_table = len(st.table)
    weight_init = RandomNormal(mean=0.0, stddev=0.05, eed=self.config.seed)
    self.model = Sequential()

    self.model.add(
      LSTM(units=self.config.units, input_shape=(None, n_table),
      return_sequences=True, kernel_initializer=weight_init, dropout=0.8))

    self.model.add(
      LSTM(units=self.config.units, input_shape=(None, n_table),
      return_sequences=True, kernel_initializer=weight_init, dropout=0.6))

    self.model.add(Dense(units=n_table, activation='softmax',
        kernel_initializer=weight_init))
    self.model.compile(optimizer=self.config.optimizer, loss='
        categorical_crossentropy')
```

FIG. 4.3. *Shortened model builder script for Model 1 in Python 3.7.11, defining the LSTM architecture and compilation.*

The built model, trainer script, training and testing dataset were combined with Keras and TensorFlow to train the model using the fit generator function. This function fitted the model on training data yielded batch-by-batch with a Python generator, which ran in parallel. The function also used parameters from setting files, including the number of epochs, verbose, and steps per epoch. The trainer script for Model 1 is shown in Figure 4.4.

```
class ModelTrainer(object):
  def __init__(self, modeler, train_data, test_data):
    self.model = modeler.model # Load model object
    self.config = modeler.config # Load model configuration file
    self.train_data_loader = train_data # Load training dataset
    self.test_data_loader = test_data # Load testing data
    self.callbacks = [] # Callback to save the model and weights at intervals.
    self.init_callbacks()

  def train(self):
    history = self.model.fit_generator(
      self.train_data_loader,
      steps_per_epoch=self.train_data_loader.__len__(),
      epochs=self.config.num_epochs, # Set number of epochs
      verbose=self.config.verbose_training, # Set verbose
      testing_data=self.test_data_loader, # Set testing data
      testing_steps=self.test_data_loader.__len__(),
      use_multiprocessing=True, # Allow multiprocessing for efficiency
      shuffle=True,
      callbacks=self.callbacks)

    last_weight_file = max(glob(self.config.checkpoint_dir), key=os.path.
        getctime)[0]
    self.config.model_weight_filename = last_weight_file

    with open(os.path.join(self.config.exp_dir, 'config.json'), 'w') as f:
      f.write(self.config.toJSON(indent=2))
```

FIG. 4.4. *Trainer script for Model 1 in Python 3.7.11, using TensorFlow and Keras to fit the model with training data.*

Training of the LSTM-based RNN models was conducted by integrating the model builder, data loader, and trainer scripts to process the 2,572,812 preprocessed SMILES strings. The training sequence executed epochs until optimal parameters minimized the loss function, with weights saved to checkpoint files for subsequent use. For Model 1, this sequence loaded the configuration settings, initialized the model architecture, and fed the training and testing datasets into the trainer, halting after 26 epochs when the validation loss stabilized. This process utilized an NVIDIA GeForce GTX 1070 Ti GPU, reducing training time compared to an Intel Core i9-9900K CPU, with each model requiring approximately 72 hours to complete. Initially planned for 42 epochs, training stopped early at 26 epochs to avoid underfitting or overfitting, as the validation loss flattened and began to rise thereafter, indicating solid performance on generating realistic molecules. The training sequence for Model 1 is shown in Figure 4.5.

```
from scripts import DataLoader1, ModelBuilder1, ModelTainer1

config = settings('Model1_Settings.json')
modeler = ModelBuilder1(config, session='train')
train_dl = DataLoader1(config, data_type='train')
test_dl = copy(train_dl)
test_dl.data_type = 'test'

trainer = ModelTainer1(modeler, train_dl, test_dl)
trainer.train()
trainer.model.save_weights(checkpoints/model1-baseline.hdf5')
```

FIG. 4.5. *Training sequence for Model 1 in Python 3.7.11, integrating builder, loader, and trainer scripts.*

5. Molecule Generation. After initial training, each model generated 4,000 SMILES sequences to evaluate baseline performance. Three metrics were defined for this assessment: validity (percentage of chemically valid SMILES), originality (percentage of valid SMILES absent from the training dataset), and uniqueness (percentage of non-duplicate valid SMILES). Results of this evaluation were presented in Table 5.1, highlighting Model 3's superior performance across all metrics. Subsequently, fine-tuning was performed to optimize the models against the SARS-CoV-2 variants (Alpha, Beta, Gamma, Delta), utilizing their protein sequence data sourced from public repositories. This process employed PyRx, a virtual screening tool, to calculate binding affinities for generated molecules [2], with each evaluation of 1,000 randomly selected SMILES per model requiring approximately 25 seconds per variant.

Metric	Model 1	Model 2	Model 3
Valid	88.9%	94.4%	98.0%
Original	85.5%	92.1%	94.1%
Unique	90.2%	94.6%	97.9%

TABLE 5.1

Evaluation of 4,000 SMILES generated by each model post-initial training, showing percentages for validity, originality, and uniqueness.

Fine-tuning proceeded over 25 generations, during which 3,000 SMILES (1,000 per model) were generated per iteration and assessed for binding affinity. The top 50% of SMILES with the highest average binding affinity across the variants were selected, filtered for diversity using similarity measures, and narrowed to the 250 best candidates. These were then used as input for retraining in the subsequent generation. This iterative process, automated via Python 3.7.11 scripts, ceased after 25 generations when performance improvements plateaued, balancing computational efficiency with molecular optimization. The resulting fine-tuned models produced SMILES sequences with enhanced potential for SARS-CoV-2 inhibition, as analyzed in the following section.

6. Results and Analysis. Training over 26 epochs yielded distinct performance outcomes across the three LSTM-based RNN models, reflecting their structural differences in dropout configuration. Model 1, with dropout rates of 0.6 and 0.8, exhibited the highest loss function values, ranging from approximately 1.2 to 1.3 for training and around 1.5 for validation. Model 2, with dropout rates of 0.4 and 0.6, achieved lower loss values, between 1.05 and 0.8 for both training and validation. Model 3, with the lowest dropout rates of 0.2 and 0.4, demonstrated the best performance, with loss values approximately three times lower than Model 1, as illustrated in Figure 6.1. This figure plots the number of epochs against the averaged total loss function for training and validation across all three models, highlighting Model 3's superior convergence.

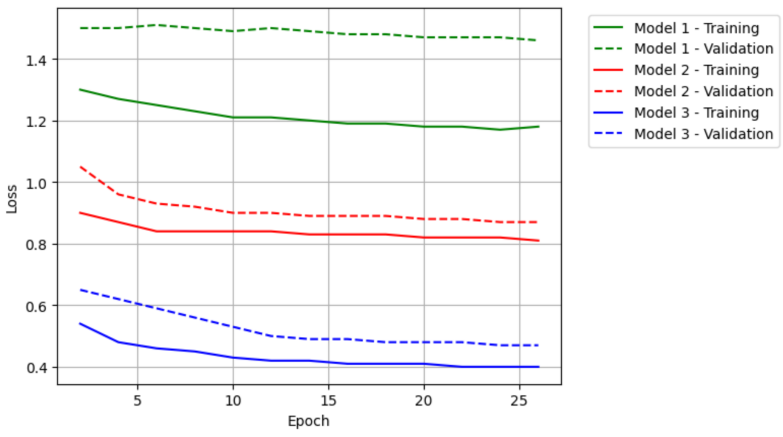

FIG. 6.1. *Number of epochs versus averaged total loss function for training and validation across Models 1, 2, and 3.*

Post-training evaluation of the 4,000 generated SMILES per model, detailed in Table 5.1, showed satisfactory validity, originality, and uniqueness across all models. However, Model 3 outperforms the others, achieving 98.0% validity, 94.1% originality, and 97.9% uniqueness, compared to Model 1's 88.9%, 85.5%, and 90.2%, respectively. The progression of the validity metric over the 25 fine-tuning generations is depicted in Figure 6.2, where Model 3's curve resembles a logarithmic function, suggesting rapid learning in the first 15 epochs, while Models 1 and 2 exhibit flatter slopes.

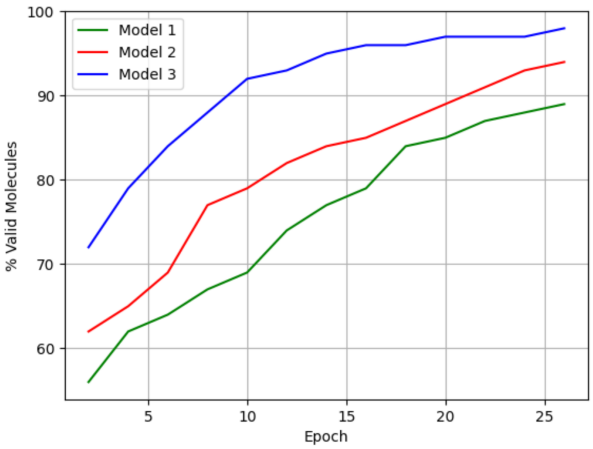

FIG. 6.2. *Percentage of valid SMILES generated over 25 fine-tuning generations for Models 1, 2, and 3.*

After 25 generations of fine-tuning, the models generated their best SMILES sequences for inhibiting the SARS-CoV-2 variants. Negative binding affinity values, as computed by PyRx, represent stronger binding strength than positive values, with greater negativity indicating enhanced inhibition efficiency [2]. All models produced valid and potentially effective SMILES, with performance differences attributed to their dropout configurations, as outlined in Section 3 and Figure 3.3. Model 1, with the highest dropout rates (0.6 and 0.8), excluded substantial input and recurrent connections during training, limiting its ability to retain critical learned data for weight and hyperparameter adjustments, resulting in the weakest performance. Model 2, with moderate dropout rates (0.4 and 0.6), performed better, while Model 3, with the lowest rates (0.2 and 0.4), minimizes overfitting and maximizes retention of learned information, achieving the best outcomes. Notably, excessively low dropout values can also degrade performance, similar to overly high values, suggesting an optimal balance in Model 3's configuration. These processes—training, fine-tuning, and SMILES generation—were conducted under identical computational conditions, reinforcing dropout as the primary factor in performance variation.

The top five SMILES sequences, based on average binding affinity across the SARS-CoV-2 variants, are presented in Table 6.1. This table includes four SMILES from Model 3 and one from Model 2, with Model 3's top performer achieving an average binding affinity of -17.40 kcal/mol.

SMILES	Model	Gen.	Weight (Da)	Alpha (kcal/mol)	Beta (kcal/mol)	Gamma (kcal/mol)	Delta (kcal/mol)	Avg. (kcal/mol)
CC(C)CN(CC(O)C(Cc1ccccc1)NC(=O)C(Cc1ccccc1)NC(=O)OC1COC2OCCC12)S(=O)(=O)c1ccc(NCc2ccccc2)cc1	3	25	619	-17.94	-19.74	-16.96	-14.94	-17.40
O=C(O)C(Cc1ccccc1)NC(=O)C(Cc1ccccc1)NC(=O)C(Cc1ccccc1)NC(=O)C(Cc1ccccc1)NC(=O)C(Cc1ccccc1)NC(=O)C1CCCN1C(=O)NCC1CCCC1	3	25	801	-17.59	-19.69	-16.65	-14.83	-17.19
NC(=O)C(Cc1ccccc1)NC(=O)C(Cc1ccccc1)NC(=O)C(Cc1ccccc1)NC(=O)OC12CC3CC(CC(C3)C1)C2	3	24	667	-17.57	-19.64	-16.37	-14.80	-17.10
CC(C)C(NS(=O)(=O)c1ccccc1)C(=O)NC(Cc1ccccc1)C(=O)NC(Cc1ccccc1)C(=O)NC(Cc1ccccc1)C(=O)NC(Cc1ccccc1)C(=O)O	3	25	855	-16.82	-19.56	-16.33	-14.77	-16.87
CC(C)(C)NC(=O)C(Cc1ccccc1)NC(=O)C1CC(CC2CCCCC2)CC2CCCCC2C(=O)NC(Cc2ccccc2)C(=O)NC(Cc2ccccc2)C(=O)NC(Cc2ccccc2)C(=O)N1	2	23	780	-16.47	-19.32	-16.21	-14.64	-16.66

TABLE 6.1

Top five SMILES sequences post-fine-tuning, showing model, generation, molecular weight (Daltons), and binding affinities (kcal/mol) for SARS-CoV-2 variants.

7. Conclusion. This study applied three LSTM-based recurrent neural network (RNN) models to investigate their utility in drug design and discovery. The sequence generation capability of LSTM nodes, recognized as highly effective among RNN architectures [6], enabled the models to produce novel pharmaceutical compounds. Results demonstrate their applicability in generating molecules that opti-

mize laboratory processes, with fine-tuning tailored to the SARS-CoV-2 variants (Alpha, Beta, Gamma, Delta) as reference points. These variants, identified during the COVID-19 pandemic, pose significant challenges due to their enhanced transmissibility and immune evasion. Model 3 exhibited superior performance compared to Models 1 and 2, generating the top SMILES sequence with an average binding affinity of -17.40 kcal/mol, reflecting strong inhibitory potential against the variants.

The methodology and models showcased the advantages of LSTM-based RNNs, producing a set of molecules capable of binding and potentially inhibiting SARS-CoV-2 variants. Unlike traditional approaches, this technique required minimal manual intervention, offering a fully customizable and automated process supervised by the user, who could adjust or halt steps as needed. The rapid generation of molecules, compared to conventional methods, enhances its appeal for in-situ chemical simulations and modeling. Additionally, these models relied on fewer parameters than alternative non-RNN approaches while achieving high percentages of valid and potent molecules [7]. Automatic checkpoints during training and fine-tuning further mitigated tendencies toward underfitting or overfitting, ensuring robust performance.

These findings, however, remain computational. The next step involves validating the most promising SMILES sequences in a laboratory setting, engaging molecular chemistry experts to assess their real-world efficacy. Further exploration could extend the models' applicability to additional molecular targets or enhance their design through alternative computational strategies. Overall, LSTM-based RNNs for generating pharmaceutical drugs via protein binding exhibit clear potential to reduce the time and cost of drug discovery, offering a scalable solution to accelerate therapeutic development during disease outbreaks and improve global health outcomes.

REFERENCES

[1] F. Campbell, et al., *Increased transmissibility and global spread of SARS-CoV-2 variants of concern as at June 2021*, Eurosurveillance, European Centre for Disease Prevention and Control, 2021, https://doi.org/10.2807/1560-7917.es.2021.26.24.2100509.

[2] S. Dallakyan and A. J. Olson, *Small-molecule library screening by docking with PyRx*, in *Chemical Biology. Methods in Molecular Biology*, vol. 1263, Humana Press, 2014, https://link.springer.com/protocol/10.1007/978-1-4939-2269-7_19.

[3] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, 2014, https://arxiv.org/abs/1412.6980.

[4] D. Mendez, et al., *ChEMBL: Towards direct deposition of bioassay data*, Nucleic Acids Research, Oxford Academic, 47 (2018), pp. D930–D940, https://academic.oup.com/nar/article/47/D1/D930/5162468.

[5] D. Polykovskiy, et al., *Molecular Sets (MOSES): A benchmarking platform for molecular generation models*, Technology and Code, Frontiers, 2020, https://doi.org/10.3389/fphar.2020.565644.

[6] A. Sherstinsky, *Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network*, ArXiv, Cornell University, 2021, https://arxiv.org/abs/1808.03314.

[7] G. Schneider and U. Fechner, *Computer-based de novo design of drug-like molecules*, Nature Reviews. Drug Discovery, U.S. National Library of Medicine, 4 (2005), pp. 649–663, https://pubmed.ncbi.nlm.nih.gov/16056391/.