Navigating the Molecular Maze: A Python-Powered Approach to Virtual Drug Screening

John Raicu[1], Valerie Hayot-Sasson[2], Kyle Chard[2], and Ian Foster[2]

[1]Glenbrook South High School
[2]University of Chicago

Abstract

The COVID-19 pandemic has highlighted the power of using computational methods for virtual drug screening. However, the molecular search space is enormous and protein docking methods are still computationally intractable without access to the world's largest supercomputers. Instead, researchers are using AI methods to help guide docking campaigns. In such approaches, a lightweight surrogate model is trained and then used to identify promising candidates for screening. We present ParslDock, a Python-based pipeline using the Parsl parallel programming library and the K-Nearest Neighbors machine learning model to screen a huge molecular space of molecules against arbitrary receptors. We achieved a 38X speedup with ParslDock compared to a brute-force docking approach.

1 Introduction

Protein docking is a key computational method used in molecular biology to predict the structure of protein complexes formed when two or more proteins interact. This technique is vital in understanding biological processes and designing therapeutic drugs. Essentially, protein docking involves simulating the process by which proteins fit together or 'dock' to form a stable complex. This is akin to finding the correct way two puzzle pieces fit together among myriad possibilities.

Docking scoring functions estimate the binding free energy or affinity between a ligand and a protein. The lower the score, the more energetically favorable the binding interaction. The docking process is driven by several factors, including shape complementarity, electrostatic attractions, and hydrophobic interactions. As proteins are highly flexible and complex molecules, the problem of predicting their interactions is computationally demanding.

A typical docking computation can take over 10 minutes on 1-core; a typical workload involves multiple protein receptors (e.g., we found studies with 15

receptors) with millions of possible ligands to dock to (e.g., we found studies with 13M ligands), yielding a total compute complexity for a brute force approach to be over 32M CPU-hours (3710 years in a serial computation on a single core). [Clyde et al.(2021)]

2 Problem Statement

ParslDock aims to identify optimal ligands from a large dataset of potential molecules by efficiently combining simulation (docking) and ML algorithms on HPC resources. The goal is to reduce the computational complexity of the brute force docking application through ML methods. Our simulation runs the Docking application which aims to predict the optimal binding conformation of a protein receptor and ligand using a binding affinity scoring function. The main challenge is to identify a ML model that is sufficiently accurate to enable the identification of the best ligands from a relatively small subset of ligand dockings.

3 Proposed Solution

We propose ParslDock, an ML-based system that enables scalable and flexible screening of molecules. We assume that users provide a target receptor (in PDBQT format) and a list of candidate molecules (as SMILES strings). We aim for ParslDock to efficiently use allocated resources (e.g., from parallel or distributed systems) combining ML models and Monte Carlo Simulations.

3.1 Software

We used AutoDock Vina (a leading docking implementation) that performs docking and utilizes a scoring function and gradient-based optimization algorithm. [Eberhardt et al.(2021)]

We used the Parsl parallel programming library to execute parts of the application in parallel. The power of Parsl lies in its ability to manage and coordinate the execution of tasks across a range of computing resources, from multicore workstations to HPC systems. Parsl's programming model relies on defining apps (to represent functions that can execute concurrently). Apps return 'futures', which are essentially placeholders for results that are computed asynchronously. Parsl automates the management of these futures, effectively handling task dependencies, synchronization, and data movement. Parsl allows users to write code that is agnostic of the underlying computing infrastructure, hence making scripts portable and scalable.[Babuji et al.(2019)]

3.2 Machine Learning Model

We used a K-Nearest Neighbor (KNN) model. [Cover and Hart(1967)] Each molecule (a SMILES string) is converted into a Morgan Fingerprint that is

represented as a bit-vector.

The SMILES "CCN(CCCC(C)NC1=C2C=CC (=CC2=NC=C1)Cl)CCO" represents the Hydroxychloroquine molecule. This SMILES string is converted into the bit-vector "111001001111010111
1100111101101101111111100111111110000100110101". We computed the euclidean distance matrix in the KNN as an all-to-all distance matrix between every molecule to every other molecule's bit-vector representation. Our ultimate goal is to find a good mapping between fingerprints and docking score.

‘

3.3 ParslDock Pipeline

The ParslDock pipeline (Figure 1) is composed of 7 stages:

1. Dataset 4M: Input data is a target receptor (PDBQT) and a random list of 100K molecules (SMILES)
2. Data Format: Data preparation for AutoDock Vina docking; SMILES string to PDB file to PDBQT file
3. Docking: Executes Monte Carlo simulations between a protein receptor PDBQT file and a ligand PDBQT file, yielding a binding-affinity score output
4. Morgan Fingerprints: Generated 128-bit vector with a depth of 8 from a SMILES string [Pattanaik and Coley(2020)]
5. Machine Learning: Morgan Fingerprints and docking scores are paired as the input to KNN model to identify correlations between fingerprints and docking scores
6. Dataset Top-4K: The top 4K ligands identified based on the predicted docking scores (lowest binding-affinity scores) of all 4M molecules
7. Docking: Runs Monte Carlo simulations on the Top-4K subset of data to obtain a final list of top molecules ordered by docking scores

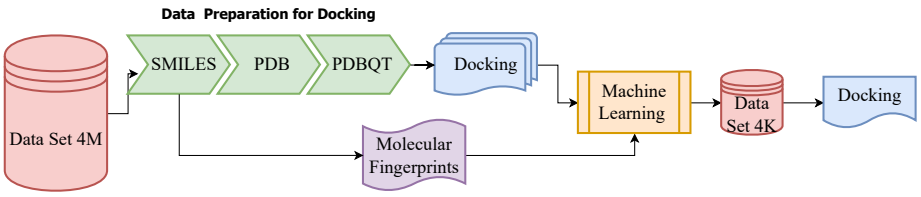


Figure 1: ParslDock pipeline

Most of these steps can be run concurrently across a set of input molecules. We implement each step as a Parsl app and establish dependencies between each

step. This allows us to scale out execution of each step across input data. All our code and datasets can be found on Github. [Raicu(2023)]

4 Software Stack, Testbeds, and Dataset

4.1 Software Stack

The Programming Tools we used involved Python 3.8.3 to implement the computational pipeline. We used Parsl 1.3.0.dev0 to parallelize various stages of the computational pipeline. We made extensive use of Jupyter Notebook 6.5.4 in the development of ParslDock.

The main docking library we used was AutoDock Vina 1.2.3 which utilizes a scoring function and gradient-based optimization algorithm. We used the Visual Molecular Dynamics 1.9.3 application to visualize and analyze molecular simulations. Finally, we used Py3Dmol 2.0.3 to enable interactive 3D molecular visualizations. The machine learning library that implemented the KNN was Scikit-learn; we made use of other popular scientific and data analysis libraries in Python, such as NumPy, Pandas, and Matplotlib.

4.2 Testbeds

The Hardware we use includes two testbeds to evaluate ParslDock. We conduct experiments on two testbeds: 8c-laptop and 192c-server. The laptop was a MacBook Pro from 2019 with an 8-core Intel Core i9 CPU, 2.4GHz, 64GB DDR4, 8TB NVMe, running MacOS 12.6.3. The server had 8x 24-core Intel Xeon Sky Lake CPUs (for a total of 192-cores), 2.1GHz, 786GB DDR4, 16TB SSD, running Ubuntu Linux 22.04.

4.3 Datset

The 0.9 GB Figshare Dataset we used contained 8 million ligands stored as SMILES strings.

5 Evaluation

5.1 Docking Scores

We used 4 million ligands from the 8 million ligand dataset and performed docking against the 1IEP receptor protein. This resulted in a new dataset that included 4M scores that are represented in the Figure 2. The scores follow a normal distribution and range from -187.3 to 46.65, with a median of -8.654. Our goal was to identify the top 0.1% of the ligands based on the docking score (green line at -34.32). All the ligand docking scores less than or equal to -34.32 were labeled as “best” while all docking scores higher than -34.32 were labeled as “worst”.

5.2 KNN Model

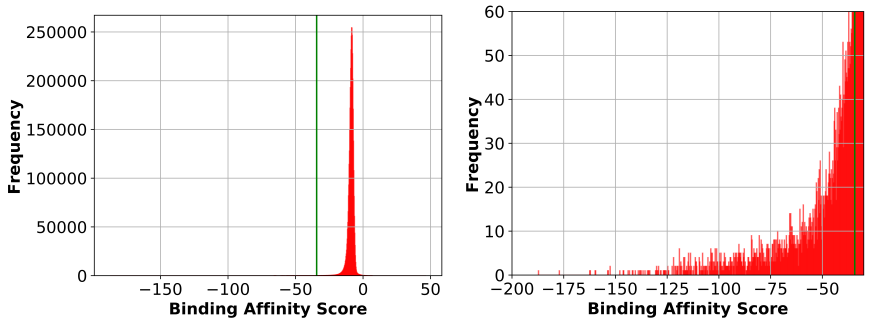
Once we obtained the docking scores for the entire dataset, we selected 100K random ligands and used the docking scores to build a KNN model as seen in Figure 3. We first explored use of the entire 100K random samples, which yielded surprisingly high accuracy results in the 99%+ range. Upon close inspection, we found that our dataset had a minority class (100 samples) and a majority class (99,900 samples). KNN are well known to perform poorly for unbalanced datasets. We set out to balance it by pulling a random sample of size 100 from the majority class of 99,900 samples.

Our final dataset we used to build the KNN was only 200 samples large, representing two classes that were of equal size. We tried larger sample sizes, but we did not see a significantly improved accuracy on the KNN testing data. We found that a relatively small number of molecules are needed to build the KNN while achieving good accuracy, so the $O(n^2 * d)$ time and space complexity of KNN is tractable for modest values of n and d.

We balanced the dataset by choosing 100 of each. Larger samples did not significantly improve accuracy. We explored different values of dataset sizes (from 10K to 10M samples), and determined that 100K offers good enough accuracy of 87%. The number of neighbors we settled on for the KNN was 5. Different values of K did not yield significantly and consistently better results.

5.3 Molecular Fingerprints

We conducted a hyperparameter search of the "depth" (radius around atoms) and "size" (number of bits) of fingerprint. Figure 4 shows the accuracy of the KNN model on a training/testing dataset partitioned in 70% training and 30% testing. The best configuration (yellow) is when size is 128 and depth is 8, achieving an accuracy of 87%. Note that sub-optimal parameters for the Morgan Fingerprint can easily yield under 50% accuracy. One critical variable in building the machine learning model was the input data: a Morgan Fingerprint



(a) All scores

(b) Zoomed view

Figure 2: Binding affinity score for docking of 4M molecules.

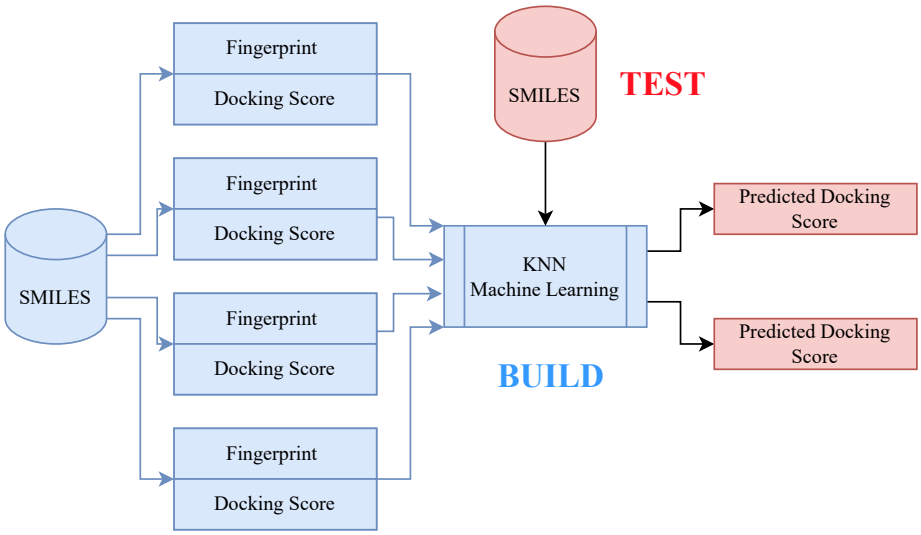


Figure 3: Machine Learning Workflow highlighting inputs and outputs

and a docking score.

The Morgan Fingerprint itself has two different parameters that significantly changes the fingerprint contents: depth and length. "Depth" typically refers to the radius parameter used in generating the fingerprints. Morgan fingerprints are circular fingerprints that capture local chemical environments around atoms in a molecule. The depth of the fingerprint determines how far from each atom the circular neighborhoods extend. By adjusting the depth parameter, you control the level of detail captured in the fingerprint. A larger radius captures more extended structural patterns and interactions, while a smaller radius focuses on local substructures.

The second parameter is "length" referring to the number of bits or dimensions used to represent the fingerprint. Each bit in the fingerprint corresponds to a specific substructure pattern or feature. The length of the fingerprint determines the level of granularity at which the molecular structure is encoded. In summary, the length of a Morgan fingerprint determines the dimensionality of the binary representation used to capture the molecular structure's features, and it plays a role in balancing representation power and computational considerations.

In order to identify the best fingerprint depth and size, we conducted a hyperparameter search of different fingerprint length and depth and measured the accuracy of the KNN model on a training/testing dataset partitioned in 70% training and 30% testing. The figure below shows the bright yellow square where fingerprint size is 128 and depth is 8, achieving an accuracy of 87%. Note that sub-optimal parameters for the Morgan Fingerprint can easily yield under 50% accuracy.

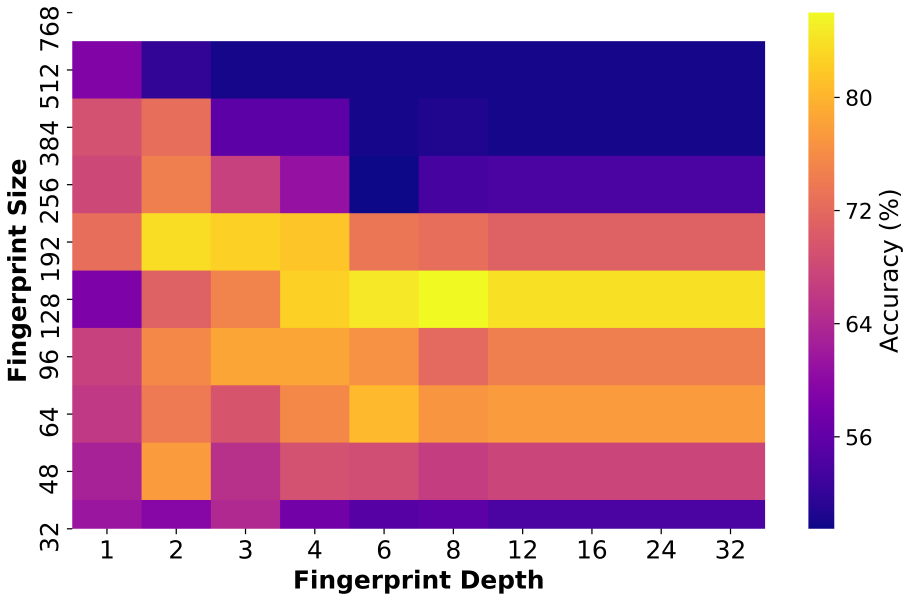


Figure 4: 2D-Heatmap showing KNN accuracy for fingerprint size vs depth

5.4 Performance

5.4.1 ParslDock vs Brute Force Docking

ParslDock is a Python-powered automated pipeline that uses machine learning to accelerate the docking process. From the start (4 million molecules represented as SMILES strings) to the end (top-4k ligands with docking scores), we evaluated the time taken to run the entire ParslDock pipeline on two systems: an 8-core laptop and a 192-core server in Mystic (see Figure 5). We measured the various stages of the pipeline, from all the data transformations to PDB and PDBQT files, docking, machine learning building and testing, and final ligand selection of the best ligands identified. Tasks ranged from less than 1 millisecond (e.g. Morgan fingerprint generation), with many format conversions taking about 300 milliseconds, to molecular docking that took on average 20 seconds to complete. I used Parsl to parallelize execution across the multicore systems in our testbed to ensure that the thousands to millions of tasks could be executed in parallel, weather it was on my 8-core laptop, or on the 192-core server. A computation that if done naïve, would have taken nearly 5 years to execute by brute force on a personal laptop without parallelism, we were able to complete the same workload on a large multi-core system with 192-cores in just 2 days.

Protein docking is a complex process that can be computationally expensive and time-consuming. Even if a personal laptop was the only computing resource one had access to, through the ParslDock pipeline that uses parallelism and machine learning to prune the search space, a workload that originally would

have taken 5-year to complete could now be done in in 47 days on the same hardware, or as little as 2 days on a 192-core server. The speedup obtained on ParslDock compared to the brute force docking of the entire 4M ligands was 38X on the laptop and 37X on the server. Our performance evaluation showed linear scalability from an 8-core laptop to a 192-core server.

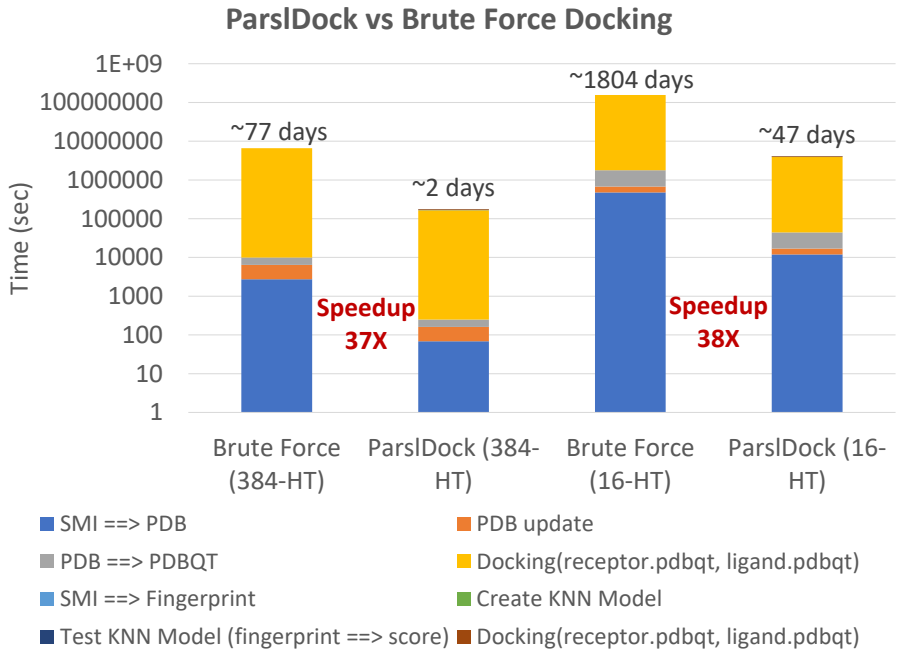


Figure 5: ParslDock and Brute Force Docking end-to-end pipeline execution time

5.4.2 Parsl Optimizations

Key to this work is the workflow graph generated by ParslDock. Each task consists of a molecule that is formatted and passed to a machine learning module for inference. This generates a bag-of-tasks graph, such a task graph is highly parallelizable. We use this scientific application to demonstrate the improved performance achieved by our optimizations.

ParslDock executed on the Mystic testbed mentioned previously. The docking simulation displayed Figure 6, Figure 7, and Figure 8 display runtime with varying numbers of workers and batch size. The simulation was run across 100k total molecules For serial, standard Parsl, and CDFK, the best runtimes are achieved with the largest batch size. This suggests that there is still work to be done to improve Parsl for fine-grained tasks.

Figure 6 shows the runtime of the docking simulation as a function of the

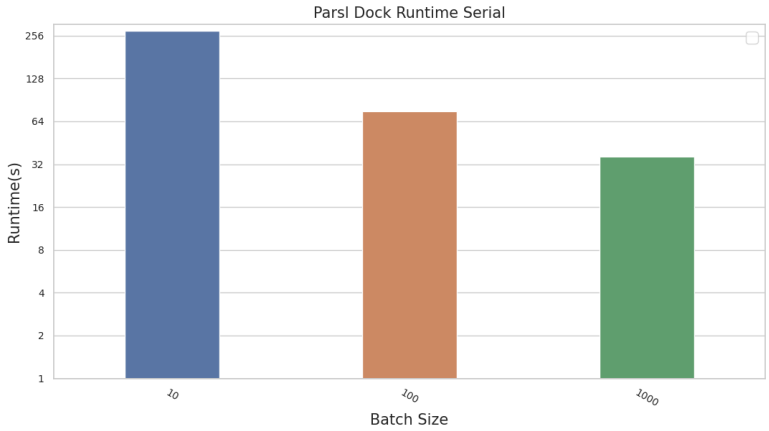


Figure 6: ParslDock runtime Serial

batch size using serial code. Using a batch size of 1000 molecules the serial code achieves a runtime of around 35 seconds.

Figure 7 displays the runtime achieved by standard Parsl. Standard Parsl achieves its lowest runtime using 8 workers with a batch size of 1000, 11s. With standard Parsl using many workers or few workers returns a high runtime, 100 seconds.

Figure 8 shows the runtime achieved by CDFK Parsl. It scales linearly with the number of workers used, plateauing with hundreds of workers. CDFK Parsl with 128 workers and a batch size of 1000 simulates 100k molecules in 1.5 seconds. This runtime is 10x faster than standard Parsl's lowest runtime and 30x faster than serial.

6 Future Work

We used a K-Nearest Neighbor (KNN) model. Each molecule (a SMILES string) is converted into a Morgan Fingerprint that is represented as a bit-vector. The SMILES "CCN(CCCC(C)NC1=C2C=CC (=CC2=NC=C1)Cl)CCO" represents the Hydroxychloroquine molecule. This SMILES string is converted into the bit-vector "11100100111101011111001111011011011111111001111111110000100110101". We computed the euclidean distance matrix in the KNN as an all-to-all distance matrix between every molecule to every other molecule's bit-vector representation. Our ultimate goal is to find a good mapping between fingerprints and docking score.

We used the default distance metric, Euclidean Distance. Euclidean distance is a general purpose distance metric that works for a variety of input datasets. The Euclidean distance is commonly used for numerical data where the magnitude and continuous values of attributes matter. It calculates the straight-line

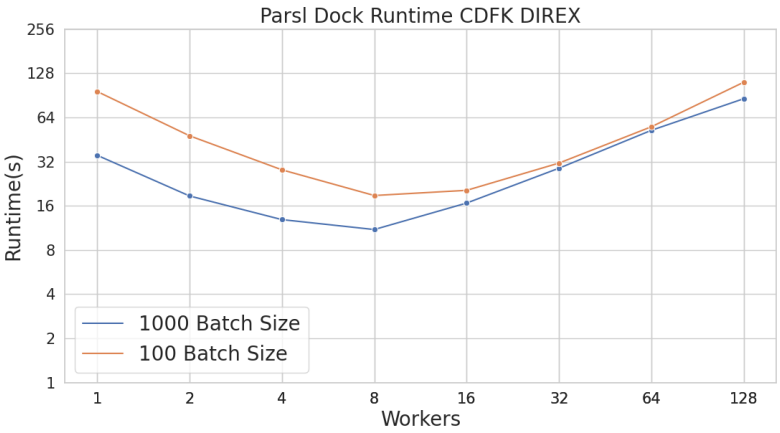


Figure 7: Standard Parsl

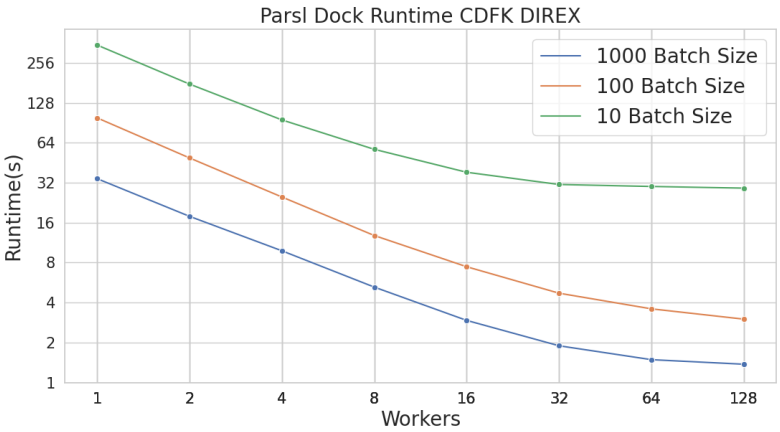


Figure 8: CDFK

distance between data points in a multidimensional space. Given that our input dataset is a bit-vector (Morgan fingerprint), we could explore other distance metrics such as Jaccard Coefficient, Tanimoto, or Hamming Distance. These similarity metrics are typically used for sets or binary data, where each data point can be thought of as a binary vector (0 or 1). Tanimoto in particular is useful when dealing with sparse binary data, where the presence or absence of features is more relevant than their values. I believe I can improve the machine learning model's accuracy by exploring different distance similarities as outlined here.

Finally, KNN is a relatively simple machine learning model that is inherently a non-parametric algorithm that assumes a constant decision boundary in the feature space. Furthermore, KNN can be computationally expensive, especially when dealing with large datasets, as it requires calculating distances between the query point and all data points. Deep neural networks (DNNs) is another machine learning algorithm that can automatically learn hierarchical representations of data, discovering complex patterns and relationships in the input features. The literature says that DNNs are typically more effective in handling high-dimensional data or data with intricate features. Furthermore, DNNs are capable of modeling non-linear relationships in data. DNN can be more computationally efficient, especially with if we take advantage of parallel processing on GPUs to speed up computations. In some cases, KNN may still be a suitable choice, especially for smaller datasets or problems where interpretability is crucial. However, DNNs could lead to a better predictive model for tasks that require feature learning, non-linear modeling, and scalability.

7 Conclusions

Protein docking is a complex process that can be computationally expensive and time-consuming. ParslDock is a Python-powered automated pipeline that uses ML to accelerate the docking process and reduce compute costs by up to 38X. Our performance evaluation showed linear scalability from an 8-core laptop to a 192-core server. With further improvements, we believe we can bring down the computational requirements to the point that ParslDock will be tractable on a modern day personal computer, making virtual drug screening accessible to all scientists around the world. Making this application more accessible will no doubt make our world a safer place the next time we are faced with a global pandemic, we will have millions of scientists ready to identify the best drug candidate that will stop the pandemic in months, not years.

References

[Babuji et al.(2019)] Yadu Babuji, Anna Woodard, Zhuozhao Li, Daniel S. Katz, Ben Clifford, Rohan Kumar, Lukasz Lacinski, Ryan Chard, Justin M. Wozniak, Ian Foster, Michael Wilde, and Kyle Chard. 2019. Parsl: Per-

vasive Parallel Programming in Python. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing* (Phoenix, AZ, USA) *(HPDC '19)*. Association for Computing Machinery, New York, NY, USA, 25–36. https://doi.org/10.1145/3307681.3325400

[Clyde et al.(2021)] Austin Clyde, Thomas Brettin, Alexander Partin, Hyunseung Yoo, Yadu Babuji, Ben Blaiszik, Andre Merzky, Matteo Turilli, Shantenu Jha, Arvind Ramanathan, et al. 2021. Protein-ligand docking surrogate models: A sars-cov-2 benchmark for deep learning accelerated virtual screening. *arXiv preprint arXiv:2106.07036* (2021).

[Cover and Hart(1967)] T. Cover and P. Hart. 1967. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory* 13, 1 (1967), 21–27. https://doi.org/10.1109/TIT.1967.1053964

[Eberhardt et al.(2021)] Jerome Eberhardt, Diogo Santos-Martins, Andreas F Tillack, and Stefano Forli. 2021. AutoDock Vina 1.2. 0: New docking methods, expanded force field, and python bindings. *Journal of chemical information and modeling* 61, 8 (2021), 3891–3898.

[Pattanaik and Coley(2020)] Lagnajit Pattanaik and Connor W Coley. 2020. Molecular representation: going long on fingerprints. *Chem* 6, 6 (2020), 1204–1207.

[Raicu(2023)] Johnny Raicu. 2023. ParslDock Repository. https://github.com/johnny-raicu/ParslDock.