JOURNAL OF
EMERGING INVESTIGATORS

Article

Hybrid Quantum-Classical Generative Adversarial Network for synthesizing chemically feasible molecules

Diptanshu Sikdar[1,6]*, Max Cui[2,6]*, Adelina Chau[3,6]*, Arjun Bhamra[4,6], Sathvik Prasanna[5,6], Larry McMahan[6]

[1]BASIS Independent Silicon Valley, San Jose, CA
[2]Sir Winston Churchill Secondary School, Vancouver, British Columbia
[3]Archbishop Mitty High School, San Jose, CA
[4]The Episcopal Academy, Newton Square, PA
[5]Monta Vista High School, Cupertino, CA
[6]Department of Computer Science and Engineering, Aspiring Scholars Directed Research Program, Fremont, CA
*These authors equally contributed to the research.

SUMMARY

Current drug discovery processes can cost billions of dollars and usually take five to ten years. People have been researching and implementing various computational approaches to search for molecules and compounds from the chemical space, which can be on the order of 10^{60} molecules. One solution involves deep generative models, which are artificial intelligence models that learn from nonlinear data by modeling the probability distribution of chemical structures and creating similar data points from the trends it identifies. These generative models can extract salient features that characterize the molecules. However, they often suffer from increased time complexity. Aiming for faster runtime and greater robustness when analyzing high-dimensional data, we designed and implemented a Hybrid Quantum-Classical Generative Adversarial Network (QGAN) to synthesize molecules. There are two parts in the QGAN: a quantum generator that creates molecules based on the probability distributions of likely combinations and a classical discriminator that differentiates real molecules from the generated ones. We hypothesized that a quantum generator would be more impactful, offering a runtime speedup and increasing generated molecule possibilities, because we could use quantum mechanical phenomena superposition, entanglement, and interference to analyze more atom-bond combinations than a classical generator. The PyTorch-PennyLane implementation of the QGAN generated seven chemically stable molecules out of 300, a 2.3% success rate. Although the QGAN is a work in progress, it has demonstrated a path towards more efficient drug development, which would accelerate the development of medicines and reduce costs for the whole R&D process.

INTRODUCTION

Drug discovery is the process of identifying potential chemical entities to serve as therapeutic agents (1). The general drug discovery pipeline for an Unknown Human Disease (UHD) consists of several steps: finding a target, generating a hit, drug development, safety and efficacy trials, and treatment approval (**Figure 1**) (2). Today, the drugs approved by the United States Food and Drug Administration only make up 13.8% of the drugs that were originally tested in clinical trials for diseases and medical conditions. Also, according to a recent study by Harvard University, developing one FDA-approved drug costs about 2.6 billion USD and takes about 12 years (3-4). Thus, the drug development process is extremely costly and time-consuming, raising the need for a more efficient approach.

One reason for the high cost of drug development is the progressive difficulty in creating treatments for more challenging diseases. Typical methods of creating and testing candidate treatments in the lab are difficult due to the quantity and complexity of possible candidates. In addition, the medical research and development process is extremely risky, so pharmaceutical companies incur high R&D costs (5). To alleviate such expenses, computational methods can be used to speed up the process of drug discovery. Companies can build complex molecules and execute chemical reactions using chemistry simulations. Depending on the type of simulator used, one can set certain specifications after considering functional groups, molecular symmetries, polarity, conductivity, and atomic charges. Other simulations allow scientists to determine electronic structures, make geometry optimizations, calculate electron and charge distributions, etc. Computational chemistry methods enable scientists to experiment with a wide variety of combinations of materials that are difficult to obtain and gives them more insight into molecular behavior without doing a physical laboratory experiment (6).

The input to these models would be the bounded chemical space, which refers to the property space consisting of all possible molecules and chemical compounds having specific

Figure 1: Drug Discovery Pipeline. Diagram showing the complete process of drug discovery that pharmaceutical companies use when creating new drugs.

sets of construction principles and boundary conditions (7). For example, MAYGEN, a chemical structure generator, takes a molecular formula as input and generates all constitutional isomers of the formula (8).

Researchers have also started to investigate computational methods other than chemistry simulations. In 2018, Nicola De Cao and Thomas Kipf published a paper on MolGAN, an implicit generative model for small molecular graphs with nine or fewer heavy atoms (9). They developed a generative, machine learning-based approach to create chemically buildable compounds. The MolGAN was then improved upon by Tsujimoto et. al. with their L-MolGAN which generates molecular graphs with up to 20 heavy atoms by penalizing generation of disconnected graphs (10). In 2021, Andrew Blanchard used a generative adversarial network with adaptive training data for drug discovery (11). Similarly, Junde Li et. al discussed the advantages and disadvantages of a hybrid generator (utilizing both quantum and classical systems) and experimented with a quantum generative adversarial network with a hybrid generator for drug discovery (12-13).

The Generative Adversarial Network (GAN) is a machine learning algorithm with a unique internal architecture which consists of two neural networks that compete against each other. These networks are called the Generator and Discriminator, and each has a different purpose. The Generator is a network that generates fake data and attempts to make it as real as possible. On the other hand, the Discriminator is a network that distinguishes the input data as real or fake (14-15).

The Generator's weights are first initialized by random values drawn from a uniform Gaussian distribution. The network has trainable parameters which transform the input noise into the desired output, such as a molecule. This output goes into the Discriminator, which is essentially a binary classifier that distinguishes between real and fake objects. Because the Discriminator needs to know what constitutes "real", it should first see several real samples, then see some fake samples, and then keep alternating between real and fake.

After the Discriminator has classified the Generator output, the loss functions (a method that evaluates how well an algorithm models a dataset) for the Generator and Discriminator are evaluated, determining the accuracy of the object and its classification (16). Then, the losses are supplied to both networks as feedback to update their weights through a process known as backpropagation. The Discriminator is trained to maximize the probability of correctly differentiating fake data from real data. Conversely, the Generator is trained to maximize the probability of making the Discriminator fail by making the fake data as real as possible (17). This training process repeats, with the Generator and Discriminator each simultaneously trying to outsmart each other, until a balance is reached when neither network can outperform the other. Reaching this balanced state is equivalent to finding the Nash equilibrium of a non-cooperative, two-player game.

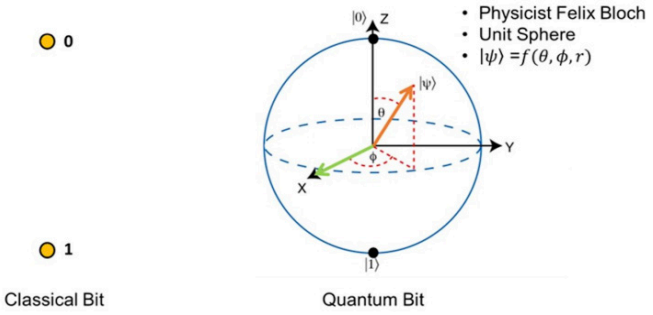

Figure 2: Classical Bit versus Quantum Bit. Self-drawn diagram that shows a visual representation of a classical bit and a quantum bit (qubit). A qubit's state is a linear combination of the basis states (0 and 1); however, a bit's state is either 0 or 1, but not both.

Currently, most GANs are implemented solely with classical computing and referred to as the classical model, although some hybrid quantum-classical variants do exist. Quantum models of the GAN are implemented using quantum computing, which utilizes mechanical concepts like superposition, interference, and entanglement to enhance its capabilities. There are several primary differences between classical computing and quantum computing. While bits in classical computers represent either a '0' or a '1', quantum bits, also known as qubits, can represent a linear combination of the '0' and '1' states, formally known as superposition (**Figure 2**). This allows qubits to hold exponentially more information than the same number of bits. As a result, a quantum computer can achieve much higher levels of parallelization than classical computers. Furthermore, the entanglement and interference properties provide quantum computers a significant advantage in terms of computational speed since they allow quantum computers to operate on many states at once.

Thus, for drug development, we preferred a quantum model over a classical model, as quantum models are able to search through a large chemical space exponentially quicker than classical models, resulting in a tremendous speedup and memory usage improvement. In addition, both drug development and quantum mechanics ultimately deal with the behavior of atoms, particles, and molecules, so it would be more efficient to simulate the performance of drugs with a quantum computer (18).

Building upon these ideas, the goal of this project was to generate candidate molecules efficiently by leveraging aforementioned quantum mechanical phenomena for efficient parallel computation. Hence, we hypothesized that a GAN with a quantum generator (hence QGAN) would be more impactful by offering a runtime speedup and increasing generated molecule possibilities because increased computing power allows the quantum generator to analyze more atom-bond combinations than a classical generator. Thus, we developed a Quantum Generative Adversarial Network, a quantum unsupervised machine learning model, that can create new, buildable molecules given a dataset of existing molecules

with similar properties. The model that we built is a proof-of-concept prototype that created 7 valid candidates out of 300 total molecules generated. In the future, the Quantum GAN may be able to accelerate drug discovery as well as other uses for chemically feasible molecule generation, saving pharmaceutical companies time and resources to make a potentially life-saving drug available.

RESULTS

We first trained the Quantum GAN (QGAN) on molecules from QM9, a dataset with 134,000 stable, small organic molecules. Next, we tested it by generating an arbitrary list of 300 molecules, and 7 of the molecules were considered chemically feasible. It is important to note that we only considered the atoms, not their x, y, z coordinates when we checked for a valid molecule because we had to sacrifice the precision of the x, y, z coordinates due to the limited number of qubits, 20, available on the quantum simulator.

Generated molecules that could be found in a large, well-known dataset provided by PubChem, such as C_4H_3 and $CHNO_4$, were deemed chemically feasible (**Figure 3**). Generated molecules that do not exist in the PubChem database, such as NH_4O_2 and N_3H_4, were not considered feasible (**Figure 4**).

To determine if the QGAN was more efficient than a classical GAN at molecule synthesis, we measured the execution times and computational resources. The QGAN was run on a quantum simulator because we did not have access to a quantum computer with more than 5 qubits. The simulation only utilized a CPU, and no GPU hardware acceleration was necessary. The total runtime for training the QGAN varied between 15 and 20 minutes for 200 epochs, or equivalently, about 4.5 to 6.0 seconds per epoch. When running the competitor, MolGAN, with the CPU, the total runtime for 2000 epochs was more than two days, or equivalently, about 86.4 seconds per epoch. Therefore, the QGAN was on average about 14 times faster.

We also ran the QGAN with GPU acceleration. It took about 20 minutes to train for 500 epochs, or equivalently, about 2.4 seconds per epoch. When MolGAN was run with GPU acceleration, the total runtime varied between 90 and 105 minutes for 2000 epochs, or equivalently, about 2.7 to 3.2 seconds per epoch. Thus, QGAN with GPU was about 1.2 times faster than MolGAN with GPU. However, it is important to note that all quantum computations were run in a simulator on a classical computer, and thus does not accurately reflect the results if we were to run QGAN on a real quantum computer.

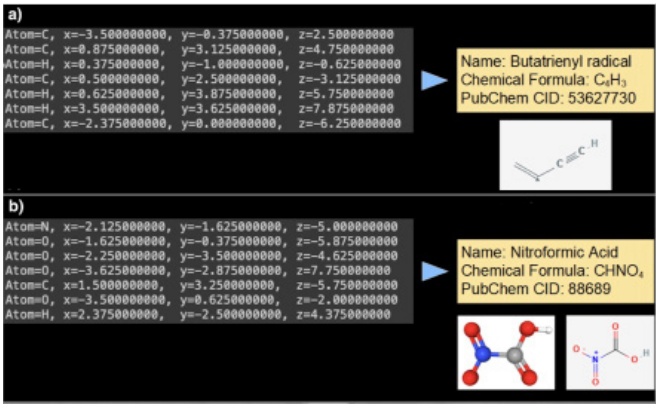

Figure 3: Chemically Feasible Molecules Generated by QGAN. Figure 3a) shows the XYZ file for C4H3 and figure 3b) shows the XYZ file for CHNO4; included are the atoms and their coordinate values. Both are chemically feasible molecules generated by the QGAN.

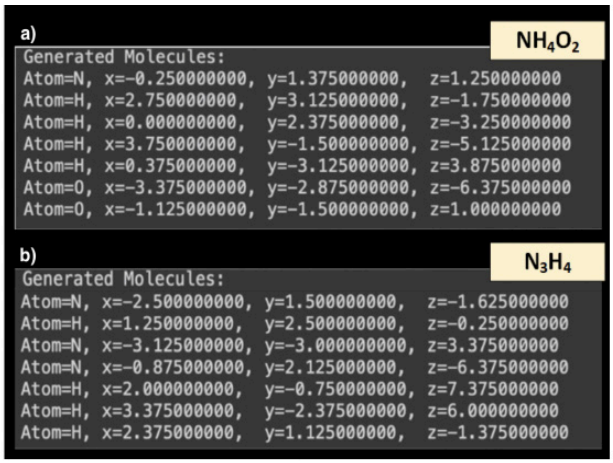

Figure 4: Chemically Unfeasible Molecules Generated by QGAN. Figure 4a) shows the XYZ file for NH4O2 and figure 4b) shows the XYZ file for N3H4; included are the atoms and their coordinate values. Both are chemically unfeasible molecules generated by the QGAN.

As for RAM, it was found that using 30 qubits in the Generator exceeded 16 GB of memory, while MolGAN used under 1 GB. Because the discriminator is a simple neural network that doesn't require many computational resources, we did not take its effect on the overall statistics into account.

DISCUSSION

In summary, the QGAN successfully generated 7 existing molecules out of 300 total molecules, achieving about 2.3% success rate. Comparing this performance to the MolGAN, which achieved 98.1% success rate, shows that the QGAN is still a major work-in-progress. However, the way by which results are recorded also has room for improvement, and it may be neglecting many molecules that the QGAN generated. In addition, the QGAN was much more computationally efficient, but not as memory efficient. However, the high RAM usage of QGAN was not a concern because the Quantum GAN was simulated on a classical computer and did not accurately represent the RAM usage of a real quantum computer.

Based on Qiskit Aer's built-in memory usage estimation function (19-20), a simulation of 32 qubits would require 68.72GB of RAM, so accounting for the exponential scaling nature of memory storage in qubits, where n qubits can store 2^n times as much information as n bits, it would still use about 17.18 GB. However, the simulator cannot clear its

memory at every epoch because we have to keep track of Generator and Discriminator loss, which results in increased memory usage as we train for more epochs. Furthermore, the simulation keeps copies of the quantum states and the mechanisms to simulate quantum operations, such as matrix multiplications, on these states is very expensive, resulting in even more memory usage. Every time we apply a gate to our qubits, we are performing a matrix multiplication between two 30x30 matrices that may contain both floating-point and complex numbers. Finally, we also store our QM9 dataset of 11,000 molecules in a gigantic 3D array containing many floating-point numbers which consumes a great deal of memory.

When using the intended QGAN algorithm and molecule verification system, generated molecules were converted from the XYZ representation into the Simplified Molecular-Input Line-Entry System (SMILES) representation — a string representation of molecules that can encode several molecule's spatial features and is often known as a 2.5-dimensional representation of molecules — and the validity of each molecule was determined by the validity of the SMILES string (21). However, because the generated XYZ files lacked precision due to the decimal interval steps of 0.125, the files could not be converted into SMILES strings. The other issue with this molecule verification method was that the novel SMILES strings may not be considered valid due to invalid SMILES string conversion and the molecule not existing on the database. To more accurately determine the validity, the molecules should be synthesized in a lab and tested for chemical stability.

Drug discovery is typically a process that takes many years but can be made more efficient with the flexibility of our QGAN. The dataset used to train our model can be varied to generate molecules with certain chemical properties such as polarity, conductivity, and the functional groups deemed necessary for the potential drug candidates targeting a disease. In doing so, our QGAN would produce the most viable drug candidates specific to the target disease, reducing the number of potential molecules to be screened and making the molecules generated by the QGAN more likely to be feasible. In addition to drug discovery, our QGAN is applicable to a variety of fields, including material science, where new molecules can be used to develop novel materials, solar energy production to develop low-cost organic solar cells, and agricultural production to discover new methods of pathogen control (22-24).

In the future, the QGAN can be improved by reducing the complexity and depth of the quantum circuit to reduce the noise during execution. The discriminator can be also implemented in the quantum domain to avoid any superfluous pre- or post-processing algorithms. In addition, the molecule verification tool can be linked with the QGAN directly during the training phase to penalize the generator in case of an invalid molecule. This would force the generator to generate valid molecules from the beginning. Furthermore, the generated molecules will be simulated and evaluated for their potential success using drug likeness analysis techniques like Lipinski's Rule of Five — which determines whether a chemical compound has the physical and chemical properties to be an orally active drug — and their molecular properties will be calculated with methods like Hartree Fock (25-28). Ultimately, while there are several avenues for improvement, the QGAN's ability to quickly and efficiently search the large chemical space for chemically feasible molecules proves it to be a practical and promising addition to drug development.

MATERIALS AND METHODS

Software platforms

All the code was written and executed using the free version of Google Colab with limited access to RAM — 13 to 16 GB depending on availability. The QGAN is programmed using Python (3.7.15). PyTorch (1.12.1+cu113) was used to implement the neural network for the discriminator, while the quantum circuits and parameter-shift optimization algorithm were implemented with PennyLane (0.26.0), an open-source software for programming quantum computers (29-30). In order to train and test the QGAN, IBM's Aer quantum simulator was used; a plug-in provided by PennyLane helped pair the Aer simulator fromIBM's quantum programming library Qiskit (0.22.0) with the QGAN's quantum circuit (31). Various other modules such as NumPy (1.21.6) and PyBEL (0.15.5) were imported to help the QGAN perform mathematical operations and check if generated molecules were realistic, respectively (32-33).

Model Training

For the research, we trained the QGAN using the QM9 dataset, which consists of 134,000 small stable, organic molecules made up of carbon, hydrogen, nitrogen, oxygen, and fluorine (34, 35). Though the original QM9 dataset is represented using the SMILES representation, we found a version of the dataset that had the molecules converted to use the XYZ file representation (36). The XYZ files were later converted into CSV files to be easier to work with.

Detailed Quantum GAN Workflow

The internal architecture of the QGAN is quite different from that of a regular GAN mainly due to the quantum portion. A hybrid quantum-classical approach with a quantum generator and a classical discriminator was implemented. There are four main components: Dataset Loader, Quantum Generator, Classical Discriminator, and Classical Post-Processor (**Figure 5**).

The QM9 molecular dataset provides the Discriminator with real examples of valid, buildable molecules, which are stored inside the Dataset Loader. Since the molecules from the QM9 dataset are represented in the XYZ file representation, the Dataset Loader acts as a helper "algorithm" and converts the XYZ files into CSV files for ease of use. The dataset provides the real molecules.

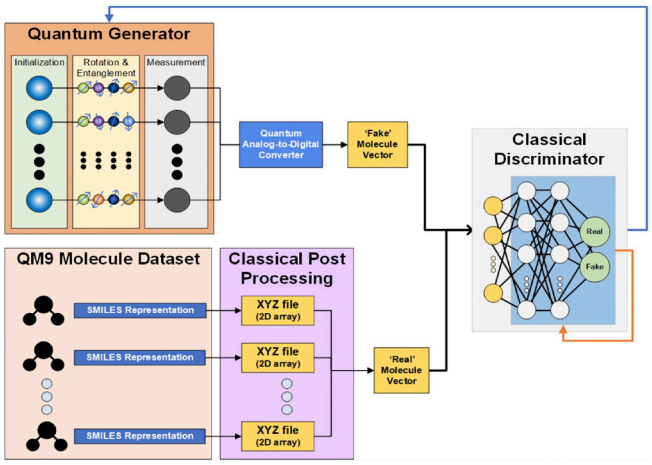

Figure 5: The Overall Architecture of the Quantum Generative Adversarial Network (QGAN). Diagram shows the architecture of each major component (Quantum Generator, Classical Discriminator, Molecule Dataset, Post-processing Algorithms) of the QGAN and how they interact with each other. The dark-orange colored box is the Quantum Generator. Random Gaussian Noise with Initialization of qubits, using quantum gates to apply superposition and entanglement on the qubits, and ending with the Pauli-Z Measurement gates. The pink and purple boxes represent the QM9 Molecule Dataset and the data processing that is done to input the molecules into the discriminator (gray box). The molecules are converted from SMILES to XYZ file (2D array) which is passed to the discriminator as a vector ('Real' molecule vector). The classical discriminator is a binary classifier that determines whether the input molecule is generated from the generator (fake) or accessed from the dataset (real). The blue arrow indicates the feedback loop between the discriminator and the generator, and the orange arrow highlights the backpropagation process of the discriminator.

In our QGAN, the generator G is a model that generates fake molecules. Thus, it does more of the "heavy-lifting" in the training process — generating molecules from scratch as opposed to simply classifying between two classes — requiring more computational resources (e.g., memory, CPU, GPU, etc.) than the discriminator. To decrease the training time of the QGAN, we leveraged quantum computation in our generator implementation. Superposition enables the model to evaluate many possible instances based on the probabilities of success; entanglement offers extreme parallelization, increasing overall efficiency.

The input to the generator is random noise from a normal Gaussian distribution (**Figure 5**). The Quantum Generator consists of a series of Parameterized Quantum Circuits (PQCs), which are quantum circuits with parameters that control how the input is transformed into the output. These parameters are the angles by which the quantum state vector rotates. Changing these parameters causes the rotation gates to transform the qubits' states from state 0 to their final state. The parameters can be adjusted via a feedback loop and an optimization function. The PQC employs superposition by performing the Walsh-Hadamard Transform and achieves linear quantum entanglement via controlled rotation gates. The repeating layers of superposition and entanglement allow the generator to be more accurate. Finally, the final quantum

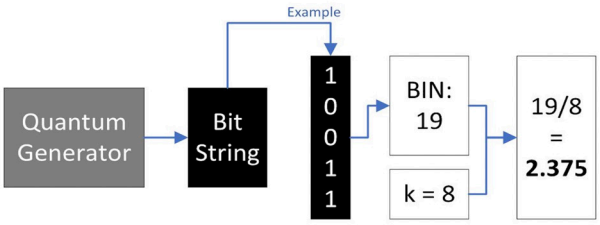

Figure 6: Quantum Analog-to-Digital Converter (QADC). Diagram showing the process of converting the bitstring output of the quantum generator into an XYZ file representing a molecule. QADC converts the bit string into a binary number (Big Endian notation—top to bottom), and it divides the resultant decimal number by a constant to get a floating-point number as the coordinate value.

state is measured on the Z basis via Pauli-Z measurement gates. The resulting measurement is a bitstring/string of 0s and 1s; the challenge lies in converting this bitstring into a meaningful vector that will represent a molecule.

To solve this problem, a Quantum Analog-to-Digital Converter (QADC) was implemented (**Figure 6**). The QADC outputs numbers to a resolution of 3 decimal points, or steps of 0.125. Note that the resolution indicates the number of possible, discrete values over a certain range and that a greater resolution requires more qubits. Once the QADC processes the output of the quantum circuit, an XYZ file, used in further classical post-processing, is created and exported to the discriminator as a Comma Separated Value (CSV) file. It is the only input to the discriminator.

The discriminator D performs the task of two-class classification compared to the generator. Because the classification task is NOT computation-intensive, we utilized a classical computer for this. In fact, to help build the discriminator, we used the classical discriminator template from PennyLane's Patch QGAN — however, theirs generated 2D images, not molecules, so the Discriminator was initially unable to run on sample input due to matrix dimension mismatches in matrix multiplication; this problem was resolved by adjusting the layer dimensions in the Discriminator (37). Though QGANs with quantum discriminators also exist, we realized that implementing a quantum discriminator to test the validity of molecules would be difficult. From a data encoding standpoint, the decision could be viewed as suboptimal since classical post-processing would still be needed to interpret the measurement data out of the quantum circuit. Despite the processing algorithm being relatively light (in terms of computational workload), having a quantum discriminator would eliminate the need for such algorithms entirely. Our classical discriminator is a vanilla neural network with three convolutional layers separated by two ReLU activation functions. The final activation function is a sigmoid function in order to classify the input CSV file as 1 (real — from the dataset) or 0 (fake — from the generator).

To train the QGAN, the training loop alternates between the generator and discriminator. The Discriminator first takes in a few examples of real molecules and trains itself on these

to get a better idea of what the "ground truth" looks like. Next, the Generator is run to get a molecule generated by feeding Gaussian noise through the parameterized quantum circuits. The quantum circuit is then measured, and a long bitstring is returned. Then using the QADC algorithm, this bitstring is decoded into an XYZ file, which is inputted into the Discriminator. The Discriminator then decides whether the generated molecule is realistic enough, and returns its verdict, from which the loss is calculated. The Discriminator determines how accurate its prediction was, and the Generator learns how realistic its generated molecule was. Using this information, the Generator uses a quantum optimization method called the Parameter-Shift method and the discriminator uses the Adaptive Moment Estimation (ADAM) optimization algorithm to update themselves to become more robust (38). The training loop repeats until an equilibrium where neither network can gain a significant advantage over the other is reached.

ACKNOWLEDGEMENTS

We would like to acknowledge the previous members of our research group, Joey Huang and Kanthi Makineedi, for contributing their team working abilities and knowledge. We'd also like to acknowledge our ASDRP advisors Mr. Edward Njoo and Mr. Robert Downing for sharing their expertise and giving us guidance on using ASDRP's computational resources. Finally, we would like to thank ASDRP themselves for giving us the opportunity to conduct this high-level and groundbreaking research.

Received: June 21, 2022
Accepted: September 21, 2022
Published: January 10, 2023

REFERENCES

1. Atkinson, Arthur J. *Principles of Clinical Pharmacology*. Elsevier Acad. Press, 2012.
2. Van Norman, Gail A. "Drugs, Devices, and the FDA: Part 1: An Overview of Approval Processes for Drugs." JACC. *Basic to translational science* vol. 1,3 170-179. 25 Apr. 2016, doi:10.1016/j.jacbts.2016.03.002
3. Wong, Chi Heem et al. "Estimation of clinical trial success rates and related parameters." *Biostatistics* (Oxford, England) vol. 20,2 (2019): 273-286. doi:10.1093/biostatistics/kxx069
4. Rowe, Sebastian. "Modern Drug Discovery: Why Is the Drug Development Pipeline Full of Expensive Failures?" *Science in the News*, 21 Apr. 2020, sitn.hms.harvard.edu/flash/2020/modern-drug-discovery-why-is-the-drug-development-pipeline-full-of-expensive-failures/.
5. Burke, Hannah. "Why Does It Cost so Much to Develop New Drugs?" *Proclinical.com*, 22 Sept. 2020, www.proclinical.com/blogs/2020-9/why-does-it-cost-so-much-to-develop-new-drugs.
6. Eliav, Ephraim. "Introduction to Computational Chemistry Laboratory." *Tel Aviv University*, www.tau.ac.il/~ephraim/intro2comp.pdf.
7. "Chemical Space." *Wikipedia*, Wikimedia Foundation, en.wikipedia.org/wiki/Chemical_space.
8. "Mehmetazizyirik/Maygen: Maygen Is an Open Source Chemical Structure Generator Based on the Orderly Graph Generation Method." *GitHub*, github.com/MehmetAzizYirik/MAYGEN.
9. De Cao, Nicola, and Thomas Kipf. "MolGAN: An implicit generative model for small molecular graphs." *arXiv* preprint, 30 May 2018, arXiv:1805.11973
10. Tsujimoto, Yutaka, Satoru Hiwa, Yushi Nakamura, Yohei Oe, and Tomoyuki Hiroyasu. "L-MolGAN: An Improved Implicit Generative Model for Large Molecular Graphs." *ChemRxiv*, 16 May 2021
11. Blanchard, Andrew E, et al. "Using Gans with Adaptive Training Data to Search for New Molecules." *Journal of Cheminformatics*, Springer International Publishing, 23 Feb. 2021. https://doi.org/10.1186/s13321-021-00494-3
12. Li, Junde, et al. "Quantum Generative Models for Small Molecule Drug Discovery." *ArXiv.org*, 23 Aug. 2021, arxiv.org/abs/2101.03438.
13. J. Li, M. Alam, C. M. Sha, J. Wang, N. V. Dokholyan and S. Ghosh, "Invited: Drug Discovery Approaches using Quantum Machine Learning," 2021 58th ACM/IEEE Design Automation Conference (DAC), 8 Nov 2021, pp. 1356-1359, doi: 10.1109/DAC18074.2021.9586268.
14. "Generative Adversarial Network." *Wikipedia*, Wikimedia Foundation, en.wikipedia.org/wiki/Generative_adversarial_network
15. Goodfellow, Ian. "Nips 2016 tutorial: Generative adversarial networks." *arXiv preprint* 31 Dec 2016. arXiv:1701.00160.
16. "Introduction to Loss Functions." *DataRobot AI Cloud*, 30 June 2022, www.datarobot.com/blog/introduction-to-loss-functions/.
17. Islam, J., Zhang, Y. GAN-based synthetic brain PET image generation. *Brain Inf.* 7, 3 30 March 2020. https://doi.org/10.1186/s40708-020-00104-2
18. Y. Cao, J. Romero and A. Aspuru-Guzik, "Potential of quantum computing for drug discovery," in *IBM Journal of Research and Development*, vol. 62, no. 6, pp. 6:1-6:20, 1 Nov.-Dec. 2018, doi: 10.1147/JRD.2018.2888987.
19. Qiskit. "Qiskit-AER/Qubitvector.hpp at 1a6d5df89a2e016afbb33a2d7088e6100348a7c4 · Qiskit/Qiskit-AER." GitHub, 18 Feb. 2021, github.com/Qiskit/qiskit-aer/blob/1a6d5df89a2e016afbb33a2d7088e6100348a7c4/src/simulators/statevector/qubitvector.hpp#L813-L819.
20. jolene. "Memory Requirements for Qiskit Aer Simulator." *Quantum Computing Stack Exchange*, 9 Mar. 2021, quantumcomputing.stackexchange.com/questions/16419/memory-requirements-for-qiskit-aer-simulator.
21. Krenn, M., Häse, F., Nigam, A., Friederich, P., & Aspuru-Guzik, A. (2020). Self-referencing embedded strings (SELFIES): A 100% robust molecular string representation. *Machine Learning: Science and Technology*, 1(4), 045024.

22. Lambert, J. (2021, Oct 6). An easier, greener way to build molecules wins the chemistry Nobel Prize. *ScienceNews*. www.sciencenews.org/article/chemistry-nobel-prize-2021-molecule-build-tool-list-macmillan
23. Iglinski, P., & Iglinski, P. (2016 March 28). Q&A: New ways to make molecules. www.rochester.edu/newscenter/qa-new-ways-to-make-molecules/
24. James Ives, M. (2020, Jul 23). New biomolecules can benefit agricultural and pharmaceutical sectors. *News-Medical*. www.news-medical.net/news/20200723/New-biomolecules-can-benefit-agricultural-and-pharmaceutical-sectors.aspx
25. Blanchard, Andrew E, et al. "Using Gans with Adaptive Training Data to Search for New Molecules." *Journal of Cheminformatics*, Springer International Publishing, 23 Feb. 2021
26. Bickerton GR, Paolini GV, Besnard J, Muresan S, Hopkins AL (2012) Quantifying the chemical beauty of drugs. *Nat Chem* 4(2):90–98. doi.org/10.1038/nchem.1243
27. OMx Personal Health Analytics Staff. "Lipinski's Rule of Five." *Lipinski's Rule of Five | DrugBank Help Center*, dev.drugbank.com/guides/terms/lipinski-s-rule-of-five.
28. Hajiabadi, Hossein. "Hartree Fock Method: A Simple Explanation." *In Silico Sciences*, 3 Sept. 2022, insilicosci.com/hartree-fock-method-a-simple-explanation/.
29. "Pytorch." *PyTorch*, 2022, pytorch.org/.
30. "Pennylane." *PennyLane*, 2022, pennylane.ai/.
31. "Qiskit 0.39.1 Documentation." *Qiskit* 0.39.1 Documentation - Qiskit 0.39.1 Documentation, 2022, qiskit.org/documentation/.
32. "NumPy." NumPy Documentation, 2022, numpy.org/doc/.
33. "Pybel." *Pybel - Open Babel v2.3.1 Documentation*, 2022, openbabel.org/docs/dev/UseTheLibrary/Python_Pybel.html.
34. L. Ruddigkeit, R. van Deursen, L. C. Blum, J.-L. Reymond, Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17, *J. Chem. Inf. Model.* 52, 2864–2875, 2012.
35. R. Ramakrishnan, P. O. Dral, M. Rupp, O. A. von Lilienfeld, Quantum chemistry structures and properties of 134 kilo molecules, Scientific Data 1, 140022, 2014.Anderson, Scott. "Visualizing Molecules with GOpenMol." *Tutorial - Department of Chemistry - The University of Utah*, n.d.
36. "PPQM/Dataset-QM9." *GitHub*, 2022, github.com/ppqm/dataset-qm9.
37. Ellis, J. "Quantum Gans." *Quantum GANs - PennyLane Documentation*, 27 Jan. 2022, pennylane.ai/qml/demos/tutorial_quantum_gans.html.
38. Kingma, Diederik P., and Jimmy Ba. "Adam: A Method for Stochastic Optimization." *ArXiv.org*, 30 Jan. 2017, arxiv.org/abs/1412.6980.