# Utilizing reinforcement learning for de novo drug design

(article starts on next page)

# Utilizing reinforcement learning for de novo drug design

Hampus Gummesson Svensson[1,2] · Christian Tyrchan[3] · Ola Engkvist[1,2] ·
Morteza Haghir Chehreghani[1]

© The Author(s) 2024

## Abstract
Deep learning-based approaches for generating novel drug molecules with specific properties have gained a lot of interest in the last few years. Recent studies have demonstrated promising performance for string-based generation of novel molecules utilizing reinforcement learning. In this paper, we develop a unified framework for using reinforcement learning for de novo drug design, wherein we systematically study various on- and off-policy reinforcement learning algorithms and replay buffers to learn an RNN-based policy to generate novel molecules predicted to be active against the dopamine receptor DRD2. Our findings suggest that it is advantageous to use at least both top-scoring and low-scoring molecules for updating the policy when structural diversity is essential. Using all generated molecules at an iteration seems to enhance performance stability for on-policy algorithms. In addition, when replaying high, intermediate, and low-scoring molecules, off-policy algorithms display the potential of improving the structural diversity and number of active molecules generated, but possibly at the cost of a longer exploration phase. Our work provides an open-source framework enabling researchers to investigate various reinforcement learning methods for de novo drug design.

**Keywords** De novo drug design · Reinforcement learning · Policy optimization · Replay
buffer · Recurrent neural network

## 1 Introduction

In recent years, there has been an increased interest in using machine learning for drug discovery. It has been applied to a large range of different tasks, including virtual screening, synthesis prediction, property prediction, and computer-assisted molecular design (Chen et al., 2018; Yang et al., 2019; Vamathevan et al., 2019). Machine learning has obtained an important position in de novo drug design — the design of novel chemical entities that fit certain constraints. De novo drug design is an iterative optimization problem whose navigation in the optimization landscape relies on finding local optima of molecular structures,

Extended author information available on the last page of the article

which does not necessarily lead to identifying the global optimum (Schneider & Fechner, 2005). Therefore, it is of interest to find a diverse set of local optima, meaning structurally different molecules with a high probability of being active against a desired target, i.e., with high activity.

Numerous deep learning-based methods have been developed for de novo drug design, including approaches based on reinforcement learning (Olivecrona et al., 2017; Zhou et al., 2019; You et al., 2018; Jin et al., 2020; Yang et al., 2021; Horwood & Noutahi, 2020; Gottipati et al., 2020; Neil et al., 2018) and variational autoencoders (Gómez-Bombarelli et al., 2018; Maus et al., 2022; Jin et al., 2018; Bradshaw et al., 2020). These approaches use several different ways to encode molecules into something that the model can learn, such as fingerprint-, string- and graph-based encodings. The string-based simplified molecular-input line-entry system (SMILES) (Weininger, 1988) is a popular way to encode the 2D structure of molecules. Previous work has demonstrated that graph-based and string-based generative models show equivalent performance in terms of chemical space coverage (Zhang et al., 2021). It is expected that the conclusions of this work are independent of how the molecules are represented. Moreover, recent evaluations of sample efficiency of de novo molecular generation methods have concluded good performance when using reinforcement learning (RL) for learning a recurrent neural network (RNN) (Rumelhart et al., 1985) to generate SMILES strings representing high-scoring molecules (Gao et al., 2022; Thomas et al., 2022). The objective is to learn a policy that can sample sequences of tokens to generate SMILES strings. Hence, policy optimization algorithms might have a significant impact on this task.

To further improve the sample efficiency of RL, it has been proposed to combine RL with a Hill-climb algorithm, which learns on the $k$ top-scoring sequences (Thomas et al., 2022; Neil et al., 2018; Brown et al., 2019). This method focuses the training on good samples from the current round of sequences. This can be interpreted as an off-policy algorithm with a replay buffer, filtering out low reward sequences and initializing the buffer memory between learning rounds.

The use of replay buffers is crucial in off-policy algorithms and is known to improve the sample efficiency of these algorithms (Fedus et al., 2020). However, to our knowledge, no previous work in de novo drug design has investigated off-policy algorithms with replay buffers utilizing past sequences. This work builds upon previous work in de novo drug design using a reward-based replay mechanism. Many of the state-of-the-art replay mechanisms used in reinforcement learning are often based on the temporal difference (TD) error (Schaul et al., 2015). However, since the TD error is not necessarily computable in a policy-based algorithm, a more general mechanism is needed for a fair comparison of algorithms. This paper focuses on reward-based replay mechanisms.

In this paper, we explore in a systematic way different on-policy and off-policy policy optimization reinforcement learning algorithms, in combination with several ways of replaying previous sequences or restricting the learning to a subset of the sequences sampled in the current episode. The objective is to investigate how large fraction of the generated molecules are predicted, with high probability, to be active to a desired target, and how structurally diverse these predicted active molecules are. Our work can be used as an open-source framework for researchers to investigate various RL methods for de novo drug design.[1]

---

[1] The source code of our framework is publicly available at: https://github.com/MolecularAI/SMILES-RL.
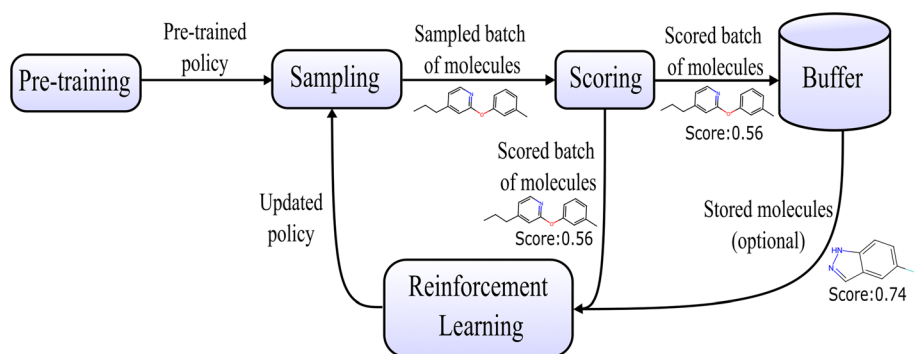
**Fig. 1** Schematic illustration of the de novo drug design process using reinforcement learning (RL)

## 2 Problem setup

The first step of de novo drug design using RL involves training a pre-trained policy (and/or encoding certain structures into the policy), as illustrated in Fig. 1. Subsequently, a batch of molecules is sampled by the policy, e.g., by the policy choosing a sequence of characters in a SMILES string. In the next step, the sampled molecules are scored by an unknown "black box" objective function, i.e., the objective function can be evaluated at any point of its domain but its full expression is unknown. The molecules and corresponding scores are both stored for final inspection and replay (optional). The current molecules and corresponding scores are also fed into the RL algorithm, where the molecular sampling policy is updated. Depending on the use of the replay buffer, current and/or previous samples are provided for the learning step. Using the updated policy, a new batch of molecules is sampled. This continues until a stopping criterion has been reached, such as a pre-defined budget of samples.

### 2.1 Problem definition

Molecular de novo design using RNNs for optimizing molecules encoded as SMILES strings, can be formulated as an RL problem. The agent interacts with the environment over discrete time steps by adding tokens to a SMILES string. The environment is episodic, where a SMILES string is provided between a start and stop token, and the episode's length depends on the SMILES string's length, which is terminated when the stop token is added to the string. At time step $t = 0$, the start token is added to the string and defines the first state $s_1$. At time step $t = 1, \ldots, T$ the agent observes a $n_s$-dimensional state vector $s_t \in \mathcal{S} \subseteq \mathbb{R}^{n_s}$, chooses an action $a_t \in \mathcal{A}$ according to a policy $\pi(a_t | s_t)$. The episode ends at time step $T + 1$, for terminal state $s_{T+1}$, when the stop token is chosen as action $a_T$. Moreover, at the end of the episode, a reward signal $R(a_{1:T}) \in [0, 1]$ is observed for a sequence of actions $a_{1:T}$. The scoring function provides this reward signal by scoring each valid SMILES string.

The state vector $s_t$ is given by the output states of the RNN at step $t - 1$ and encodes information about the actions taken in previous steps. A discrete action space $\mathcal{A} = \{0, \ldots, 33\}$ which tokenizes the feasible characters in the SMILES string, including
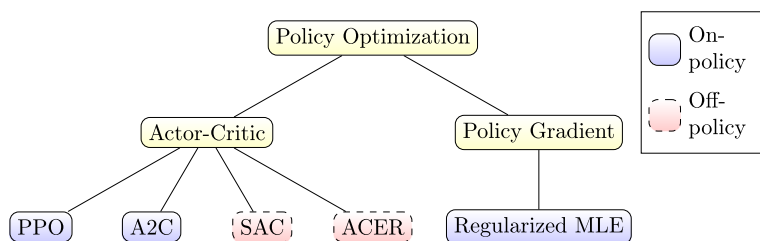
**Fig. 2** Taxonomy of the reinforcement learning (RL) algorithms explored in this work

start and stop tokens, is considered. Under this setup, we explore various policy optimization RL methods, where the goal is to learn the policy directly parameterized by $\theta$, $\pi_\theta(a_t|s_t)$. The output gates of the RNN are fed into a fully connected layer to provide either the probability (utilizing a softmax layer) or values of each action.

# 3 Policy optimization algorithms for de novo drug design

In this paper, we explore the following policy optimization algorithms for de novo drug design, to generate diverse molecules with high scores: (1) Regularized maximum likelihood estimation (Reg. MLE); (2) Advantage Actor-Critic (A2C); (3) Proximal Policy Optimization (PPO); (4) Actor-Critic with Experience Replay (ACER); (5) Soft Actor-Critic (SAC). Figure 2 illustrates the taxonomy of these algorithms. These are the major on- and off-policy policy optimization algorithms.

## 3.1 Regularized maximum likelihood estimation

The regularized maximum likelihood estimation (Reg. MLE) algorithm is currently used in REINVENT (Blaschke et al., 2020). Recent evaluations by both Gao et al. (2022) and Thomas et al. (2022) have concluded good performance compared to both RL-based and non-RL-based approaches for de novo drug design. The idea is that the likelihood of the agent's policy should stay close to that of the pre-trained policy (See Sect. 4.2) while still focusing on the high-scoring sequences. It minimizes the following policy loss

$$L^{\text{Reg. MLE}}(\theta) = \left(\log \pi_{\text{prior}}(a_{1:T}) + \sigma R(a_{1:T}) - \log \pi_\theta(a_{1:T})\right)^2, \tag{1}$$

where $\pi_{\text{prior}}(a_{1:T}) = \pi_{\text{prior}}(a_1|s_1) \cdots \pi_{\text{prior}}(a_T|s_T)$ is the likelihood of the pre-trained policy for a sequence of length $T$ (excluding start token), and $\pi_\theta(a_{1:T}) = \pi_\theta(a_1|s_1) \cdots \pi_\theta(a_T|s_T)$ is the corresponding likelihood of the policy that is optimized. The policy has the same network architecture and is initialized as the pre-trained policy. $\sigma$ is a hyperparameter (Blaschke et al., 2020) that determines the importance of the reward signal.

Note that it uses a margin guard, which resets the agent to the prior and adjusts sigma if the margin between the augmented likelihood and the agent likelihood $\log \pi_{\text{prior}}(a_{1:T}) + \sigma R(a_{1:T}) - \log \pi_\theta(a_{1:T})$ is below a pre-defined threshold. Table 1 in Appendix A displays a list of hyperparameters and the values employed in this paper.

## 3.2 Proximal policy optimization

Proximal Policy Optimization (PPO) (Schulman et al., 2017) uses a clipping loss function defined by

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t\big[\min\big(r(\theta)\hat{A}(s_t), \text{clip}\big(r_t(\theta), 1 - \epsilon, 1 + \epsilon\big)\hat{A}(s_t)\big)\big], \tag{2}$$

where $r(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio and advantage $\hat{A}(s_t) = \gamma^{T-t}R(a_{1:T}) - V_\phi(s_t)$ is used, where $V_\phi(s_t)$ is the value function. This corresponds to Monte-Carlo (MC) samples where the reward $R(a_{1:T})$ is only given at time step $T$. $\gamma$ is the discount factor and $\epsilon$ is a hyperparameter determining the clipping range. We adapt PPO to the setting where the state vectors are represented by the RNN outputs of the latest recurrent node. The actor is a neural network with parameters $\theta$, providing probabilities $\pi_\theta(a|s)$ for an action $a$ at state vector $s$ using a softmax layer. It is initialized as the pre-trained policy (see 4.2) and the parameters are updated using the loss function in Eq. (2). The value function $V_\phi(s_t)$ is a neural network with (non-shared) parameters $\phi$. It has the same network architecture as the actor, but the output layer only consists of one output, i.e., the value, and uses no softmax layer. The output layer is reset at initialization, while the initial embedding and long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997) layer are the same as the pre-trained policy network. The value network is trained by minimizing the following mean squared error loss

$$L^{\text{MSE}}(\phi) = \mathbb{E}_t\Big[\frac{1}{2}\big(\gamma^{T-t}R(a_{1:T}) - V_\phi(s_t)\big)^2\Big]. \tag{3}$$

For each batch of sequences, the actor and critic loss are minimized over 4 epochs, each doing minibatch updates where sequences are shuffled into 4 mini-batches.

It is possible for the actor to diverge from the pre-trained actor (policy), which has learned how to sample a valid SMILES string. This rarely happens but should be properly handled when happening. Therefore, if the fraction of valid SMILES strings (out of the 128 sampled in each episode) is less than 0.8 for more than 10 consecutive episodes, the parameters of the algorithm will be reset to that of the pre-trained model. Table 2 in Appendix A displays a list of hyperparameters and the values employed in this paper.

## 3.3 Advantage actor-critic

Advantage Actor-Critic (A2C) is a synchronous version of the A3C algorithm (Mnih et al., 2016), wherein the following definition of the advantage is used

$$\hat{A}(s_t) = \gamma^{T-t}R(a_{1:T}) - V_\phi(s_t), \tag{4}$$

where $V_\phi(s_t)$ is the value network with (non-shared) parameters $\phi$ and discount factor $\gamma$. We adapt A2C to the setting where the state vectors are represented by the RNN outputs of the latest recurrent node. The actor is a neural network with parameters $\theta$, providing probabilities $\pi_\theta(a|s)$ for an action $a$ and state vector $s$ using a softmax layer. It is initialized as the pre-trained policy (see 4.2) and the parameters are updated using the policy gradient with the advantage. A discount factor slightly smaller than 1 is used to slightly favor small molecules. The value network has the same network architecture as the actor, but the output

layer only consists of one output, i.e., the value, and uses no softmax layer. The output layer is reset at initialization, while the initial embedding and LSTM layer are the same as the pre-trained policy network. The value network is trained by minimizing the following mean squared error loss

$$L^{\text{MSE}}(\phi) = \mathbb{E}_t \left[ \frac{1}{2} \left( \gamma^{T-t} R(a_{1:T}) - V_\phi(s_t) \right)^2 \right]. \tag{5}$$

If the actor generates a large fraction of invalid SMILES strings, the algorithm is reset in the same way as for PPO (see Sect. 3.2). Table 3 in Appendix A displays a list of hyperparameters and the values employed in this paper.

## 3.4 Actor-critic with experience replay

Actor-Critic with Experience Replay (ACER) (Wang et al., 2016) is an off-policy actor-critic algorithm with experience replay. ACER is the off-policy counterpart of the A3C algorithm (Mnih et al., 2016) where the aim is to stabilize the off-policy estimator, e.g, by applying a trust region policy optimization (TRPO) method. The algorithm performs one on-policy update and $r \sim \text{Pois}(\lambda)$ off-policy updates using replay, where each replay samples 128 sequences. We adapt ACER to the setting where the state vectors are represented by the RNN outputs of the latest recurrent node. It uses a shared network utilizing the same architecture as the pre-trained policy network (see Sect. 4.2) but with an additional value head, i.e., a parallel fully connected layer with output dimension 1. The value head is randomly initialized, while the other weights are the same as the pre-trained policy network, including the policy head. We add an entropy term to the loss with weight 0.001, slightly favoring sequences with larger cumulative entropy.

We use retrace Q-value estimation, as proposed in the original algorithm (Wang et al., 2016). Using retrace Q-value estimations, instead of Monte-Carlo samples, does slightly improve the stability. For each sequence, $a_{1:T}$, reward $R(a_{1:T})$ is only given at time step $T$ for action $a_T$, when the stop token is chosen as action. A reward signal of $-1$ is given to invalid SMILES strings, which are by default given a reward of 0 for the on-policy algorithms. The penalty for invalid SMILES bias the algorithm strongly toward valid SMILES. This seems to be crucial, especially when performing many off-policy updates using the replay memory. Furthermore, 10 initial steps without updating the policy are performed, only using the pre-trained policy to store initial sequences in the replay memory. Table 4 in Appendix A displays a list of hyperparameters and the values employed in this paper.

## 3.5 Soft actor-critic

**Algorithm 1** Discrete Soft Actor-Critic for de novo drug design

---

**Input:** $\phi$, $\theta$, initial episodes $K_{\text{init}}$, total budget of episodes $K_{\text{E}}$,
**Init:** $\phi' \leftarrow \phi$, $\theta' \leftarrow \theta$, $\mathcal{D} \leftarrow \emptyset$

1: **for** each initial episode $1, \ldots, K_{\text{init}}$ **do**
2:      Sample a batch $\mathcal{T}$ of $M$ sequences using pre-trained policy $\pi_\theta$
3:      Score each sequence in $\mathcal{T}$
4:      Add unique, valid sequences to replay memory $\mathcal{D}$
5: **end for**
6: **for** each episode $K_{\text{init}} + 1, \ldots, K_{\text{E}}$ **do**
7:      Sample a batch $\mathcal{T}$ of $M$ sequences using current policy $\pi_\theta$
8:      Score each sequence in $\mathcal{T}$
9:      Add unique, valid sequences to replay memory $\mathcal{D}$
10:      $\phi \leftarrow \phi - \lambda_Q \hat{\nabla}_\phi J_Q(\phi|\mathcal{T})$   ▷ On-policy update of Q-function parameters
11:      $\theta \leftarrow \theta - \lambda_\pi \hat{\nabla}_\theta J_\pi(\theta|\mathcal{T})$        ▷ On-policy update of policy parameters
12:      $\alpha \leftarrow \alpha - \lambda_\alpha \hat{\nabla}_\alpha J_\alpha(\alpha|\mathcal{T})$          ▷ On-policy update of temperature
13:      $\phi' \leftarrow \tau\phi' + (1-\tau)\phi$               ▷ Update target parameters
14:      $\theta' \leftarrow \tau\theta' + (1-\tau)\theta$         ▷ Update average policy parameters
15:      **for** each off-policy update **do**
16:          $\phi \leftarrow \phi - \lambda_Q \hat{\nabla}_\phi J_Q(\phi|\mathcal{D})$
17:          $\theta \leftarrow \theta - \lambda_\pi \hat{\nabla}_\theta J_\pi(\theta|\mathcal{D})$
18:          $\alpha \leftarrow \alpha - \lambda_\alpha \hat{\nabla}_\alpha J_a(\alpha|\mathcal{D})$
19:          $\phi' \leftarrow \tau\phi' + (1-\tau)\phi$
20:          $\theta' \leftarrow \tau\theta' + (1-\tau)\theta$
21:      **end for**
22: **end for**

---

Soft Actor-Critic (SAC) (Haarnoja et al., 2018) is an off-policy algorithm that incorporates the entropy of the policy into the reward signal to encourage a stochastic policy with more randomness, while still fulfilling the task. It is based on the maximum entropy objective, with the aim of optimal policy $\pi^*$ to maximize both its reward and entropy at each visited state.

It uses an automatic entropy adjustment to control the temperature parameter $\alpha$ that determines the relative importance of the entropy term. The original algorithm is formulated for a continuous action space, but Christodoulou (2019) has extended it to discrete action spaces. The soft actor-critic algorithm for discrete action spaces is utilized in this work, with some adaptions discussed below. We adapt SAC to the setting where the state vectors are represented by the RNN outputs of the latest recurrent node. The actor is a neural network with parameters $\theta$, providing probabilities $\pi_\theta(a|s)$ for an action $a$ and state vector $s$ using a softmax layer. It is initialized as the pre-trained policy (see 4.2) and the parameters are updated by minimizing the loss in Eq. (10). The value function $Q_\phi(a,s)$ is given by a neural network that has the same network architecture as the actor, where each output corresponds to the action-state value of an action $a$ at current state $s$ but uses no softmax

layer. We initialize the parameters $\phi$ to that of the parameters of the pre-trained policy network. We perform no updates of parameters during the $K_{\text{init}} = 10$ first episodes, where only experiences are sampled to the replay buffer. Since a budget of 2000 episodes is considered in the following experiments, there are 1990 episodes left for learning to generate high-scoring molecules. When updating parameters, we utilize a reward of $-1$ for invalid SMILES. Moreover, we perform one on-policy update, using all current sequences, before doing any off-policy update. Four off-policy updates are performed, where 64 sequences are sampled from the replay memory, for each episode. We observed no significant difference in performance when using the larger replay size of 128 as for ACER (seer Sect. 3.4).

The following entropy-augmented reward is defined

$$r_\pi(s_t, a_t) \triangleq r_{(s_t, a_t)} + \mathbb{E}_{s_{t+1} \sim p}\big[\alpha \mathcal{H}\big(\pi_\theta(\cdot | s_{t+1})\big)\big], \tag{6}$$

where $p$ is the state transition probability of the environment. Full sequences are used for updating, i.e., Monte-Carlo (MC) samples, instead of one-step update in Haarnoja et al. (2018), Christodoulou (2019). The reward signal $R(a_{1:T})$ is given at the end of the episode, at time step $T$, and uses no discount, i.e., discount factor $\gamma = 1$ is utilized. This gives the following target for each soft Q-value update

$$y_{s_i} = R(a_{1:T}) + \sum_{l=i}^{T-2} \mathbb{E}_{s_{l+1} \sim p}\big[\alpha \mathcal{H}\big(\pi_\theta(\cdot | s_{l+1})\big)\big], \tag{7}$$

where $y_{s_{T+1}} = 0$, and $y_{s_{T-1}} = y_{s_T} = R(a_{1:T})$, since it is defined that $\mathcal{H}\big(\pi_\theta(\cdot | s_{T+1})\big) = \mathcal{H}\big(\pi_\theta(\cdot | s_T)\big) = 0$ to keep the pre-trained high probabilities of stop tokens at certain states. This gives each action an equal contribution from the reward signal of the full sequence and an additional cumulative entropy term that favors actions where future states have high entropy.

To improve the stability, Zhou et al. (2022) has proposed to include more regularization in the actor and critic losses. The proposed clipping of the critic loss with target critic(s) is used, which is found to improve the stability, giving the following loss of the critic network

$$J_Q(\phi | \Pi) = \mathbb{E}_{\mathcal{T} \sim \Pi}\Big[\mathbb{E}_{(a_t, s_t) \sim \mathcal{T}}\Big[\max\Big(\big(Q_\phi(a_t, s_t) - y_{s_t}\big)^2,$$
$$\big(Q_{\phi'}(a_t, s_t) + \text{clip}\big(Q_\phi(a_t, s_t) - Q_{\phi'}(a_t, s_t), -c, c\big) - y_{s_t}\big)^2\Big)\Big]\Big], \tag{8}$$

where $\Pi$ is a set of sequences (either current sequences or replay memory), $\mathcal{T}$ is a sequence, $Q_\phi$ is the estimate of the critic network, and $Q_{\phi'}$ is the estimate of the target-critic network. When $\Pi$ contains the current sequences, all sequences are sampled; otherwise, 64 sequences are sampled when using a full replay memory. The weights of the target-critic are updated using the moving average between the current weights of the target-critic and critic,

$$\phi' \leftarrow \tau\phi' + (1 - \tau)\phi, \tag{9}$$

where $\tau$ is the smoothing coefficient, determining how much of the updated critic network that will be transferred to the target-critic network. We only use one critic and target-critic network, since we observe that using the pre-trained model as the initial critic network is advantageous, instead of using randomly initialized weights. This yields more stable learning in terms of the validity of generated SMILES strings and biases the learning towards

what the pre-trained model knows. It could possibly be useful to use two critic networks if their updates utilize different replay experiences.

For regularizing the actor loss, we include the Kullback–Leibler (KL) divergence term between the (current) actor and the average policy network. The average policy network $\pi_{\theta'}$ is initialized as the actor and whose parameters $\theta'$ is updated using moving average (as in Eq. (9)) with $\tau = 0.99$. Adding this KL divergence term yields the following actor loss

$$
\begin{aligned}
J_\pi(\theta|\Pi) = \mathbb{E}_{\mathcal{T}\sim\Pi}\big[\mathbb{E}_{s_t\sim\mathcal{T}}\big[\pi_\theta(\cdot|s_t)\big(\alpha \log\big(\pi_\theta(\cdot|s_t)\big) - Q_\phi(\cdot,s_t)\big) \\
+ D_{\mathrm{KL}}\big(\pi_\theta(\cdot|s_t)\|\pi_{\theta'}(\cdot|s_t)\big)\big]\big].
\end{aligned}
\tag{10}
$$

The temperature $\alpha$ is updated by minimizing the following objective, extending the objective proposed by Christodoulou (2019),

$$
J_\alpha(\alpha|\Pi) = \mathbb{E}_{\mathcal{T}\sim\Pi}\big[\mathbb{E}_{(a_t,s_t)\sim\mathcal{T}}\big[-\alpha\big(\log\pi_\theta(a_t|s_t) + \bar{H}\big)\big]\big],
\tag{11}
$$

where $\bar{H}$ is the target entropy. Algorithm 1 illustrates all steps in the (discrete) soft actor-critic algorithm used for de novo design. Table 5 in Appendix A displays a list of hyperparameters and the values employed in this paper.

# 4 De novo drug design using RNNs for generating SMILES generation

This section describes the experimental setup, including the replay buffers used to investigate the on- and off-policy policy optimization algorithms. A batch of $M = 128$ sequences is sampled in each episode, generating 128 SMILES strings in each episode. Any duplicates of SMILES strings are removed afterward, possibly yielding a list of fewer unique molecules. At the end of each episode, using on-policy or/and off-policy batches, the policies are updated by the full roll-out of each sequence, as described in Sect. 3. Technical details of the experiments are discussed in Appendix B.

## 4.1 Molecular and topological scaffolds

The scaffold of a molecule is defined as its core structure. This is a common structure characterizing a group of molecules. This provides a basis for a systematic investigation of molecular cores and building blocks. This assists in finding structurally distinct molecules having similar activity, providing several structural alternatives when optimizing the properties of potential drug candidates (Hu et al., 2016). Hence, scaffolds provide a diversity measure of the identified active molecules.

The molecular scaffold defined by Bemis and Murcko (1996), also known as the Bemis-Murcko scaffold, is used in this work to generate scaffolds. The topological scaffold is defined as the generic molecular scaffold where all atom types are converted into carbon atoms and all bonds are converted into single bonds, as illustrated in Fig. 3. All scaffolds in this work are generated using (Landrum, 2006).

## 4.2 Sampling

We use the pre-trained policy network provided by Blaschke et al. (2020). The model is trained on a data set derived from the ChEMBL databse (Gaulton et al., 2012) and is
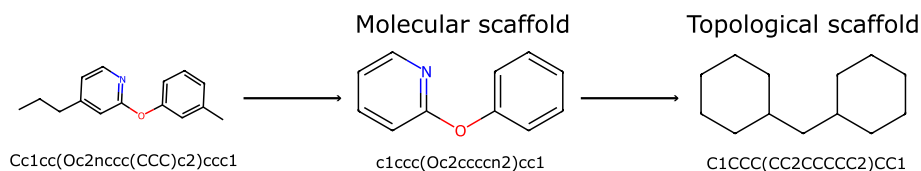
Molecular scaffold Topological scaffold



Cc1cc(Oc2nccc(CCC)c2)ccc1          c1ccc(Oc2ccccn2)cc1          C1CCC(CC2CCCCC2)CC1

**Fig. 3** The structural formula and SMILES strings for an arbitrary molecule, and its corresponding molecular and topological scaffold

capable of generating molecules in terms of SMILES strings. The parameters of the policy network are sequentially updated using reinforcement learning (see Sect. 3). The policy consists of an embedding layer, an LSTM layer, and a fully connected output layer. The embedding layer consists of 256-dimensional embedding vectors. The LSTM layer has an input size of 256 and an output size of 512 and consists of 3 recurrent layers, i.e., three LSTMs stacked together. The LSTM output is fed to an output layer of output dimension 34. Each output entity corresponds to a token in the vocabulary, including start and stop tokens, defined by Blaschke et al. (2020).

When sampling a sequence, the output layer is fed through a softmax function to obtain estimates of the probabilities of each token (action). Multimodal sampling, using the estimated probabilities from the softmax function, is performed to select the next action in a sequence. Small molecules are of interest and, therefore, the length of a sequence is limited to 256. If a sequence reaches this length, the sampling is stopped, returning the sequence sampled so far.

### 4.3 Scoring

A scoring function provides the rewards signal $R(a_{1:T})$ of each sequence. The scoring function is given by a random forest model with 1300 trees and a maximum depth of 300 for each tree. Class weights, which determine the sample probability during bootstrapping, are inversely proportional to the class frequencies. The random forest model is trained to predict the binary activity of a molecule to the Dopamine receptor D2 (DRD2), using the activity data in ExCAPE-DB (Sun et al., 2017). 2048-bit Morgan-like fingerprints, computed by RDKit (Landrum, 2006), with radius 2, utilizing features and counts are used as feature vectors. Each SMILES string is encoded into a such feature vector for scoring. Class probabilities, of the binary activity, are given by the fraction of trees predicting the corresponding class. The reward of a sequence is defined as the probability of predicting a positive label for the corresponding sequence. A sequence (or SMILES string) is defined to be valid if the corresponding SMILES string can be constructed into a Mol object by RDKit (Landrum, 2006), which is done when computing the fingerprints for scoring. When constructing a Mol object, RDKit first performs a grammar check and then applies basic chemical rules. An invalid SMILES string is given a reward of 0 and −1 for the on- and off-policy algorithms, respectively. The score can also be modified by the diversity filter, see Sect. 4.4.

A molecule is defined to be active if the corresponding sequence has a reward greater than or equal to 0.7, and a scaffold is defined as active if it contains at least one active molecule. These definitions are used to compare the policy optimization algorithm utilizing different replay buffers.

## 4.4 Diversity filter

A diversity filter is a memory-assisted approach to improve the diversity of generated molecules. It keeps track of the molecules with similar structures, e.g., scaffolds. We use the diversity filter based on identical molecular scaffolds, proposed by Blaschke et al. (2020). This diversity filter consists of a scaffold memory that stores molecules and their corresponding molecular scaffold. A molecule is saved into the scaffold memory if the corresponding sequence reaches at least a reward of 0.4. No molecules with the same canonical SMILES are allowed in the scaffold memory, hence only storing unique molecules. If the number of saved molecules with the same molecular scaffold reaches 25, all future molecules with the same molecular scaffold are given a reward of 0, and consequently not saved in the scaffold memory. This changes the reward function used for learning when a certain number of molecules with the same molecular scaffold have been generated.

## 4.5 Replay buffers

In light of the current use of the Hill-climb algorithm (Thomas et al., 2022; Neil et al., 2018; Brown et al., 2019) for training, we study different approaches using both current and previous sequences. All of these approaches are collected under the term *replay buffers*.

In each episode, a batch of $M = 128$ sequences are sampled. For on-policy algorithms with replay buffers considering historical data, the current batch of sequences plus $k = 64$ sequences from history, not including the current sequences, are used for learning. Moreover, for replay buffers only using current data, $k = 64$ sequences from the current batch of sequences are used for learning, except for *All current* where the entire current batch is used for training. For the off-policy algorithms, only replay buffers using historical data are considered, since they are defined to always do one on-policy update with the current sequences. For each off-policy update, SAC and ACER replay $k = 64$ and $k = 128$ sequences, respectively, from the buffer. Opposite of what is done for the on-policy algorithms, the sequences of the current episode are immediately stored in the replay memory, i.e., sequences from the current batch can be sampled from the replay memory when performing off-policy updates in the current episode. Below follow descriptions of each replay buffer investigated in this paper, seven in total.

**All current (AC)**
For the *All current* (AC) replay buffer, the entire batch of sampled sequences in the current episode is used during learning. No sequences from previous episodes are utilized. In practice, this corresponds to performing a full on-policy update.

**Bin history (BH)**
The *Bin history* (BH)replay buffer sorts sequences into bins with respect to their reward. It consists of the following fixed binds with respect to rewards: $[0, 0.1], (0.1, 0.2], \dots, (0.9, 1]$. For the off-policy algorithms, the buffer also includes a bin storing invalid SMILES strings, i.e., SMILES string with a score $-1$. Each bin has a maximum size of 1000 sequences. First in, first out (FIFO) is applied if the bin is full. To the extent possible, it does, without replacement, sample an equal number of sequences from each bin and otherwise uniformly samples, without replacement, from the bins with elements that have not been sampled until $k$ sequences have been acquired.

**Bin current (BC)**

*Bin current* (BC) replay buffer sorts the current batch of sampled sequences into bins with respect to their rewards. It consists of the following fixed binds with respect to rewards: $[0, 0.1], (0.1, 0.2], \dots, (0.9, 1]$. It does, to the extent possible, sample an equal number of sequences from each bin without replacement. If $k$ sequences have not been acquired after this, it uniformly samples from the bins with unsampled sequences (i.e., bins whose sequences are not yet in the set used for update) until $k$ sequences have been sampled in total.

**Top-Bottom history (TBH)**

When using *Top-Bottom history* (TBH) replay buffer with an on-policy algorithm, the previous $k/2$ highest and $k/2$ lowest rewarding sequences, and the current sequences, are used for updating the policy. It prioritizes storing the newest sequence(s) if several sequences have an equal reward. No duplicates with the same canonical SMILES string are allowed, keeping the newest sequence with the lowest reward.

For off-policy algorithms, it consists of three sub-buffers, each with a memory size of 1000 sequences. It consists of one sub-buffer with the highest rewarding sequences and two sub-buffers with low rewarding sequences. One of these low-reward sub-buffers stores only sequences with $-1$ reward, i.e., invalid SMILES strings, and the other one stores the lowest-scoring sequences that correspond to a valid molecule (having a reward of at least zero). It uniformly samples, without replacement, from a buffer consisting of all three sub-buffers. FIFO is utilized for each sub-buffer, where the oldest sequences are removed when a sub-buffer is full. No duplicates with the same canonical SMILES string are allowed, where the newest sequence with the lowest score is kept.

**Top-Bottom current (TBC)**

For the *Top-Bottom current* (TBC) replay buffer, $k/2$ highest and $k/2$ lowest rewarding sequences of the current batch are used for the update.

**Top history (TH)**

When using the *Top history* (TH) replay buffer with an on-policy algorithm, the top-$k$ rewarding sequences from previous episodes and sequences from the current episode are used for updating the actor and critic. Hence, it only needs to store the top-$k$ sequences from previous episodes. Any duplicate with the same canonical SMILES string as another stored sequence is removed.

For off-policy algorithms, it consists of a buffer with the highest rewarding sequences. It has a memory size of 1000 sequences and does not allow any sequences with the same canonical SMILES string, keeping the newest sequence with the lowest score. It uniformly samples $k$ sequences, without replacement, from the buffer to use for the update of the parameters.

**Top current (TC)**

For the *Top current* (TC) replay buffer, the top-$k$ rewarding sequences of the current batch is utilized for updating. This is similar to what is utilized in the Hill-climb algorithm.

# 5 Results

In this section, we investigate various policy optimization algorithms, described in Sect. 3, for the de novo drug design setup defined in Sect. 4. To compare all algorithms under the same budget constraint, the generation is limited to 2000 episodes, giving a budget
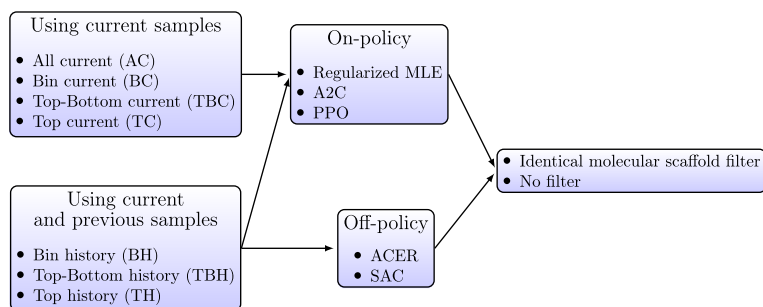
**Fig. 4** Illustration of the different combinations of replay buffer, policy optimization algorithm and diversity filter that are investigated in this paper

of 256 000 possible SMILES strings in total. We investigate both the use of the diversity filter based on identical molecular scaffolds (see Sect. 4.4) and the use of no diversity filter. The results are divided into on- and off-policy algorithms. Figure 4 displays the different combinations of replay buffer, policy optimization algorithm and diversity filter that are investigated. Figures 13, 14, 15, 16, 17, 18, 19, 20, 21, 22 in Appendix C display the ten top-scoring molecules, each from a unique topological scaffold, of one representative run for each combination. The generated molecules look like what has been published in previous work targeting DRD2 (Olivecrona et al., 2017; Thomas et al., 2022).

## 5.1 On-policy algorithms

Firstly, we study de novo drug design utilizing on-policy algorithms. We investigate the algorithms A2C, Regularized MLE and PPO, when either using the identical molecular scaffold diversity filter or no diversity filter.

### 5.1.1 With diversity filter

Figure 5 shows box plots of the number of (unique) active molecules and active scaffolds over 11 runs for the on-policy algorithms utilizing different replay buffers, and identical molecular scaffolds diversity filter. As illustrated in Fig. 5a, for most replay buffers, both A2C and PPO sample a significantly larger number of active molecules compared to Regularized MLE; whereas there is relatively little difference between A2C and PPO in terms of the number of active molecules generated. Regularized MLE only generates notably more unique active molecules when using the *top current* replay buffer, compared to A2C and PPO using the same replay buffer. When using PPO and A2C, the replay buffers *All current* and *Bin current* yield the largest number of active molecules. Moreover, when using Regularized MLE, utilizing *All current* generates the largest number, whereas *Top current* is the second-best replay buffer.

As seen in Fig. 5b, Regularized MLE utilizing *All current* show the largest median of (unique) active scaffolds with low variability, compared to A2C and PPO. This combination also shows the largest median of generated (unique) active topological scaffolds, as seen in Fig. 5c. Regularized MLE generates a significantly larger number of active
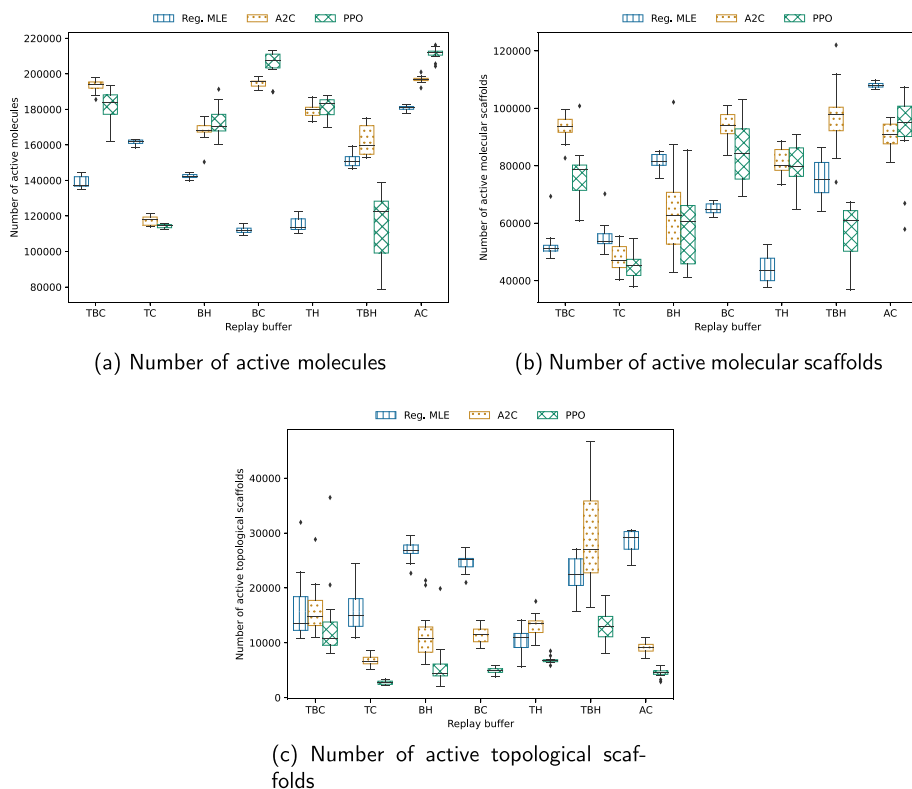
(a) Number of active molecules



(b) Number of active molecular scaffolds



(c) Number of active topological scaffolds

**Fig. 5** Box plots of the number of unique active molecules and active scaffolds for the on-policy algorithms A2C, Regularized MLE, and PPO (higher is better) when utilizing identical molecular scaffold filter. It shows mean and standard deviation over 11 runs for each policy and replay buffer. 128 SMILES strings are sampled in each episode, with a budget of 2000 episodes in total

topological scaffolds with most replay buffers, compared to A2C and PPO. However, A2C with *Top-Bottom history* performs on par, in terms of active scaffolds, with Regularized MLE but has a larger variability. In fact, A2C with *Top-Bottom history* displays the best runs in terms of the number of active scaffolds. Moreover, in most cases, A2C generates a significantly larger number of topological scaffolds than PPO; while comparable numbers of active molecular scaffolds are often generated.

Figure 6 displays means and standard deviations, over 11 repeated runs, of the average episodic rewards of sampled molecules for all combinations of on-policy algorithms and replay buffers over 2000 episodes of batch size 128. The average episodic rewards are displayed using a moving average with a window size of 50. All runs use the identical molecular scaffold filter. For both PPO and A2C, *Top current* gives the lowest average episodic reward, as shown in Fig. 6a and b, respectively. Furthermore, PPO utilizing *Bin current* and *All current* performs on par, giving moving averages between 0.8 and 0.7 after roughly 125 episodes. It seems that most replay buffers converge after around 125 episodes. Moreover, for A2C, *All current*, *Top-Bottom current* and *Bin current* perform among the best in terms of the average episodic reward, when using a diversity filter. Compared to PPO, it takes slightly more episodes for the rewards to converge. For Regularized MLE, utilizing *All*
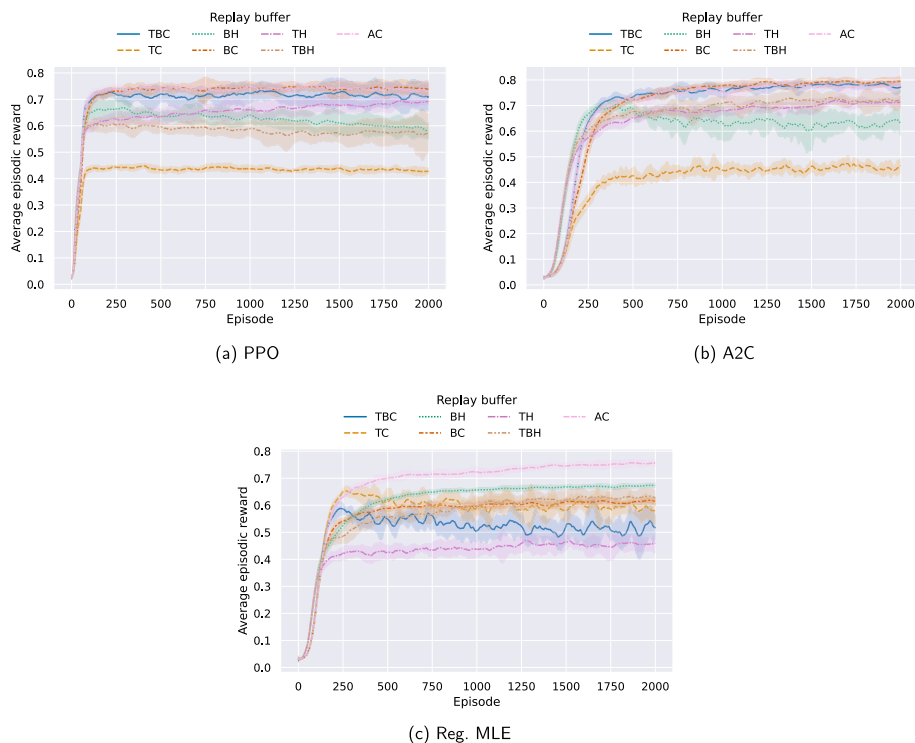
**Fig. 6** Average episodic reward computed over a batch of sequences, for the on-policy algorithms A2C, Regularized MLE and PPO (higher is better) when utilizing identical molecular scaffold filter. It shows mean and standard deviation of moving average, of window size 50, over 11 runs for each policy and replay buffer. 128 SMILES strings are sampled in each episode, with a budget of 2000 episodes in total

*current* gives the largest average episodic reward, converging to a reward between 0.7 and 0.8 with low variance. It is the only replay buffer for Regularized MLE that can reach an average episodic reward above 0.7. Utilizing *Top history* yields the lowest episodic reward.

### 5.1.2 Without diversity filter

Figure 7 shows boxplots of the number of (unique) active molecules and active scaffolds over 11 runs for the on-policy algorithms utilizing different replay buffers. No diversity filter is used. In this setting, Regularized MLE using either *Bin history*, *Bin current* or *All current* consistently generates a larger number of active molecules, molecular scaffolds and topological scaffolds, compared to all other combinations of on-policy algorithm and replay buffer. *Bin current* yields the largest number for all these metrics. For all replay buffers, Regularized MLE gives the largest number of active molecules and scaffolds, while A2C gives a higher number than PPO.

Figure 8 displays means and standard deviations, over 11 repeated runs, of the average episodic rewards of sampled molecules for all combinations of on-policy algorithms and replay buffers using no diversity filter. A budget of 2000 episodes of batch size 128 is investigated. For visualization purposes, the average episodic rewards correspond to
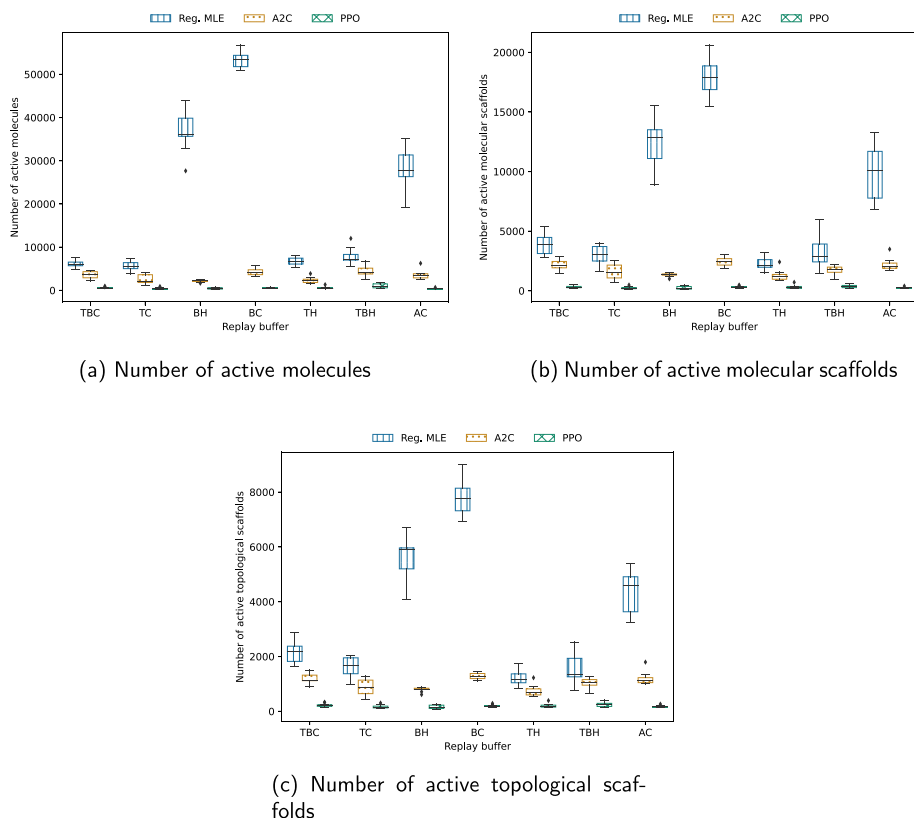
(a) Number of active molecules

(b) Number of active molecular scaffolds



(c) Number of active topological scaffolds

**Fig. 7** Box plots of the number of unique active molecules and active scaffolds for the on-policy algorithms A2C, Regularized MLE and PPO (higher is better) when utilizing no diversity filter. 128 SMILES strings are sampled in each episode, with a budget of 2000 episodes in total

moving averages using a window size of 50. PPO seems to require the least number of episodes to converge but, on the other hand, shows a higher variance.

As seen in Fig. 8a, PPO with *Top-Bottom current*, *All current* and *Top-Bottom history* reaches an average episodic reward 1 approximately. Using *Top-bottom history* shows a substantially lower episodic reward but has a larger variance.

For A2C, displayed in Fig. 8b, using replay buffers with data samples from previous episodes, except for *Top current*, gives slightly lower episodic reward compared to using only data sampled in the current episode. All replay buffers only using immediate samples give an average episodic reward close to 1. One should note that the short drop of reward for *All current* occurs due to the resetting of the parameters of one run to that of the pretrained model, since the policy samples less than 80% valid molecules for more than 10 consecutive episodes. After restarting, it quickly gets back on track. This run is kept to highlight that it is possible for the networks to diverge from the pre-trained model and forget how to generate valid SMILES strings. When this happens, it can quickly find its way back by restarting from the pre-trained model.

For Regularized MLE, there is generally a lower variance for the episodic reward compared to PPO and A2C. When using either *Top-Bottom history*, *Top current* or *Top history*,
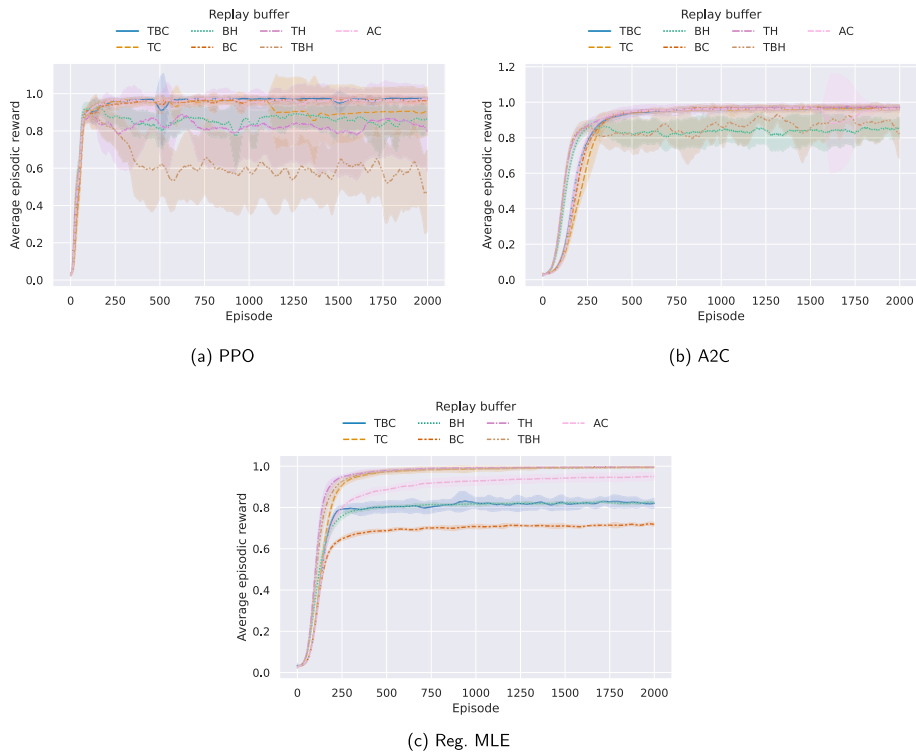
(a) PPO

(b) A2C

(c) Reg. MLE

**Fig. 8** Average episodic reward computed over a batch of sequences, for the on-policy algorithms A2C, Regularized MLE and PPO (higher is better) when utilizing no diversity filter. It shows mean and standard deviation of moving average, of window size 50, over 11 runs for each policy and replay buffer. 128 SMILES strings are sampled in each episode, with a budget of 2000 episodes in total

the average episodic reward converges to 1, as illustrated in Fig. 8c. Note that neither PPO nor A2C consistently reaches such high episodic reward. Also, for regularized MLE, *All current* is not among the best ones, which is the case for both PPO and A2C.

## 5.2 Off-policy algorithms

To further investigate the benefits of a replay buffer, two off-policy algorithms have been explored: (1) Soft Actor-Critic (SAC); (2) Actor-Critic with Experience replay (ACER).

The off-policy algorithms perform one step of on-policy update (using all sampled data in the current episode) and several off-policy updates using sequences from both current and previous episodes. The off-policy updates use replays from either *Bin history*, *Top history* or *Top-Bottom history*. Opposite to the on-policy algorithms, the sequences of the current episode are stored in the replay memory before use in the current episode, i.e., it is possible to utilize sequences from the current batch for off-policy updates in the current episode.
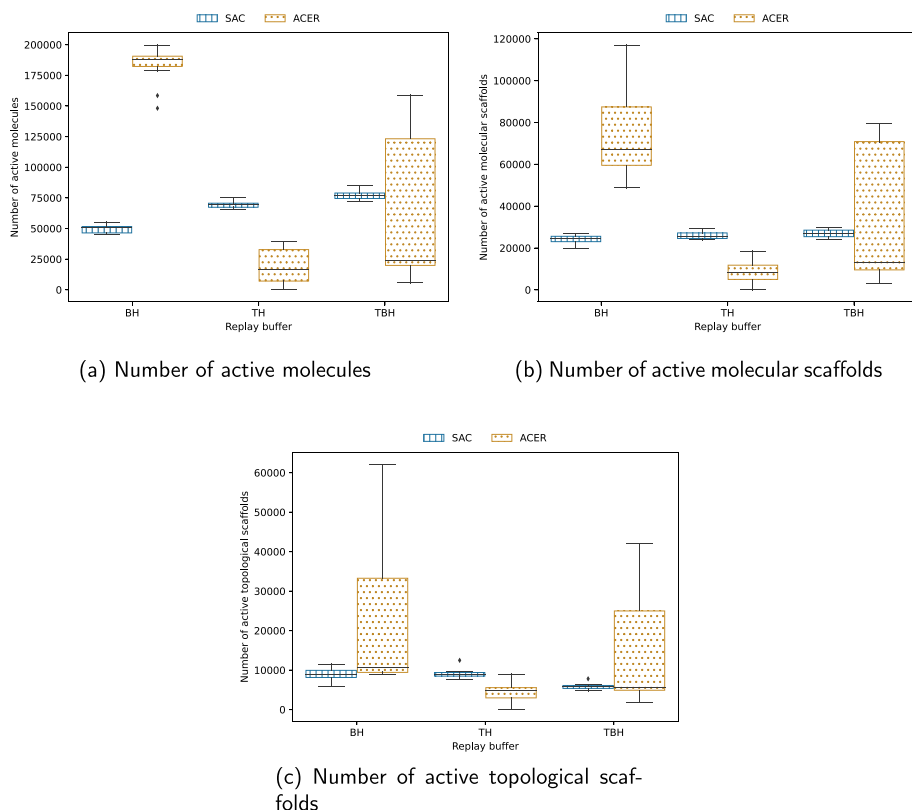
(a) Number of active molecules



(b) Number of active molecular scaffolds



(c) Number of active topological scaffolds

**Fig. 9** Box plots of the number of unique active molecules and active scaffolds for the off-policy algorithms ACER and SAC (higher is better) when utilizing identical molecular scaffold filter. The box plot is computed over 11 repeated runs for each combination of policy and replay buffer. 128 SMILES strings are sampled in each episode, with a budget of 2000 episodes in total

### 5.2.1 With diversity filter

Figure 9 displays box plots of the number of active molecules and scaffolds for 11 repeated runs of each combination of off-policy algorithm and replay buffer. 128 molecules are generated in each episode, not necessarily valid and/or unique molecules, with a total budget of 2000 episodes. ACER using *Bin history* generates the largest number of active molecules and scaffold. It is able to yield numbers close to or better than the best on-policy results in Sect. 5.1.1, which the other off-policy combinations are not able to. However, in general, ACER shows a larger variability compared to SAC, where ACER using *Top-Bottom history* shows the largest variability over the repeated runs.

Figure 10 shows means and standard deviations, over 11 repeated runs, of the average episodic rewards of sampled molecules for all combinations of off-policy algorithms and replay buffers using the identical molecular scaffold filter. A budget of 2000 episodes, each sampling a batch of 128 molecules, is investigated. For visualization purposes, each average episodic reward corresponds to the moving average using a window size of 50. Note that, in this figure, invalid SMILES are displayed with a reward of 0 but are given a reward of $-1$ during training.
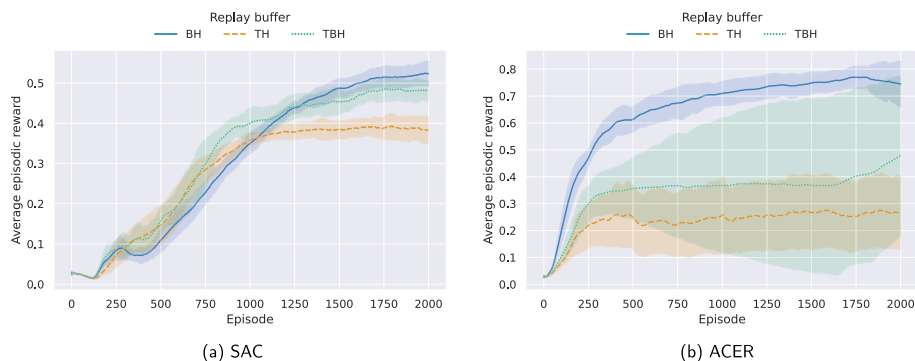
**Fig. 10** Average episodic reward computed over a batch of sequences, for the off-policy algorithms SAC and ACER (higher is better) when utilizing identical molecular scaffold filter. It shows mean and standard deviation of moving average, of window size 50, over 11 runs for each policy and replay buffer. 128 SMILES strings are sampled in each episode, with a budget of 2000 episodes in total. Note that invalid SMILES are displayed with a reward of 0 in this figure, but are given a reward of −1 during training

When using SAC, both *Bin history* and *Top-Bottom history* reach an average episodic reward of 0.5 approximately; while *Top history* reaches an average episodic reward of 0.4 approximately, which is the minimum reward for a sequence to be saved in the diversity filter. For ACER, *Bin history* reaches an episodic reward of around 0.75; while using the other two replay buffers yields substantially lower episodic rewards and higher variances, in particular for *Top-Bottom history*. *Top history* reaches an episodic reward below 0.3, not reaching above the diversity filter threshold. ACER with *Bin history* is the only combination that is able to obtain an average episodic reward comparable to the best on-policy algorithms, but with a slower increase in average episodic reward. SAC displays a significantly slower increase in average episodic reward than both ACER and the on-policy algorithms.

### 5.2.2 Without diversity filter

Figure 11 displays box plots of the number of active molecules and scaffolds, over 11 repeated runs for each combination of off-policy algorithms and replay buffers. Note that invalid SMILES are displayed again with a reward of 0 in this figure but are given a reward of −1 during training. No diversity filter is used, i.e., the generation of similar molecules is not penalized between episodes. It is observed that SAC using *Top history* generates the largest number of (unique) active molecules and scaffolds, on par with the best on-policy without a diversity filter, Regularized MLE with *Bin current*. For all replay buffers, SAC seems to generate a significantly larger number of active molecules and scaffolds compared to ACER, except for topological scaffolds when using *Top-bottom* history where they display similar performance.

Figure 12 displays means and standard deviations, over 11 repeated runs, of the average episodic rewards of sampled molecules for all combinations of off-policy algorithms and replay buffers using no diversity filter. For SAC, using *Top-Bottom history* gives an average episodic reward of over 0.8; while *Bin history* converges to an average episodic reward of around 0.6 and *Top history* keeps improving, almost reaching an average episodic reward of 0.8. For ACER, *Top-Bottom history* reaches an average episodic reward close to 1. When using *Bin history*, a significantly faster improvement in the first 500 episodes, compared to
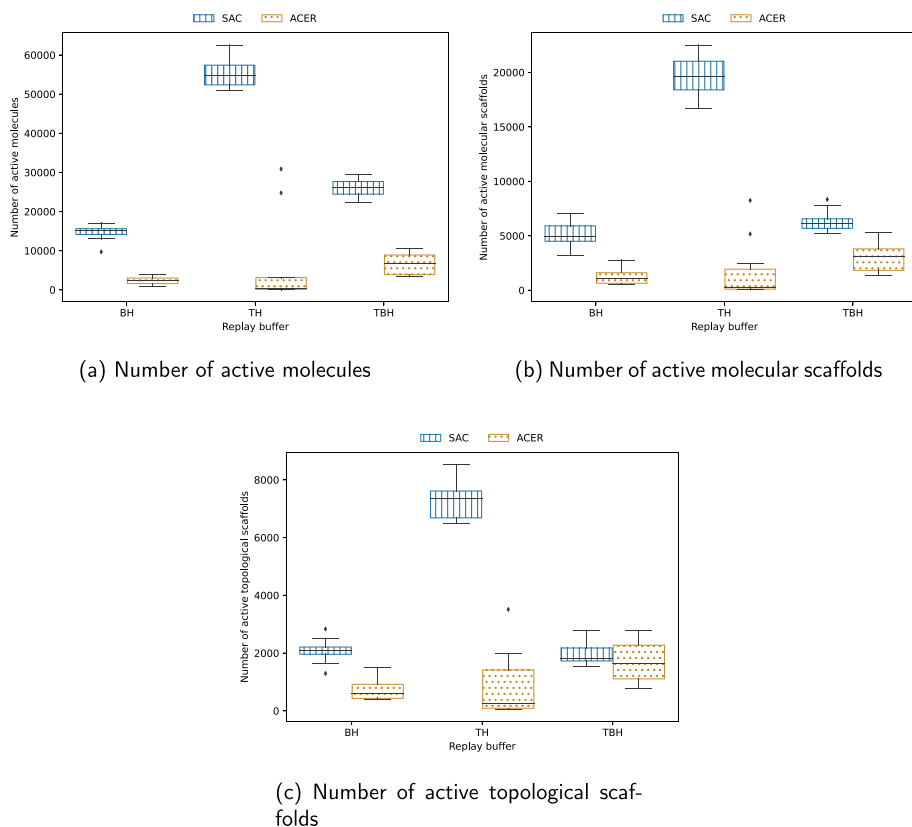
(a) Number of active molecules

(b) Number of active molecular scaffolds



(c) Number of active topological scaf-
folds

**Fig. 11** Box plots of the number of unique active molecules and active scaffolds for the off-policy algo-rithms ACER and SAC (higher is better) when utilizing no diversity filter. It shows mean and standard deviation over 11 runs for each policy and replay buffer. 128 SMILES strings are sampled in each episode, with a budget of 2000 episodes in total

*Top-Bottom history*, is observed but then diverges. A possible explanation for this is that it finds a mode with higher entropy, but that gives a lower score, it accidentally forgets too much of the pre-trained model. For *Top history*, ACER shows a large variability and does not consistently reach an average episodic reward above 0.4.

## 6 Discussion

In this section, we discuss the de novo drug design of investigated on- and off-policy algo-rithms and replay buffers. Both the use of a diversity filter and the use of no diversity filter are discussed. We consider a diversity filter that penalizes molecules with identical molec-ular scaffolds. As a result, there is a natural increase in diversity regarding molecular scaf-folds, but not necessarily topological scaffolds since different molecular scaffolds can have the same topological scaffold. When using no diversity filter, a large number of molecular scaffolds seems to lead to a large number of topological scaffolds. However, this is not as
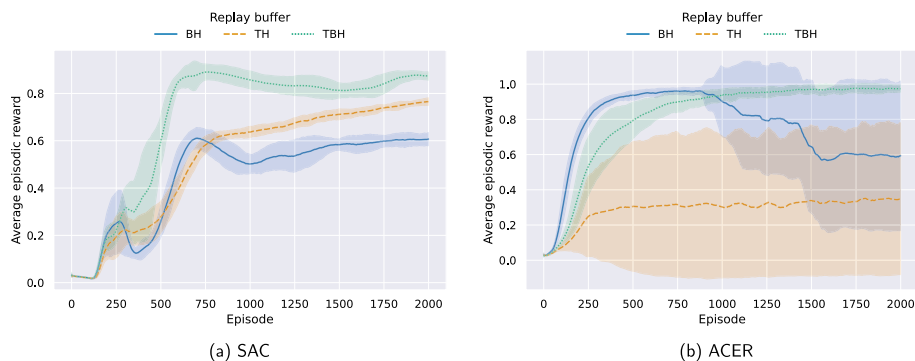
**Fig. 12** Average episodic reward computed over a batch of sequences, for the off-policy algorithms SAC and ACER (higher is better) when utilizing no diversity filter. It shows mean and standard deviation of moving average, of window size 50, over 11 runs for each policy and replay buffer. 128 SMILES strings are sampled in each episode, with a budget of 2000 episodes in total. Note that invalid SMILES are displayed with a reward of 0 in this figure, but are given a reward of −1 during training

evident when using the diversity filter. A reason for this can be that there is a substantially larger number of molecular scaffolds generated when using a diversity filter which naturally leads to more topological scaffolds being the same.

Molecules are generated using a model pre-trained on a data set derived from the ChEMBL database which will teach the agent to generate ChEMBL-like molecules. The DRD2 data set, which is used to create the scoring function, comprises data from both the ChEMBL and PubChem databases (Wang et al., 2017). Hence, this experimental setup only considers small molecules and the agent is not anticipated to have the ability to produce molecules that differ significantly from those present in the ChEMBL database. Previous work suggests that there are close to 500 000 molecular scaffolds in total in the ChEMBL database (Liang et al., 2019). On the other hand, all scaffolds do not necessarily contain molecules active toward DRD2. The budget of 2000 episodes used in this work also limits the search for new scaffolds. Typically, we are able to identify scaffolds consisting of SMILES with a score of 1 and hence successfully traverse the reward landscape.

## 6.1 On-policy algorithms

This section discusses the results of the on-policy algorithms and replay buffers; both with and without a diversity filter.

### 6.1.1 With diversity filter

Both A2C and PPO generate more active molecules but fewer active scaffolds, compared to Regularized MLE. One possible explanation for this could be that A2C and PPO stay for a longer time close to a penalized scaffold before moving to the next scaffold. Regularized MLE uses a fixed pre-trained actor in the loss and can, therefore, easily jump between scaffolds without forgetting how to generate valid SMILES strings. This comes at the cost of its likelihood staying close to that of the pre-trained model. However, we do not observe that this notably limits its performance. We observe that on-policy algorithms can enhance

diversity by not only relying on high-scoring molecules, particularly regarding the diversity of topological scaffolds.

A2C and PPO achieve high average episodic rewards both by either using all on-policy data or a subset comprising both high- and low-scoring molecules. For some combinations of algorithm and replay mechanism is it advantageous to use historical samples instead of only current samples. However, there seems to be no significant added performance gain in using experiences from previous iterations, particularly compared to using all current samples. One possible explanation for this could be that the diversity filter can impose a significant change in the reward landscape after each episode by initiating penalization of frequently generated scaffolds and therefore possibly making previously acquired knowledge outdated, which may need to be accounted for. The investigated diversity filter uses a discrete threshold to determine when molecules of a scaffold should obtain a zero reward. A diversity filter that in a stepwise or continuous manner penalizes molecules from the same scaffold could have other effects on the reward landscape which possibly could be more suitable for some replay buffers.

### 6.1.2 Without diversity filter

As expected, the number of actives molecules and scaffolds is significantly lower compared to when using a diversity filter. There is an evident relationship between the number of active molecules, molecular scaffolds, and topological scaffolds. This is not necessarily evident when utilizing diversity filter.

When using no diversity filter, Regularized MLE generates a substantially larger number of active molecules and scaffolds, compared to A2C and PPO, especially when used in combination with the replay buffers *Bin current*, *Bin history* and *All current*. There is no significant difference in using a replay mechanism utilizing historical or current samples, except for the bin-based replay mechanisms in combination with Regularized MLE. *Bin current* in combination with Regularized MLE displays the overall largest number of active molecules and scaffolds while displaying among the lowest average episodic reward. This replay buffer seems to be able to enhance the exploration of Regularized MLE, possibly because it does not only exploit high-scoring molecules. Hence, it is evidently important to use a diverse set, in terms of reward, for learning without a diversity filter.

Why Regularized MLE performs better, in general, is likely because it is heavily regularized to stay close to the pre-trained model in terms of likelihood and, therefore, inherently can jump between scaffolds without using the initial knowledge. Without a diversity filter, PPO generates the least number of active molecules and scaffolds, while still displaying a high average episodic reward. This must be because of an early mode collapse, leading to PPO generating more or less only the same molecules through all episodes.

### 6.2 Off-policy algorithms

In this section, the results of the off-policy algorithms and replay buffers are discussed; both with and without a diversity filter.

### 6.2.1 With diversity filter

For the best off-policy algorithm and replay buffer combination, i.e., ACER with *Bin history*, the number of active molecules is on par with the best on-policy combinations. On the other hand, several off-policy results struggle to reach an episodic reward significantly above 0.4, which is the minimum reward for a generated SMILES string to be stored in the diversity filter. Only ACER with *Bin history* can consistently reach an average episodic reward above 0.7. With the help of the diversity filter, this leads to the highest number of active molecules and scaffolds. Hence, the slower convergence rate can possibly be explained by a more elaborate exploration phase. However, ACER shows a large variability. Even though SAC does generally achieve lower average episodic rewards, it is more stable and shows a larger median of active molecules and scaffolds compared to ACER utilizing *Top history* and *Top-bottom history*. It appears that the average episodic reward of SAC with *Bin history* has not converged yet, indicating that it is still in the learning phase. It is possible that this specific combination requires over 2000 episodes to conduct adequate exploration.

### 6.2.2 Without diversity filter

For the off-policy algorithms with no diversity filter, it is observed that SAC using *Top history* generates the largest number of unique active molecules and scaffolds. However, it does not reach the highest average episodic reward in this setting, which is achieved by ACER. ACER with *Top-bottom* history converges to the highest average episodic reward, among the off-policy algorithms without diversity filter, but is still outperformed by SAC for the generation of active molecules and scaffolds. This is because it generates many duplicates of the same high-scoring molecules. Hence, SAC will generate more unique molecules when no diversity filter is used and, consequently, improve the exploration. This behavior yields an enhancement in the diversity of the active molecules, especially when the top-scoring molecules are used for off-policy updates. Overall, this highlights the positive impact an appropriate replay buffer can have on the generation of diverse molecules. It also emphasizes the positive effect that the usage of a diversity filter has and that inherent exploration of the algorithms is not necessarily enough to generate a structurally diverse set of active molecules.

## 7 Conclusions

We explore on- and off-policy RL algorithms for SMILES-based molecular de novo drug design using recurrent neural networks. The investigation has focused on how well the algorithms sample structurally diverse and high-rewarding molecules for different replay buffers. This has been done by studying their behaviors both with and without using a diversity filter that penalizes the generation of similar molecules between episodes.

For on-policy algorithms, we observe that it is often favorable to use all generated molecules from the current batch for learning. Regularized MLE utilizing the full batch for learning, in combination with a diversity filter, leads to the overall best performance in terms of both the reward and diversity. However, it is possible to obtain similar performance by learning from fewer samples if the training data includes at least both

high-rewarding and low-rewarding data points. For these on-policy algorithms applied with no diversity filter, it is also important to use intermediate-rewarding samples, either from the current batch of sampled molecules or previously sampled molecules.

There is a potential performance gain in using off-policy algorithms with a suitable replay buffer. When using no diversity filter, we observe that SAC provides good exploration, leading to a more structurally diverse generation. Hence, when no diversity filter is used, the policy must keep its randomness to avoid mode collapse. Interestingly, when using a diversity filter ACER yields better performance, displaying the potential to be on par with Regularized MLE or even better.

We release the source code of the methods as an open-source framework, to enable further exploration of reinforcement learning algorithms and replay buffers mechanisms.

## Appendix A: Hyperparameters

### Regularized maximum likelihood estimation

See Table 1.

**Table 1** Regularized MLE Hyperparameters

| Parameter | Value |
|---|---|
| Optimizer | Adam (Kingma & Ba, 2014) |
| $\sigma$ | 128 |
| Margin threshold | 50 |
| Learning rate | 0.0001 |
| Gradient steps per update | 1 |

### Proximal policy opimization

See Table 2.

**Table 2** PPO Hyperparameters

| Parameter | Value |
|---|---|
| Optimizer | Adam (Kingma & Ba, 2014) |
| Clipping range ($\epsilon$) | 0.2 |
| Number of updates per episode | 4 |
| Discount ($\gamma$) | 0.99 |
| Number of mini-batches | 4 |
| Norm for gradient clipping | L2 norm |
| Maximum gradient norm | 0.5 |
| Learning rate | 0.0001 |
| Gradient steps per update | 1 |

## Advantage actor-critic

See Table 3.

**Table 3** A2C Hyperparameters

| Parameter | Value |
|---|---|
| Optimizer | Adam (Kingma & Ba, 2014) |
| Discount ($\gamma$) | 0.99 |
| Norm for gradient clipping | L2 norm |
| Maximum gradient norm | 0.5 |
| Learning rate actor | 0.0001 |
| Learning rate critic | 0.0001 |
| Gradient steps per update | 1 |

## Actor-critic with experience replay

See Table 4.

**Table 4** ACER Hyperparameters

| Parameter | Value |
|---|---|
| Optimizer | Adam (Kingma & Ba, 2014) |
| Replay rate ($\lambda$) | 4 |
| Discount ($\gamma$) | 0.99 |
| Target smoothing coefficient ($\tau$) | 0.95 |
| Entropy weight | 0.001 |
| Trust factor variance reduction ($\delta$) | 1 |
| Trust region clipping range ($c$) | 10 |
| Norm for gradient clipping | L2 norm |
| Maximum gradient norm | 0.5 |
| Learning rate | 0.0001 |
| Number of off-policy replay samples ($k$) | 128 |
| Gradient steps per update | 1 |
| Number of initial episodes without update | 10 |

## Soft actor-critic

See Table 5.

**Table 5** SAC Hyperparameters

| Parameter | Value |
| --- | --- |
| Optimizer | Adam (Kingma & Ba, 2014) |
| Number of off-policy updates | 4 |
| Discount ($\gamma$) | 1 |
| Smoothing coefficient ($\tau$) | 0.99 |
| Initial temperature ($\alpha_0$) | 0.001 |
| Norm for gradient clipping | L2 norm |
| Maximum gradient norm | 0.5 |
| Entropy target ($\bar{H}$) | 0.3 |
| Learning rate actor ($\lambda_\pi$) | 0.0001 |
| Learning rate critic ($\lambda_Q$) | 0.0001 |
| Learning rate temperature ($\lambda_\alpha$) | 0.0001 |
| Average policy weight | 0.5 |
| Number of off-policy replay samples ($k$) | 64 |
| Gradient steps per update | 1 |
| Number of initial episodes without update ($K_{\text{init}}$) | 10 |
| clipping range ($c$) | 0.5 |

## Appendix B: Technical details

For all models, training was done using Python 3.8.15 and PyTorch 1.13.1. Computations in this work were performed on a Linux cluster using Nvidia Tesla K80 and Nvidia V100 graphic cards utilizing CUDA 11.4. The experiments were run in parallel on around 50 graphic cards, where each run of an experiment was restricted to one graphic card at runtime.

For the scoring function and diversity filter, reinvent-scoring 0.0.73 is used with reinvent-chemistry 0.0.51. To train the DRD2 predictive model, which is used as scoring function, scikit-learn 1.2.0 is used. To compute fingerprints and scaffolds, RDKit 2022.9.3 is used.

## Appendix C: Visual comparison of generated SMILES

Top 10 generated SMILES of unique topological scaffolds of one representative run of each combination.

### On-policy algorithms

### With diversity filter

See Figs. 13, 14, 15.
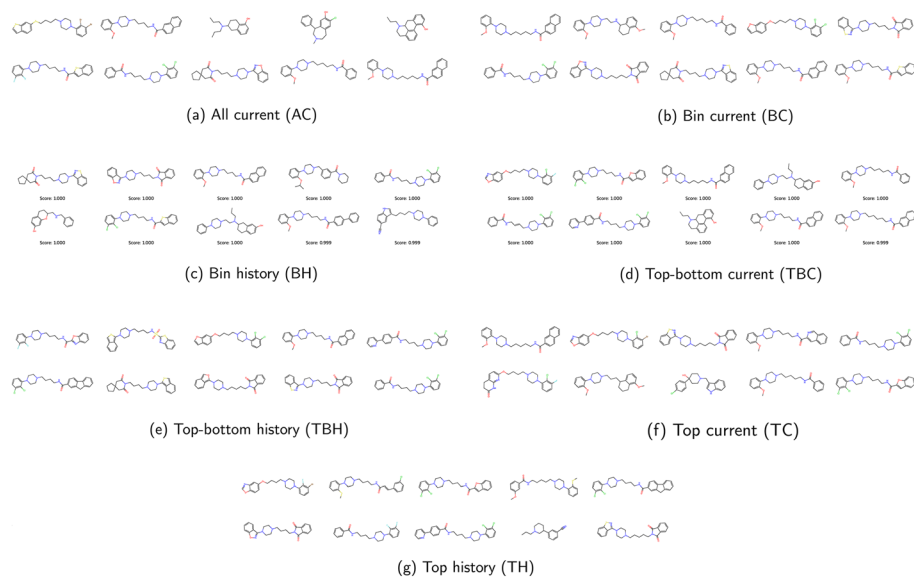
**Fig. 13** A2C with diversity filter penalizing the generation of SMILES with the same molecular scaffold. If no score is displayed, all scores are at least 0.9995. Otherwise, scores are rounded upwards to 3 decimals
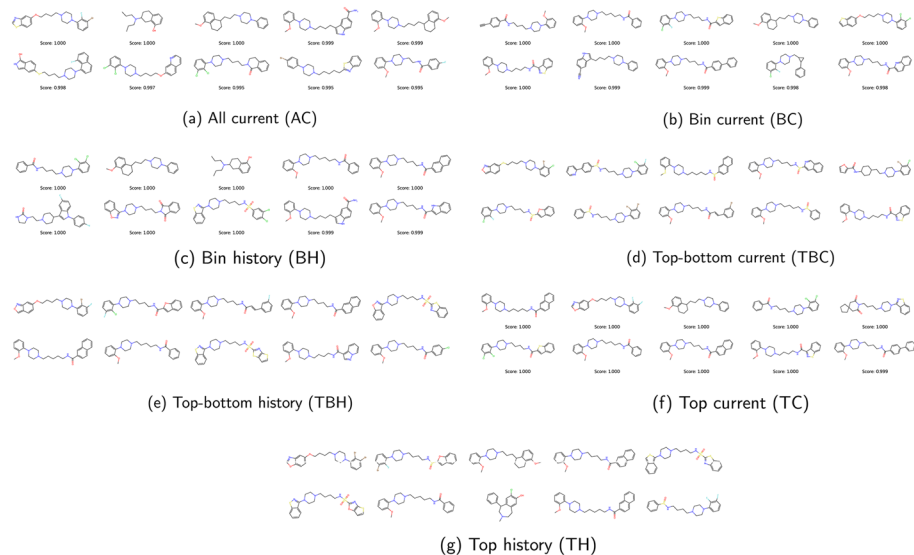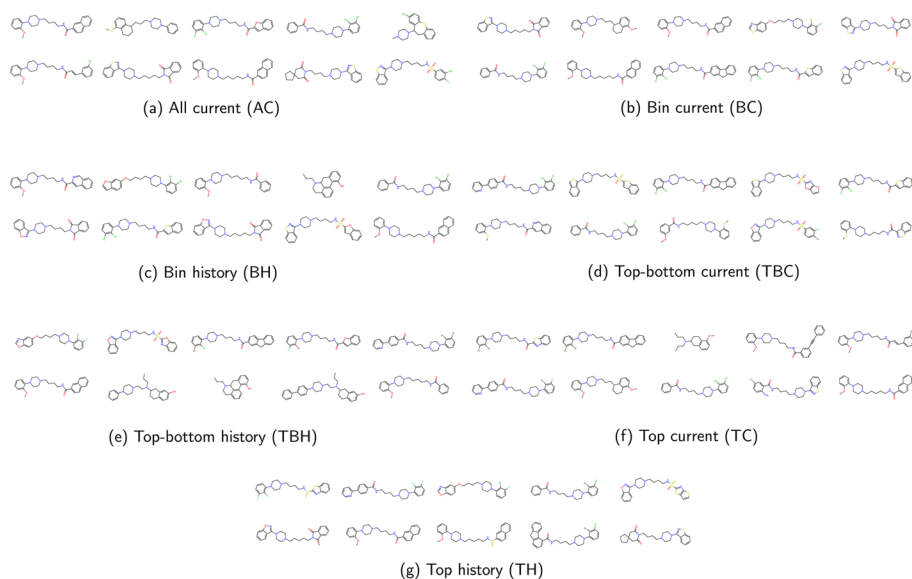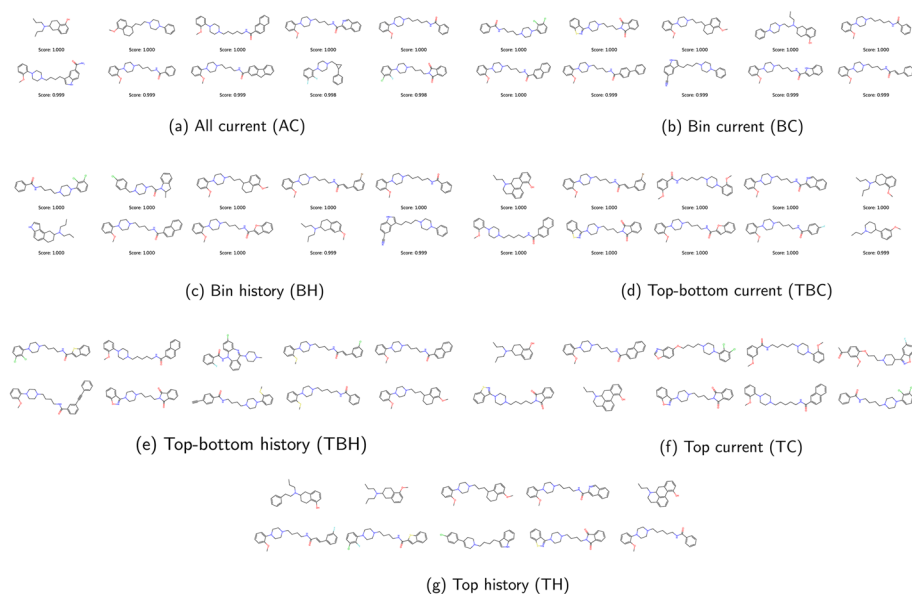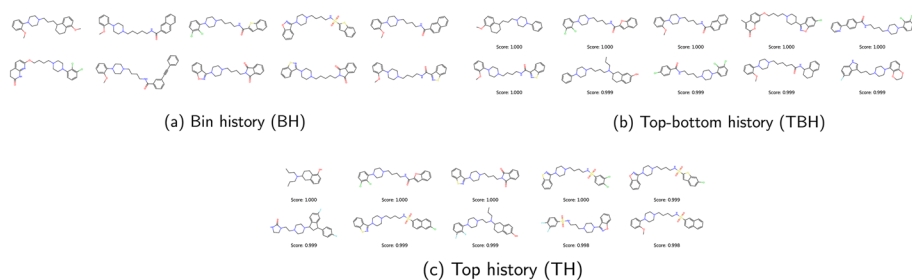


**Fig. 14** PPO with diversity filter penalizing the generation of SMILES with the same molecular scaffold. If no score is displayed, all scores are at least 0.9995. Otherwise, scores are rounded upwards to 3 decimals

(a) All current (AC)  (b) Bin current (BC)

(c) Bin history (BH)  (d) Top-bottom current (TBC)

(e) Top-bottom history (TBH)  (f) Top current (TC)

(g) Top history (TH)

**Fig. 15** Regularized MLE with diversity filter penalizing the generation of SMILES with the same molecular scaffold. If no score is displayed, all scores are at least 0.9995. Otherwise, scores are rounded upwards to 3 decimals
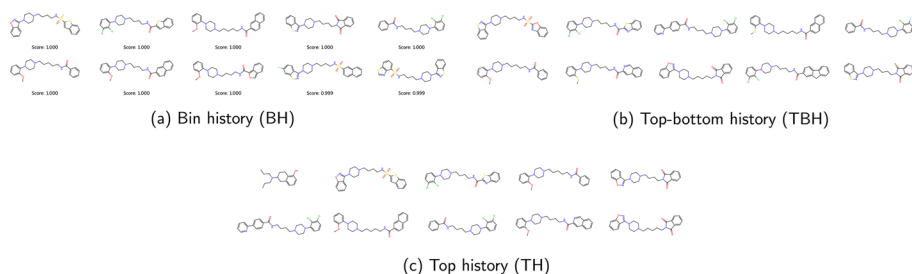
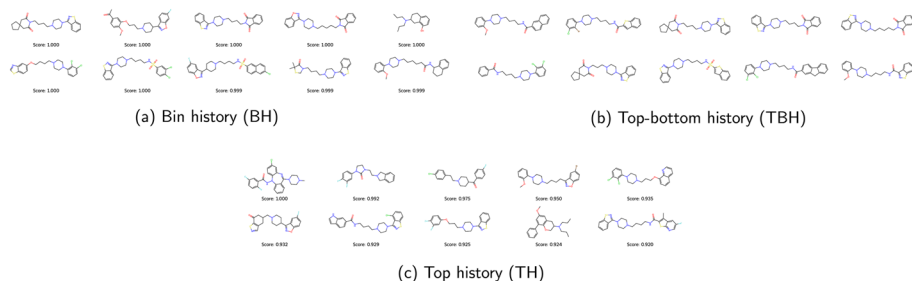## Without diversity filter

See Figs. 16, 17, 18.



(a) All current (AC)  (b) Bin current (BC)

(c) Bin history (BH)  (d) Top-bottom current (TBC)

(e) Top-bottom history (TBH)  (f) Top current (TC)

(g) Top history (TH)

**Fig. 16** A2C without diversity filter. If no score is displayed, all scores are at least 0.9995
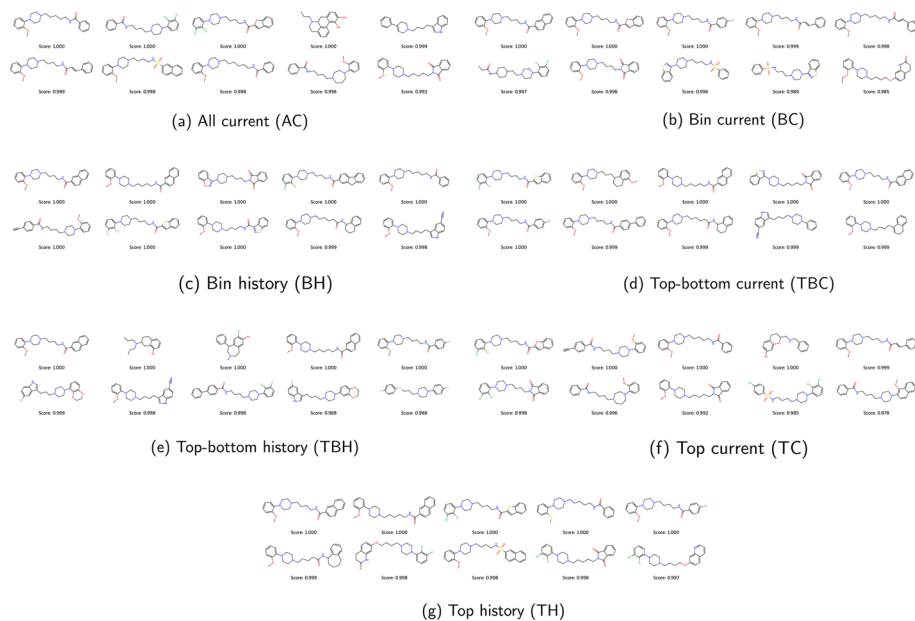
**Fig. 17** PPO without diversity filter. If no score is displayed, all scores are at least 0.9995
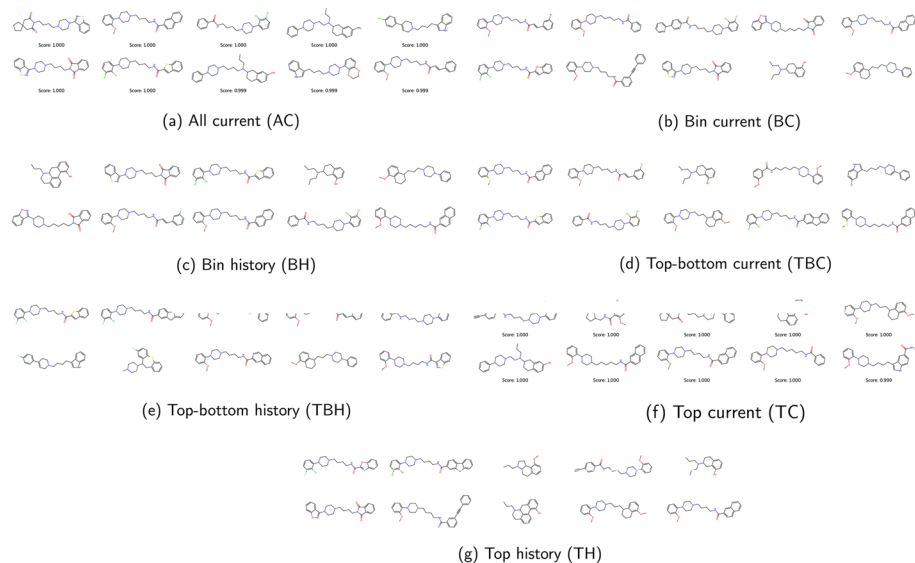


**Fig. 18** Regularized MLE without diversity filter. If no score is displayed, all scores are at least 0.9995

## Off-policy algorithms

## With diversity filter

See Figs. 19, 20.



(a) Bin history (BH)

(b) Top-bottom history (TBH)

(c) Top history (TH)

**Fig. 19** ACER with diversity filter penalizing the generation of SMILES with the same molecular scaffold. If no score is displayed, all scores are at least 0.9995. Otherwise, scores are rounded upwards to 3 decimals



(a) Bin history (BH)

(b) Top-bottom history (TBH)

(c) Top history (TH)

**Fig. 20** SAC with diversity filter penalizing the generation of SMILES with the same molecular scaffold. If no score is displayed, all scores are at least 0.9995. Otherwise, scores are rounded upwards to 3 decimals

## Without diversity filter

See Figs. 21, 22.



(a) Bin history (BH)

(b) Top-bottom history (TBH)

(c) Top history (TH)

**Fig. 21** ACER without diversity filter. If no score is displayed, all scores are at least 0.9995

(a) Bin history (BH)

(b) Top-bottom history (TBH)

(c) Top history (TH)
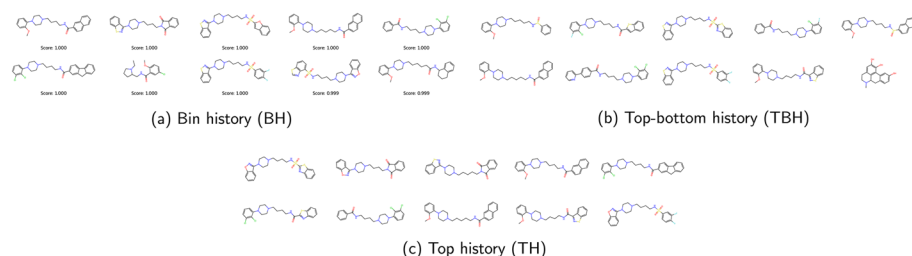
**Fig. 22** SAC without diversity filter. If no score is displayed, all scores are at least 0.9995

**Author's contribution** All authors have contributed significantly to the conception, design, and writing of this work. HGS developed the algorithms and performed the experimental studies. The work has been jointly supervised by MHC, OE and CT.

**Data availibility** Only public datasets, clearly cited in the text, are used.

**Code availability** The code is publicly available at https://github.com/MolecularAI/SMILES-RL.

## Declarations

**Conflict of interest** There is no conflict of interest.

## References

Bemis, G. W., & Murcko, M. A. (1996). The properties of known drugs. 1. molecular frameworks. *Journal of Medicinal Chemistry, 39*(15), 2887–2893.

Blaschke, T., Arús-Pous, J., Chen, H., Margreitter, C., Tyrchan, C., Engkvist, O., Papadopoulos, K., & Patronov, A. (2020). Reinvent 2.0: An ai tool for de novo drug design. *Journal of Chemical Information and Modeling, 60*(12), 5918–5922.

Blaschke, T., Engkvist, O., Bajorath, J., & Chen, H. (2020). Memory-assisted reinforcement learning for diverse molecular de novo design. *Journal of Cheminformatics, 12*(1), 1–17.

Bradshaw, J., Paige, B., Kusner, M. J., Segler, M., & Hernández-Lobato, J. M. (2020). Barking up the right tree: An approach to search over molecule synthesis dags. *Advances in Neural Information Processing Systems, 33*, 6852–6866.

Brown, N., Fiscato, M., Segler, M. H., & Vaucher, A. C. (2019). Guacamol: Benchmarking models for de novo molecular design. *Journal of Chemical Information and Modeling, 59*(3), 1096–1108.

Chen, H., Engkvist, O., Wang, Y., Olivecrona, M., & Blaschke, T. (2018). The rise of deep learning in drug discovery. *Drug Discovery Today, 23*(6), 1241–1250.

Christodoulou, P. (2019). *Soft actor-critic for discrete action settings*. arXiv preprint arXiv:1910.07207

Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., & Dabney, W. (2020). Revisiting fundamentals of experience replay. In *International Conference on Machine Learning*, pp. 3061–3071. PMLR.

Gao, W., Fu, T., Sun, J., & Coley, C. W. (2022). *Sample efficiency matters: a benchmark for practical molecular optimization*. arXiv preprint arXiv:2206.12411

Gaulton, A., Bellis, L. J., Bento, A. P., Chambers, J., Davies, M., Hersey, A., Light, Y., McGlinchey, S., Michalovich, D., Al-Lazikani, B., et al. (2012). Chembl: A large-scale bioactivity database for drug discovery. *Nucleic Acids Research, 40*(D1), 1100–1107.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., & Aspuru-Guzik, A. (2018). Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science, 4*(2), 268–276.

Gottipati, S. K., Sattarov, B., Niu, S., Pathak, Y., Wei, H., Liu, S., Blackburn, S., Thomas, K., Coley, C., Tang, J., et al. (2020). Learning to navigate the synthetically accessible chemical space using reinforcement learning. In *International Conference on Machine Learning,* pp. 3668–3679 . PMLR.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018). *Soft actor-critic algorithms and applications*. arXiv preprint arXiv:1812.05905

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780.

Horwood, J., & Noutahi, E. (2020). Molecular design in synthetically accessible chemical space via deep reinforcement learning. *ACS Omega, 5*(51), 32984–32994.

Hu, Y., Stumpfe, D., & Bajorath, J. (2016). Computational exploration of molecular scaffolds in medicinal chemistry: Miniperspective. *Journal of Medicinal Chemistry, 59*(9), 4062–4076.

Jin, W., Barzilay, R., & Jaakkola, T. (2018). Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning*, pp. 2323–2332. PMLR.

Jin, W., Barzilay, R., & Jaakkola, T. (2020). Multi-objective molecule generation using interpretable substructures. In *International Conference on Machine Learning*, pp. 4849–4859. PMLR

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980

Landrum, G. (2006). *RDKit: Open-source Cheminformatics*. Retrieved from https://www.rdkit.org/docs/Overview.html

Liang, L., Ma, C., Du, T., Zhao, Y., Zhao, X., Liu, M., Wang, Z., & Lin, J. (2019). Bioactivity-explorer: A web application for interactive visualization and exploration of bioactivity data. *Journal of Cheminformatics, 11*, 1–6.

Maus, N., Jones, H. T., Moore, J. S., Kusner, M. J., Bradshaw, J., & Gardner, J. R. (2022). *Local latent space bayesian optimization over structured inputs*. arXiv preprint arXiv:2201.11872

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In: Balcan, M.F., Weinberger, K.Q. (eds.) *Proceedings of The 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research,* vol. 48, pp. 1928–1937. PMLR, New York, New York, USA . https://proceedings.mlr.press/v48/mniha16.html

Neil, D., Segler, M., Guasch, L., Ahmed, M., Plumbley, D., Sellwood, M., & Brown, N. (2018). Exploring deep recurrent models with reinforcement learning for molecule design. In *6th International Conference on Learning Representations*.

Olivecrona, M., Blaschke, T., Engkvist, O., & Chen, H. (2017). Molecular de-novo design through deep reinforcement learning. *Journal of Cheminformatics, 9*(1), 1–14.

Rumelhart, D.E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation*. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). *Prioritized experience replay*. arXiv preprint arXiv:1511.05952

Schneider, G., & Fechner, U. (2005). Computer-based de novo design of drug-like molecules. *Nature Reviews Drug Discovery, 4*(8), 649–663.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal policy optimization algorithms*. arXiv preprint arXiv:1707.06347

Sun, J., Jeliazkova, N., Chupakhin, V., Golib-Dzib, J.-F., Engkvist, O., Carlsson, L., Wegner, J., Ceulemans, H., Georgiev, I., Jeliazkov, V., et al. (2017). Excape-db: An integrated large scale dataset facilitating big data analysis in chemogenomics. *Journal of Cheminformatics, 9*, 1–9.

Thomas, M., O'Boyle, N. M., Bender, A., & De Graaf, C. (2022). *Re-evaluating sample efficiency in de novo molecule generation*. arXiv preprint arXiv:2212.01385.

Thomas, M., O'Boyle, N. M., Bender, A., & De Graaf, C. (2022). Augmented hill-climb increases reinforcement learning efficiency for language-based de novo molecule generation. *Journal of Cheminformatics, 14*(1), 1–22.

Vamathevan, J., Clark, D., Czodrowski, P., Dunham, I., Ferran, E., Lee, G., Li, B., Madabhushi, A., Shah, P., Spitzer, M., et al. (2019). Applications of machine learning in drug discovery and development. *Nature Reviews Drug Discovery, 18*(6), 463–477.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., & de Freitas, N. (2016). *Sample efficient actor-critic with experience replay*. arXiv preprint arXiv:1611.01224

Wang, Y., Bryant, S. H., Cheng, T., Wang, J., Gindulyte, A., Shoemaker, B. A., Thiessen, P. A., He, S., & Zhang, J. (2017). Pubchem bioassay: 2017 update. *Nucleic Acids Research, 45*(D1), 955–963.

Weininger, D. (1988). 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.

Yang, S., Hwang, D., Lee, S., Ryu, S., & Hwang, S. J. (2021). Hit and lead discovery with explorative rl and fragment-based molecule generation. *Advances in Neural Information Processing Systems, 34*, 7924–7936.

Yang, X., Wang, Y., Byrne, R., Schneider, G., & Yang, S. (2019). Concepts of artificial intelligence for computer-assisted drug discovery. *Chemical Reviews, 119*(18), 10520–10594.

You, J., Liu, B., Ying, Z., Pande, V., & Leskovec, J. (2018). Graph convolutional policy network for goal-directed molecular graph generation. *Advances in Neural Information Processing Systems*, *31*, 6410–6421.

Zhang, J., Mercado, R., Engkvist, O., & Chen, H. (2021). Comparative study of deep generative models on chemical space coverage. *Journal of Chemical Information and Modeling, 61*(6), 2572–2581.

Zhou, H., Lin, Z., Li, J., Ye, D., Fu, Q., & Yang, W. (2022). *Revisiting discrete soft actor-critic*. arXiv preprint arXiv:2209.10081

Zhou, Z., Kearnes, S., Li, L., Zare, R. N., & Riley, P. (2019). Optimization of molecules via deep reinforcement learning. *Scientific Reports, 9*(1), 1–10.

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

## Authors and Affiliations

**Hampus Gummesson Svensson[1,2]** · **Christian Tyrchan[3]** · **Ola Engkvist[1,2]** · **Morteza Haghir Chehreghani[1]**

✉ Hampus Gummesson Svensson
hamsven@chalmers.se

Christian Tyrchan
christian.tyrchan@astrazeneca.com

Ola Engkvist
ola.engkvist@astrazeneca.com

Morteza Haghir Chehreghani
morteza.chehreghani@chalmers.se

[1] Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden

[2] Molecular AI, Discovery Sciences, R&D, AstraZeneca, Gothenburg, Sweden

[3] Medicinal Chemistry, Research and Early Development, Respiratory and Immunology (R&I), BioPharmaceuticals R&D, AstraZeneca, Gothenburg, Sweden