2023 S.T. Yau High School Science Award

Research Report

The Team

Name of team member: Haorun Li
School: Horace Mann School
City, Country: Bronx, USA

Name of supervising teacher: Liting Sun
Job Title: Postdoc
School/Institution: UC Berkeley
City, Country: Berkeley, USA

Title of Research Report

Multi-Property De Novo Drug Design Using Multi-Objective Deep Reinforcement Learning

Date

06/21/2023

Multi-Property De Novo Drug Design Using Multi-Objective Deep Reinforcement Learning

HAORUN LI, Horace Mann School, USA

Drug design has been an expensive and slow but vital process for saving lives, especially in the presence of world pandemics such as COVID-19. To make the drug design process more cost-effective and efficient, researchers have been exploring the aid of computers and even artificial intelligence. In this work, we developed a deep reinforcement learning (RL) based multi-objective method to enable multi-property de novo drug design. The goal of such an algorithm is to make RL-based drug design processes more applicable in real life compared to single-property-driven design approaches. We have studied two multi-objective methods in our experiments: the weighted sum method and the lexicographic-order method. The experimental results showed that compared to the single-objective method, both methods can effectively improve the efficiency of the drug design process. We ran the multi-objective algorithms to increase Janus kinase 2 (Jak2) inhibition and hydrophobicity (LogP) drug-likeness and found that the lexicographic-order method (the most optimal method in our experiments) can effectively increase the Jak2 inhibition by 26.9% and LogP drug-likeness by 10.0% simultaneously compared to the baseline (unbiased drug design) while 54.5% of the generated compounds are chemically valid, which far outperforms previously proposed methods in literature such as the single-objective approaches. We also found that since the weighted-sum method is sensitive to the selection of different weights, the lexicographic method is more reliable. The proposed method can be easily extended to account for more properties. Overall, it will greatly reduce the time frame and costs of drug design.

Keywords— Multi-Objective, Deep Reinforcement Learning, Multi-Property De Novo Drug Design, Targeted Therapy

1 INTRODUCTION

Drug design historically has been a lengthy and expensive process, taking on average 10 to 15 years and costing around US $2 billion for the discovery of a new drug [1]. The ability to design a drug more efficiently and cheaply will save lives. Recent world events, such as COVID-19, have only reinforced the necessity for more efficient drug design methods.

Recent development in artificial intelligence (AI) has shown promising results of significantly decreasing drug development times and costs, and it is revolutionizing the way how scientists approach drug design. Consider Fig. 1, which shows the major paradigms and historical development of the drug design process.

Historically, drug design approaches have been phenotypic, meaning that new drugs either accidentally found, such as penicillin, or through forward pharmacology, which is the screening of intact cells or whole organisms to identify substances of the desired therapeutic effect. In forward pharmacology, the lead compound (i.e., the protein giving the therapeutic effect) might remain unknown even after the drug's activity and efficacy are determined. The complications even after discovering a potential drug further contribute to the slow design process [2].

In recent years, reverse pharmacology has become more popular; under this paradigm, a hypothesis is made about a lead compound, then it is further developed into drug candidates or formulations. Compared to forward pharmacology, this new technique is much more efficient, with a drug design process of fewer than five years, whereas forward pharmacology takes more than ten years to launch [3].

The advancement in computational resources allows for de novo drug design. The word "de novo" means "from the beginning," showing that the molecules are generated from scratch with computer assistance. Under de novo design, there are two main approaches, such are structure-based and ligand-based design. These approaches are guided by protein structure and function or binding characteristics, respectively. Computer programs such as quantitative structure-activity relationships (QSAR) and pharmacophore modeling are used to obtain this information [4].

Property-based drug design is a more specific type of de novo drug design, and it is an evolution of structure or ligand-based design. It considers factors such as size, shape, lipophilicity, hydrogen bonding capability, and polarity. The characteristics of a drug, or its physicochemical profiles, can be manipulated under property-based design in order to fit the lead compound hypothesis [5].

Author's address: Haorun Li, rainli0907@gmail.com, Horace Mann School, Bronx, NY, USA.

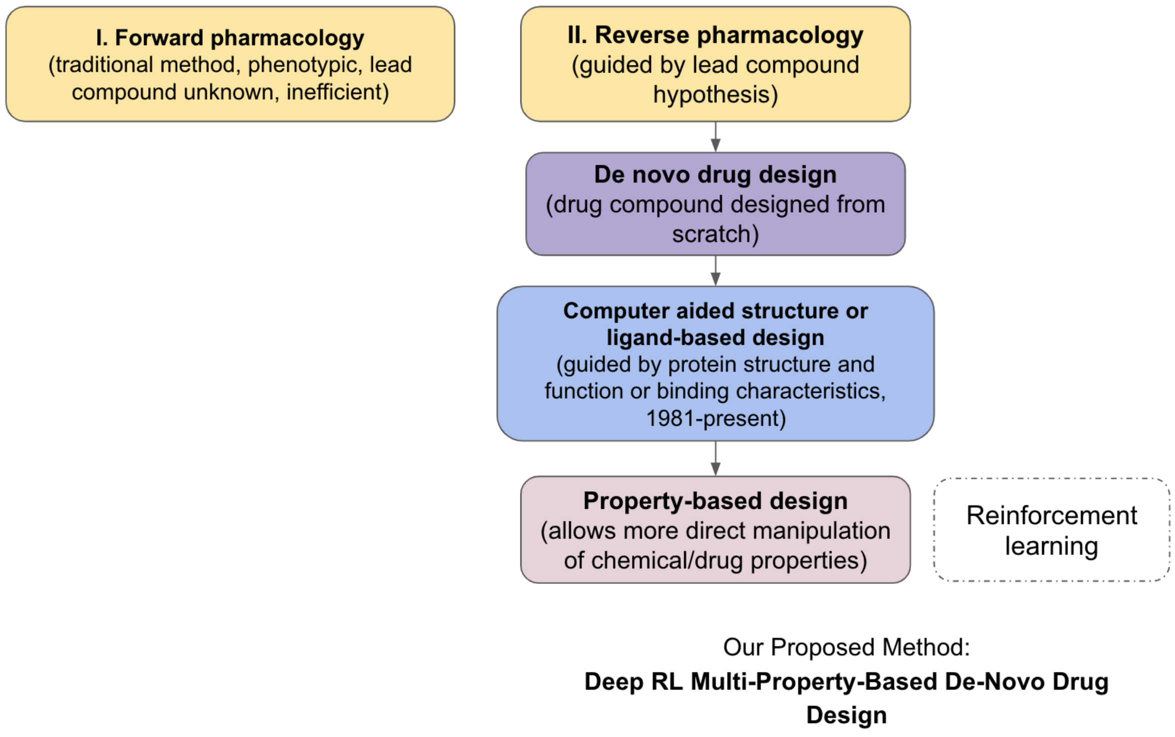

I. Forward pharmacology
(traditional method, phenotypic, lead compound unknown, inefficient)
II. Reverse pharmacology
(guided by lead compound hypothesis)
De novo drug design
(drug compound designed from scratch)
Computer aided structure or ligand-based design
(guided by protein structure and function or binding characteristics, 1981-present)
Property-based design
(allows more direct manipulation of chemical/drug properties)
Reinforcement learning
Our Proposed Method:
Deep RL Multi-Property-Based De-Novo Drug Design

Fig. 1. Different drug design paradigms and processes.

The main challenge to property-based design, and de novo drug design as a whole, in recent years is the lack of computational resources and technology to efficiently utilize this form of drug design. Previous studies have estimated the size of the "drug-like" space; the estimated number of molecules with viable drug structures is around 10^{60}, and about 10^{20} to 10^{24} for all molecules up to 30 atoms [6]. Prior to advancements in machine learning and computational power in recent years, the drug space was too large for property-based design.

There have been recently proposed reinforcement learning (RL) based solutions for drug design, but all of them have major drawbacks. For example, we referenced Popova et al. [7], which proposed a solution focusing on a single-objective RL approach, meaning that only one property can be optimized at a time. This type of RL solution is inefficient in any real application, as most drug designs are complex and require the manipulation of many properties. In Zhou et al. [8], the algorithm modified *existing* molecules to achieve 100% chemical validity, which greatly reduced the possible space of molecules it can generate. Similarly, Ståhl et al. [9] used fragment-based drug discovery (i.e., replacing fragments of a known set of lead compounds to achieve desired properties). Because this method is dependent on known lead compounds, it heavily limits the possible drugs discovered. We aim to address all of these issues in our algorithm, which uses multi-objective RL for de novo drug design, allowing drug compounds to be constructed from scratch while satisfying multiple desired properties.

In this work, we chose to optimize two important drug design properties simultaneously: Janus kinase 2 (Jak2) inhibitor activity and hydrophobicity using our algorithm. Jak2 inhibitor is an enzyme involved in controlling cellular growth and division, making it a common target in cancer mutations. We aim to maximize Jak2 inhibition, which can be measured in terms of pIC50 values; pIC50 is the negative log of IC50, and a higher pIC50 means more Jak2 inhibition [10]. Next, hydrophobicity is a key component in drug design, affecting drug absorption, transportation, and distribution within the body; it is measured in octanol-water partition coefficient values, LogP. Lipinski's rule of five, which evaluates drug-likeness, states that the LogP should not exceed 5 in oral drugs [11].

We consider two types of algorithms to accomplish multi-objective RL, the weighted-sum method and the lexicographic method. Both algorithms accomplished the desired optimization on multiple properties with high efficiency. The single-objective algorithms like the ones previously done in literature cannot even produce a representative result due to how few of the molecules it generated can fit both optimization criteria, showing much worse efficacy. The lexicographic multi-objective algorithm yielded a 25.9% improvement on Jak2 inhibition and 10.0% increase in the number of drugs in a desired range of LogP, with a 54.5% of the $10,000$ generated molecules being valid. On the other hand, with the method in Popova et al. [7], we had to take the intersection of two single-objective models, one trained for Jak2, while the other for LogP, to accomplish both optimization goals. However, it yielded a mere 10 valid molecules out of $10,000$ generated, which is 0.1%. Such a low valid percentage is hard to be deployed in real applications.

In summary, in this work, we propose a novel de novo drug design process under the property-based design paradigm that is aided by multi-objective RL, which allows for multi-property optimization. Our experimental results showed that the proposed algorithm enables property-based RL de novo drug design to be better used in real-world applications.

2 FRAMEWORK

2.1 Introduction to RL

RL solves decision-making problems by rewarding the agent for certain actions, and the agent has the goal of maximizing its reward. RL offers a key advantage compared to other forms of machine learning in that it relies on the agent's prior experience and the interaction between the agents and the environment to make decisions rather than a set of pre-collected training data.

A simple RL problem can be modeled in the framework of a Markov decision process (MDP):

- a set of environment and agent states, S;
- a set of actions of the agent, A;
- probability of a transition from state s to s' after taking action a: $P(s, s', a) = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$
- the reward after transitioning from s to s' after taking action a: $R(s, s', a)$

A policy $\pi(s)$ is a set of actions that the agent takes starting from state s. The goal is to discover the optimal policy π^* such that the reward is maximized, i.e., π^* will result in the maximal value of

$$\mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(s_t, s_{t+1}, a_t)\right] \tag{1}$$

where γ is a discount factor, and T is a time frame that the agent will make decisions on, which can possibly be infinite. The expectation is taken on $s_{t+1} \sim P(s_t, s_{t+1}, a_t)$, which is the probability the agent goes from s_t to s_{t+1} on the action of a_t. After obtaining π^*, the agent is able to make the most optimal next action based on a given state. While this is a general RL problem format, the formulation for this application will be described below.

2.2 RL Problem Formulation

Digital Representation of Molecules. We represent molecules using the simplified molecular-input line-entry system (SMILES) [12], in which molecular structures are represented as short ASCII strings denoting the atoms and bonds of that molecule. For example, Ibuprofen can be denoted as CC(C)CC1=CC=C(C=C1)C(C)C(=O)O.

The initial state s_0 is unique and represents the empty SMILES string. Given a target length T of the generated molecule, the state s_T is a terminal state. Importantly, all rewards at intermediate states, $r(s_t)$ where $t < T$, is set to zero, and the reward of the entire SMILES string is given by $r(s_T)$, and

$$r(s_T) = f(P(s_T)) \tag{2}$$

where P is the predictive model that evaluates s_T based on some property (see Section 3.2), and f is some customized reward function for a specific property. Our goal is to find a vector of parameters Θ, which represents the most optimal policy, which will return the next optimal action (i.e., the next character in the SMILES string). The formats of the optimal policy can be any explicit functions or deep neural networks. Consider $J(\Theta)$ to be some measure of performance, which is the total reward of the SMILES string in our case. Let S^* be the set of all terminal states, then we can write

$$J(\Theta) = \mathbb{E}\left[r\left(s_T\right) \mid s_0, \Theta\right] = \sum_{s_T \in S^*} p_\Theta\left(s_T\right) r\left(s_T\right) \rightarrow \max \tag{3}$$

Since we want to maximize this quantity, we need to compute the gradient; with the gradient, we can adjust the parameters Θ to achieve gradient descent or ascend in the form of

$$\Theta_{t+1} = \Theta_t + \alpha \nabla J\left(\Theta_t\right) \tag{4}$$

However, the gradient cannot be computed directly because $|S^*|$ can be up to 10^{60}, which is too large to be iterated over. Thus, we need to approximate by sampling from the model's distribution.

$$\begin{aligned} J(\Theta) &= \mathbb{E}\left[r\left(s_T\right) \mid s_0, \Theta\right] \\ &= \mathbb{E}_{a_1 \sim p_\Theta(a_1|s_0)} \mathbb{E}_{a_2 \sim p_\Theta(a_2|s_1)} \ldots \mathbb{E}_{a_T \sim p_\Theta(a_T|s_{T-1})} r\left(s_T\right) \end{aligned} \tag{5}$$

The above equation says $J(\Theta)$ can be estimated via sampling actions a_t where t goes from 0 to T, and the unbiased estimation for $J(\Theta)$ is equal to the reward given at each step, but recall we assume that all intermediate rewards are 0.

We also need to utilize a trick popularized by the REINFORCE algorithm [13], which says

$$\partial_\Theta J(\Theta) = J(\Theta) \frac{\partial_\Theta J(\Theta)}{J(\Theta)} = J(\Theta) \partial_\Theta \log J(\Theta) \tag{6}$$

When used in combination with the idea of approximating a mathematical expectation as a sum shown in Eq. 5, we can differentiate $J(\Theta)$

$$\begin{aligned} \partial_\Theta J(\Theta) &= \sum_{s_T \in S^*} \left[\partial_\Theta p_\Theta\left(s_T\right)\right] r\left(s_T\right) \\ &= \sum_{s_T \in S^*} p_\Theta\left(s_T\right) \left[\partial_\Theta \log p_\Theta\left(s_T\right)\right] r\left(s_T\right) \\ &= \sum_{s_T \in S^*} p_\Theta\left(s_T\right) \left[\sum_{t=1}^{T} \partial_\Theta \log p_\Theta\left(a_t \mid s_{t-1}\right)\right] r\left(s_T\right) \\ &= \mathbb{E}_{a_1 \sim p_\Theta(a_1|s_0)} \mathbb{E}_{a_2 \sim p_\Theta(a_2|s_1)} \ldots \mathbb{E}_{a_T \sim p_\Theta(a_T|s_{T-1})} \left[\sum_{t=1}^{T} \partial_\Theta \log p_\Theta\left(a_t \mid s_{t-1}\right)\right] r\left(s_T\right) \end{aligned} \tag{7}$$

In simpler terms, the gradient of $J(\Theta)$ can be obtained by sampling a trajectory from the distribution, then differentiating the natural log of the probability to obtain that trajectory, multiplied by a reward at the end. We consider the reward to be the advantage function, which specifies the magnitude of the gradient descent.

Deep RL. Deep RL refers to incorporating neural networks into the RL decision-making process. In this work, we utilize deep RL, hence, the paramters Θ is a vector of parameters that represents the deep network which gives the next optimal action given a state. In other words, Θ is a neural network that takes a state as an input and returns the next action as an output.

Reward Functions. First, we define the reward function of Jak2 to be

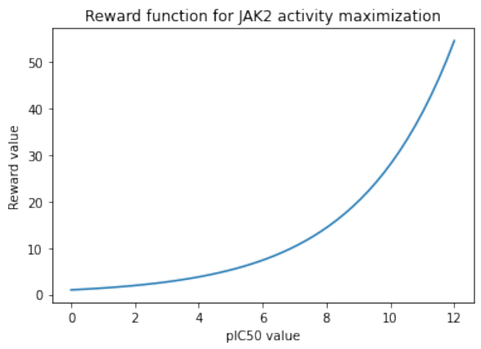
$$\text{reward Jak2} = e^{\frac{x}{3}} \tag{8}$$

as shown in Fig. 2(a). The majority of all known compounds lie within a pIC50 range of 0 to 12 [7]. The mean of the normal distribution is about 6. This corresponds to a range of rewards from about 1 to 55, with the mean of the normal distribution having a reward of 7.39.

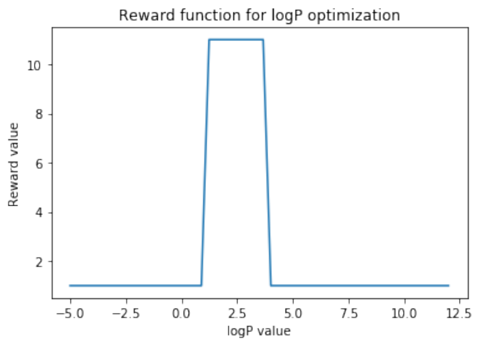
For hydrophobicity, one of the components of Lipinski's rule of five is that orally active drugs should have LogP of less than 5 [11]. We define a stricter range of LogP from 1 to 4. Therefore, we designed a reward function to be a piecewise function,

$$\text{reward LogP} = \begin{cases} 11.0, & \text{if } 1.0 < \log P < 4.0 \\ 1.0, & \text{otherwise} \end{cases} \tag{9}$$

as shown in Fig. 2(b).



(a) Jak2 reward function



(b) LogP reward function

Fig. 2. Rewards functions for maximizing Jak2 inhibition and LogP drug-likeness optimization.

3 MODEL STRUCTURE

The model is consisted of three parts, as shown in Fig 3. First, a generative model is responsible for generating SMILES sequences. Next, the generated SMILES will be passed to a predictive model that predicts certain properties. Lastly, the multi-objective RL methods optimize the generative model using the predicted properties.

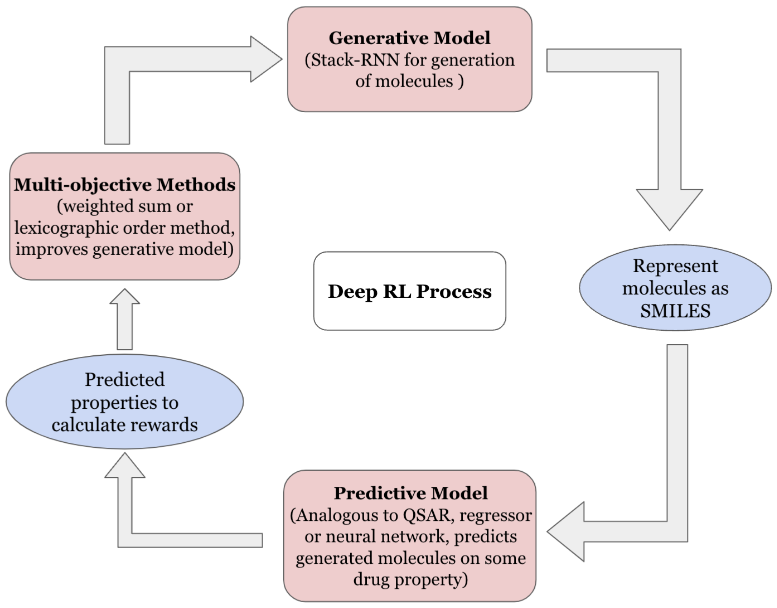

Generative Model
(Stack-RNN for generation of molecules)
Multi-objective Methods
(weighted sum or lexicographic order method, improves generative model)
Deep RL Process
Represent molecules as SMILES
Predicted properties to calculate rewards
Predictive Model
(Analogous to QSAR, regressor or neural network, predicts generated molecules on some drug property)

Fig. 3. Model structure involves three main processes, a generative model, a predictive model, and multi-objective RL methods

3.1 Generator

The generator uses a stack-augmented RNN (Stack-RNN). A Stack-RNN provides long-term memory in ways that a general long short-term memory (LTSM) or gated recurrent unit (GRU) layer cannot, as proposed in Joulin and Mikolov [14]. One common problem that can be modeled by a Stack-RNN but not by a regular RNN is the Dyck language, which consists of open and closed square brackets. A Stack-RNN is able to correctly learn all open square brackets must be matched with the respective closed brackets [15]. Furthermore, normal RNNs often fail to capture longer sequences due to a *vanishing gradient*.

Fig. 4 shows the flow of the Stack-RNN used for our specific application. The input layer is passed through GRU layers, and the hidden layer will be modified by a stack. The stack is modified via a stack control vector, indicating the probabilities of the three possible operations: PUSH (current hidden layer), POP, and NO-OPERATION. This control vector is also learned via deep learning, and the final stack is the expected value considering the three possible resulting stacks. The hidden layer from each time will be augmented to contain the top of the stack, achieving long-term memory.

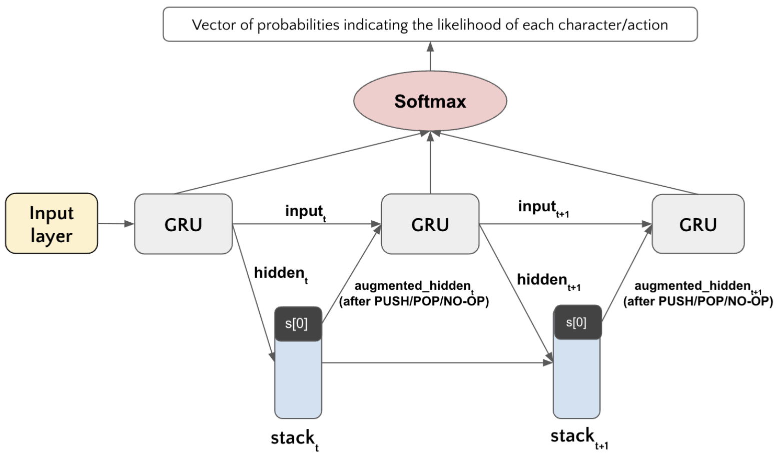

Vector of probabilities indicating the likelihood of each character/action
Softmax
Input layer
GRU
GRU
GRU
input_t
input_t+1
hidden_t
hidden_t+1
augmented_hidden_t (after PUSH/POP/NO-OP)
augmented_hidden_t+1 (after PUSH/POP/NO-OP)
s[0]
s[0]
stack_t
stack_t+1

Fig. 4. Structure of the Stack-RNN

3.2 Predictor

An implementation analogous to QSAR will suffice. The predictors we selected for Jak2 and LogP were different. For Jak2, we opted for a regressor-based random forest due to the number of data points that were available. With only 4,000 data points, training a reliable neural network-based predictor is difficult. There were about 18,000 data points for LogP, so we trained an RNN model.

3.3 Multi-objective RL

Formally, the multi-objective RL problem can be defined as:

$$\begin{aligned} \text{min/max} \quad & f_i(x) \\ \text{subject to} \quad & \text{some set of constraints} \end{aligned} \tag{10}$$

where $f_i(x)$ is the i-th objective function that we want to optimize. In this project, we have i=2 where $f_1(x)$ is the reward for Jak2 and $f_2(x)$ is the reward for LogP. Typically, in multi-objective optimization, we say solution x_1 dominates another solution x_2 when x_1 is strictly better than x_2 in all possible properties. Moreover, x_1 and x_2 have a non-dominating relationship when neither is strictly better than another. The set of non-dominating solutions in the valid decision space is called the Pareto-optimal set. The task of multi-objective RL is to discover the Pareto-optimal set.

There have been quite a few approximate solutions to find the optimal set for multi-objective optimization. In this work, we have explored two different methods. First, we explored the weighted sum method which is a straightforward approach to accomplish multi-property optimization. By creating a weight vector w, where w_i represents the importance of the i-th property, then we can represent the reward of a trajectory

$$r(s_T) = w_i \times f_i(P_i(s_T)) \tag{11}$$

where f_i is the reward function for the ith property, and P_i is the predictor model for the i-th property.

The pseudo-code for the weighted sum method is given in Algorithm 1.

Algorithm 1 Weighted sum method

```
1: for $iterations = 1, 2, \ldots, M$ do
2:     sample trajectory $t$
3:     let total_reward = 0
4:     for $ith\_property = 1, 2, \ldots, N$ do
5:         let current_reward = reward of t on ith_property
6:         total_reward += weight of ith_property * current_reward
7:     run policy gradient on the total_reward
```

While this method is intuitive to come up with, it has some practical challenges. The weights need to be carefully adjusted to achieve an optimal result, especially when there are multiple properties. Furthermore, it cannot find some Pareto-optimal solutions in the case of a nonconvex objective space, shown in the Fig. 5.

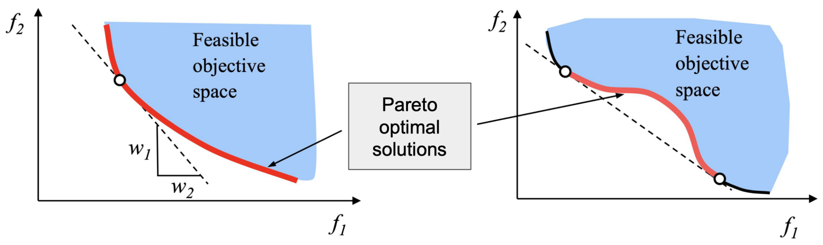


Fig. 5. Convex versus concave objective space

Therefore, we further studied the lexicographic method [16]; given the order of importance of the properties, then we can solve the following problem one at a time, where x is some model we train, $x_j{}^*$ is the most optimal model on the j-th property, f_i is the objective function for the i-th property, and k is the number of properties:

$$\begin{array}{lll} \min & f_i(x) & i = 1, 2, \ldots, k \\ \text{subject to} & f_j(x) \leq f_j(x_j^*), & j = 1, 2, \ldots, i-1, i > 1 \end{array} \tag{12}$$

We train a new model x for every additional property to optimize, and the network x must be at least as good as all previously trained, most optimal models on their respective properties. Note that $f_j(x_j^*)$ is not equivalent to the single-objective optimum on the objective f_j after $j = 1$ because additional constraints are introduced when training x_j^* (i.e., the other properties).

The pseudo-code for the lexicographic method is given in Algorithm 2.

Algorithm 2 Lexicographic method

```
1: for $ith\_property = 1, 2, \ldots, N$ do
2:     begin training the current model x
3:     for $iterations = 1, 2, \ldots, M$ do
4:         sample trajectory $t$ from the x
5:         for $j = 1, 2, \ldots, ith\_property - 1$ do
6:             sample trajectory $t\_prev$ from $x_j^*$ (i.e., the model most optimal for the j-th property)
7:             if reward of t on the $j\_th$ property is worse than $t\_prev$ then
8:                 discard $t$ and go to line 4
9:         run policy gradient on the reward of $t$.
```

4 EXPERIMENTS

4.1 Experiment Data and Design

The generative model is trained on over 2 million SMILES strings obtained from ChEMBL [17]. The data for Jak2 inhibition and LogP were also obtained from ChEMBL, with some LogP data coming from PubChem [18]. We converted units on the Jak2 inhibition data from IC50 to pIC50 for easier training and visualization purposes. For both properties, inconclusive data was discarded for more accurate results. In the end, there are about 4,000 Jak2 data points and 18,000 for LogP.

We tested two different types of algorithms to accomplish maximizing Jak2 inhibition while range-optimizing LogP: the weighted sum method and the lexicographic order method.

4.2 Results with the weighted sum method

First, we attempted to optimize Jak2 inhibition and hydrophobicity (LogP) using the weighted sum method, the best result was achieved with a weight of 0.8 and 0.2 on Jak2 and LogP respectively. The results are displayed in Fig. 6. The Jak2 mean value increased to 8.2 from 6.5 (unbiased), which is a 25.5% increase. The percentage of drugs in the LogP region we defined previously increased to 99.1% from 90.1%, which is a 10.0% increase. Overall, the valid percentage of all molecules remained high at 54.5%, though the valid percentage showed a slight decrease compared to the baseline of 63.3%[1].

However, using other weights does not achieve the same results. Fig. 7 shows the results after training 3,000 iterations on a 0.5 and 0.5 split. The Jak2 value remains to be similar to the unbiased baseline results with a mere 2.5% increase. LogP did show some improvement, but that only reflects it being an easier property to optimize compared to Jak2. This means one must bias more of the reward to Jak2 than LogP. Jak2 is likely a much harder property to optimize, which we have shown when conducting the single-objective baseline testing. Fig. 8 shows the number of iterations until convergence during single-objective training: LogP took about 300 iterations for the rewards to show convergence, while Jak2 took 2,000 iterations to show convergence.

Note, we also tried other weights, such as a 0.65 and 0.35 split, all of which yielded similar, but less optimal, results compared to the 0.8 and 0.2 splits.

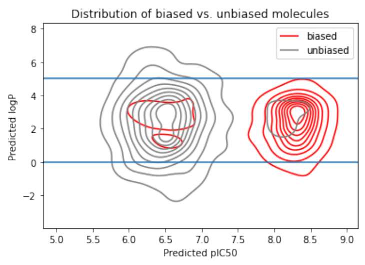
4.3 Results with the Lexicographic order method

Given the fact that different weights in the weighted sum method can generate significantly different results, we studied the lexicographic order method on the same properties (i.e., Jak2 and LogP). The benefits of this method are clear, as there was no need to set a definite reward split. We assigned Jak2 to be more important than LogP in this experiment. After 3,000 training iterations on the Jak2 property and only 1,000 on LogP, we obtained even better results compared to the 0.8 and 0.2 splits. The advantage of the lexicographic method is that it eliminates any uncertainty in whether the selected weight is the most optimal compared to the weight sum method. The results are shown in Fig. 9. The Jak2 mean value showed a 25.9% increase from 6.5 (unbiased) to 8.2. The LogP druglike percentage increased 10.0% from 90.0% to 99.1%. The valid percentage, similar to the 0.8 and 0.2 splits, showed a slight decrease from 63.3% to 54.4%. The lexicographic method achieved slightly better

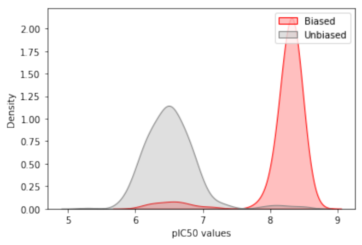
[1]We believe that such a slight decrease in valid percentage is not a concern since 54.5% is still a very high valid percentage in real applications.

	Biased results	Baseline (unbiased) results	Percent change
Jak2 mean value	8.216515271	6.543937658	0.2555919235
LogP drug-like percentage	0.990990991	0.9008394544	0.1000750314
Valid percentage	0.545	0.633	-0.1390205371

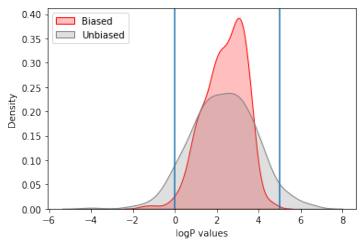
(a) The results of using the weighted sum method with split = $(0.8, 0.2)$



(b) Overall distribution



(c) Jak2 only

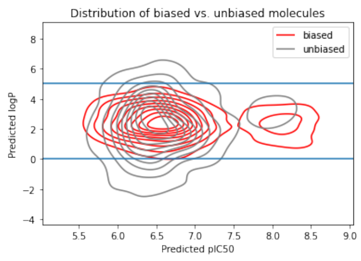


(d) LogP only

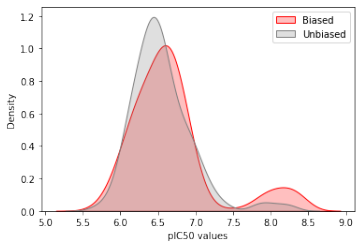
Fig. 6. The Jak2 and LogP data and distribution of generated molecules using a 0.8 to 0.2 split

	Biased results	Baseline (unbiased) results	Percent change
Jak2 mean value	6.708821593	6.543937658	0.02519644044
LogP drug-like percentage	0.9879518072	0.9008394544	0.09670130727
Valid percentage	0.82	0.633	0.2954186414

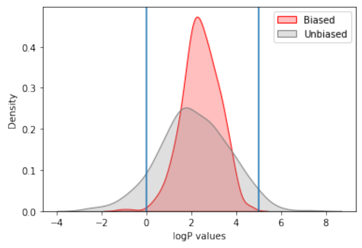
(a) The results of using the weighted sum method with split = $(0.5, 0.5)$



(b) Overall distribution



(c) Jak2 only



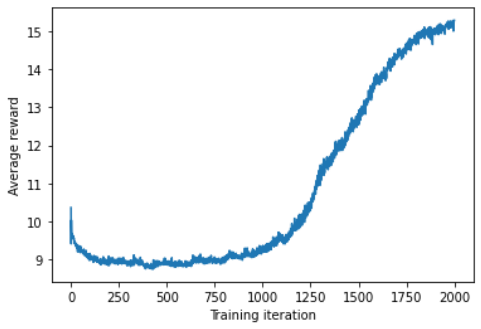
(d) LogP only

Fig. 7. The Jak2 and LogP data and distribution of generated molecules using a 0.5 to 0.5 split

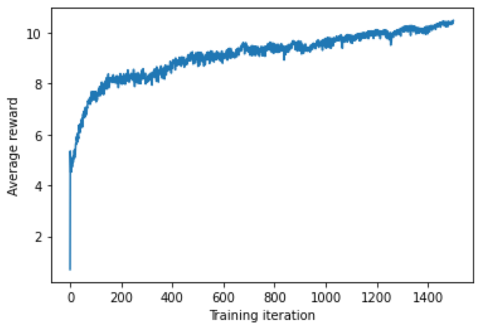
performance than the 0.8 and 0.2 splits, which is expected. The main advantage of the lexicographic method is that it is not sensitive to user-given weight compared to the weighted sum method.
Generated molecules. Fig. 10 shows some examples of the generated molecules after the RL process.

5 CONCLUSION

In this work, we proposed a multi-objective deep RL method to design drugs with multiple property requirements. The experiments that were conducted demonstrate that multi-objective RL for property-based de novo drug design is feasible and



(a) Jak2 training reward

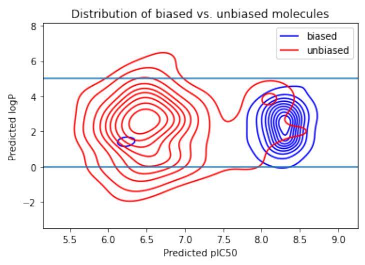


(b) LogP training reward

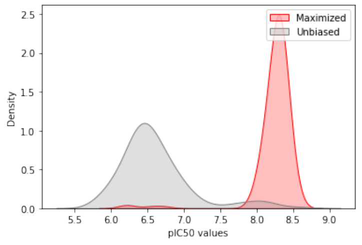
Fig. 8. The single-objective training rewards of Jak2 and LogP. Jak2 used 2,000 iterations until convergence, while LogP took only 300 iterations. The rate of convergence is much slower for Jak2.

	Biased results	Baseline (unbiased) results	Percent change
Jak2 mean value	8.239986845	6.543937658	0.2591786896
LogP drug-like percentage	0.9909090909	0.9008394544	0.09998411606
Valid percentage	0.545	0.633	-0.1390205371

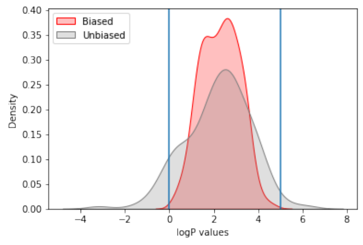
(a) The results of using the lexicographic method



(b) Overall distribution



(c) Jak2 only



(d) LogP only

Fig. 9. The Jak2 and LogP distribution of generated molecules using the lexicographic method, with Jak2 having precedence over LogP.

effective. In order to achieve a similar goal using previously proposed single-objective property-based drug design methods [7], one has to run the model separately for all properties, then take the intersection of the output sets. The performance of this approach is far worse. The single-objective models generated 10,000 molecules each, but the intersection only consisted of 10 molecules, which is 0.1%, which means this approach is unusable. Therefore, the only way to achieve multi-property-based drug design is by using the approaches proposed in this paper.

One can easily extend this algorithm to account for more molecular properties beyond Jak2 and LogP, which involves creating a new predictor. The predictor can be any model that can capture patterns within a string and predict some value based on the training data. This can be anything from a random forest regressor, such as the Jak2 predictor, to a full-scale RNN, like the LogP predictor.

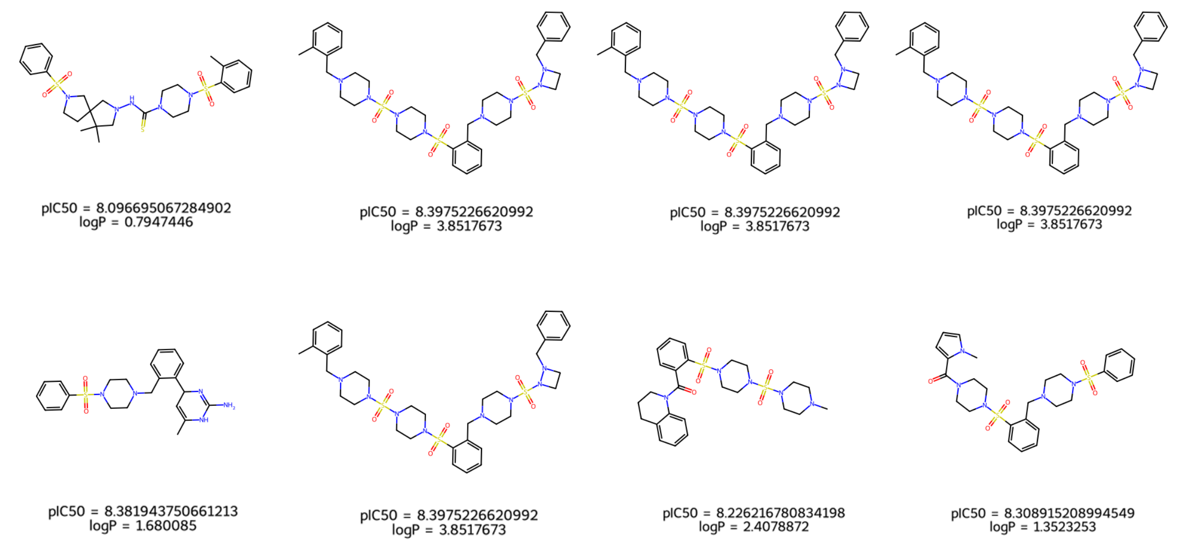


Fig. 10. Generated compounds after lexicographic order multi-objective RL.

The application of this type of drug design is widespread and in demand. Pharmaceutical industries can modify and customize the components of this algorithm to train models to create drugs that target specific diseases, given that the properties of drugs that are effective towards that disease are known. Pharmaceutical companies can begin testing, such as in-vivo, in-vitro, and clinical, in a much faster time using this algorithm. When drugs are needed as quickly as possible, this algorithm can save lives by increasing efficiency and reducing the cost of drug development.

6 ACKNOWLEDGEMENTS

I want to thank my advisor, Liting Sun, for guiding me through this process. Also, I want to thank my family for their endless support.

REFERENCES

[1] Mootaz M Salman, Zaid Al-Obaidi, Philip Kitchen, Andrea Loreto, Roslyn M Bill, and Richard Wade-Martins. Advances in applying computer-aided drug design for neurodegenerative diseases. *International journal of molecular sciences*, 22(9):4688, 2021.

[2] Wei Zheng, Natasha Thorne, and John C. McKew. Phenotypic screens as a renewed approach for drug discovery. *Drug Discovery Today*, 18(21):1067–1073, 2013. ISSN 1359-6446. doi: https://doi.org/10.1016/j.drudis.2013.07.001. URL https://www.sciencedirect.com/science/article/pii/S135964461300202X.

[3] Young-Joon Surh. Reverse pharmacology applicable for botanical drug development - inspiration from the legacy of traditional wisdom, Oct 2011. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3943000/.

[4] Varnavas D Mouchlis, Antreas Afantitis, Angela Serra, Michele Fratello, Anastasios G Papadiamantis, Vassilis Aidinis, Iseult Lynch, Dario Greco, and Georgia Melagraki. Advances in de novo drug design: From conventional to machine learning methods. *International journal of molecular sciences*, 22(4):1676, 2021.

[5] Paul D. Leeson and Robert J. Young. Molecular property design: Does everyone get it? *ACS Medicinal Chemistry Letters*, 6(7):722–725, 2015. doi: 10.1021/acsmedchemlett.5b00157. URL https://doi.org/10.1021/acsmedchemlett.5b00157.

[6] Jean-Louis Reymond. The chemical space project. *Accounts of Chemical Research*, 48(3):722–730, 03 2015. doi: 10.1021/ar500432k. URL https://doi.org/10.1021/ar500432k.

[7] Mariya Popova, Olexandr Isayev, and Alexander Tropsha. Deep reinforcement learning for de novo drug design. *Science advances*, 4(7): eaap7885, 2018.

[8] Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N Zare, and Patrick Riley. Optimization of molecules via deep reinforcement learning. *Scientific reports*, 9(1):1–10, 2019.
[9] Niclas Ståhl, Goran Falkman, Alexander Karlsson, Gunnar Mathiason, and Jonas Bostrom. Deep reinforcement learning for multiparameter optimization in de novo drug design. *Journal of chemical information and modeling*, 59(7):3166–3176, 2019.
[10] Kavitha Gnanasambandan and Peter P Sayeski. A structure-function perspective of jak2 mutations and implications for alternate drug design strategies: the road not taken. *Current medicinal chemistry*, 18(30):4659–4673, 2011.
[11] Christopher A Lipinski, Franco Lombardo, Beryl W Dominy, and Paul J Feeney. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Advanced drug delivery reviews*, 23(1-3):3–25, 1997.
[12] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988. doi: 10.1021/ci00057a005. URL https://doi.org/10.1021/ci00057a005.
[13] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3): 229–256, 1992. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.
[14] Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. *Advances in neural information processing systems*, 28, 2015.
[15] Mirac Suzgun, Sebastian Gehrmann, Yonatan Belinkov, and Stuart M Shieber. Memory-augmented recurrent neural networks can learn generalized dyck languages. *arXiv preprint arXiv:1911.03329*, 2019.
[16] R Timothy Marler and Jasbir S Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.
[17] David Mendez, Anna Gaulton, A Patrícia Bento, Jon Chambers, Marleen De Veij, Eloy Félix, María Paula Magariños, Juan F Mosquera, Prudence Mutowo, Michał Nowotka, et al. Chembl: towards direct deposition of bioassay data. *Nucleic acids research*, 47(D1):D930–D940, 2019.
[18] Sunghwan Kim, Jie Chen, Tiejun Cheng, Asta Gindulyte, Jia He, Siqian He, Qingliang Li, Benjamin A Shoemaker, Paul A Thiessen, Bo Yu, et al. Pubchem in 2021: new data content and improved web interfaces. *Nucleic acids research*, 49(D1):D1388–D1395, 2021.

7 APPENDIX

The most important code snippets are shown below, including the policy gradient code for weighted sum and the edits made to accomplish the lexicographic order method.

7.1 Weighted sum method

```python
import torch
import torch.nn.functional as F
import numpy as np
from rdkit import Chem, RDLogger
from utils import canonical_smiles
from utils import get_desc, get_fp
RDLogger.DisableLog('rdApp.*')

class Reinforcement(object):
    def __init__(self, generator, predictors, get_reward_func_list, weights, gen_data):
        super(Reinforcement, self).__init__()
        self.generator = generator
        self.predictors = predictors
        self.get_reward_func_list = get_reward_func_list
        self.weights = weights
        self.gen_data = gen_data

    def get_valid_percentage(self, predictor, n_to_generate, **kwargs):
        generated = []
        for i in range(n_to_generate):
            generated.append(self.generator.evaluate(self.gen_data, predict_len=120)[1:-1])

        sanitized = canonical_smiles(generated, sanitize=False, throw_warning=False)[:-1]
        unique_smiles = list(np.unique(sanitized))[1:]
        smiles, prediction, nan_smiles = predictor.predict(unique_smiles, get_features=get_fp)
        return len(smiles)/n_to_generate

    # Get weighted reward
    def get_total_reward(self, trajectory, **kwargs):
        total_reward = 0
        for i in range(len(self.predictors)):
            cur_reward = self.get_reward_func_list[i](trajectory[1:-1], self.predictors[i], **kwargs)
            total_reward += self.weights[i] * cur_reward
        return total_reward

    def policy_gradient(self, data, n_batch=10, gamma=0.97,
                        std_smiles=False, grad_clipping=None, **kwargs):
        rl_loss = 0
        self.generator.optimizer.zero_grad()
        total_reward = 0
        for _ in range(n_batch):

            # Sampling new trajectory
            reward = 0
            trajectory = '<>'
            while reward == 0:
                trajectory = self.generator.evaluate(data)
                if std_smiles:
                    try:
                        mol = Chem.MolFromSmiles(trajectory[1:-1])
                        trajectory = '<' + Chem.MolToSmiles(mol) + '>'
                        reward = self.get_total_reward(trajectory, **kwargs)

                    except:
                        reward = 0
                else:
                    reward = self.get_total_reward(trajectory, **kwargs)

            # Converting string of characters into tensor
            trajectory_input = data.char_tensor(trajectory)
            discounted_reward = reward
            total_reward += reward

            # Initializing the generator's hidden state
            hidden = self.generator.init_hidden()
            if self.generator.has_cell:
                cell = self.generator.init_cell()
                hidden = (hidden, cell)
            if self.generator.has_stack:
                stack = self.generator.init_stack()
            else:
                stack = None

            # "Following" the trajectory and accumulating the loss
            for p in range(len(trajectory)-1):
                output, hidden, stack = self.generator(trajectory_input[p],
                                                       hidden,
                                                       stack)
                log_probs = F.log_softmax(output, dim=1)
                top_i = trajectory_input[p+1]
                rl_loss -= (log_probs[0, top_i]*discounted_reward)
                discounted_reward = discounted_reward * gamma

        # Doing backward pass and parameters update
        rl_loss = rl_loss / n_batch
        total_reward = total_reward / n_batch
        rl_loss.backward()
        if grad_clipping is not None:
            torch.nn.utils.clip_grad_norm_(self.generator.parameters(),
                                           grad_clipping)

        self.generator.optimizer.step()

        return total_reward, rl_loss.item()
```

7.2 Lexographic order method

Major changes to 7.1, from lines 41-57. Minor changes are omitted, instead, newly introduced variables are explained as comments.

```
for _ in range(n_batch):
    # Sampling new trajectory
    reward = 0
    trajectory = '<>'
    while reward == 0:
        trajectory = self.generator.evaluate(data)
        if std_smiles:
            try:
                mol = Chem.MolFromSmiles(trajectory[1:-1])
                trajectory = '<' + Chem.MolToSmiles(mol) + '>'
                if ith_property != 0:
                    dominate = True
                    for i in range(ith_property):
                        # prev_rewards is a list containing the previously most optimal rewards,
                        ↪   obtained by sampling from the the generator corresponding to that
                        ↪   property
                        prev_reward = self.prev_rewards[i]
                        # get_reward_func_list is a list of reward functions for all the properties,
                        ↪   passed in upon instantiation
                        cur_reward = self.get_reward_func_list[i](trajectory[1:-1],
                        ↪   self.predictors[i], **kwargs)
                        if cur_reward < prev_reward:
                            dominate = False
                    reward = self.get_reward_func_list[ith_property](trajectory[1:-1], self.predictors[ith_property], **kwargs) if dominate else 0
                else:
                    reward = self.get_reward_func_list[0](trajectory[1:-1], self.predictors[0], **kwargs)
            except:
                reward = 0
        else:
            if ith_property != 0:
                dominate = True
                for i in range(ith_property):
                    prev_reward = self.prev_rewards[i]
                    cur_reward = self.get_reward_func_list[i](trajectory[1:-1], self.predictors[i], **kwargs)
                    if cur_reward < prev_reward:
                        dominate = False
                reward = self.get_reward_func_list[ith_property](trajectory[1:-1], self.predictors[ith_property], **kwargs) if dominate else 0
            else:
                reward = self.get_reward_func_list[0](trajectory[1:-1], self.predictors[0], **kwargs)
```

Names of team members Haorun Li

Signatures of team members [illegible]

Name of the instructor Liting Sun

Signature of the instructor [illegible]

Date 06/21/2023