ACEGEN: A TORCHRL-BASED TOOLKIT FOR REINFORCEMENT LEARNING IN GENERATIVE CHEMISTRY

Albert Bou
Universitat Pompeu Fabra, Acellera

Morgan Thomas
Universitat Pompeu Fabra

Sebastian Dittert
Universitat Pompeu Fabra

Carles Navarro Ramírez
Acellera

Maciej Majewski
Acellera

Ye Wang
Biogen

Shivam Patel
PSIVANT

Gary Tresadern
Janssen Belgium

Mazen Ahmad
Janssen Belgium

Vincent Moens
PyTorch Team, Meta

Woody Sherman
PSIVANT

Simone Sciabola
Biogen

Gianni De Fabritiis
ICREA, Universitat Pompeu Fabra, Acellera

ABSTRACT

In recent years, reinforcement learning (RL) has been increasingly used in drug design to propose molecules with specific properties under defined constraints. However, RL problems are inherently complex, featuring independent and interchangeable components with diverse method signatures and data requirements, leading existing applications to convoluted code structures. This complexity not only complicates code comprehension but also hampers modification, hindering the smooth exploration of new ideas in the field and ultimately slowing down research. In this work, we apply TorchRL - a modern general decision-making library that provides well-integrated reusable components - to make a robust toolkit tailored for generative drug design. AceGen leverages general RL solutions which enhance simplicity, making the solutions more understandable, modifiable, and reliable. We demonstrate the application of AceGen for conditioned compound library generation implementing various RL algorithms to optimize drug design targets. Furthermore, with the tools made available we propose a novel algorithm inspired by the PPOD algorithm that outperforms all baselines as benchmarked on 23 drug design relevant targets. The library is accessible at `https://github.com/Acellera/acegen-open`.

1 INTRODUCTION

Drug design is a complex and challenging task that involves the identification of biomolecules with multiple optimal properties, such as potency, selectivity, bioavailability, and toxicity. However, traditional approaches to drug discovery are often slow and inefficient in delivering viable hits, leading researchers to explore machine learning solutions to improve the efficiency and success rate of virtual screening. In recent years, a diversity of generative model solutions have been proposed as a promising approach to partially automate this process Elton et al. (2019); Thomas et al. (2022a); Wang et al. (2022). These models typically employ machine learning algorithms to generate and suggest molecular candidates. However, these models can be computationally inefficient in generating molecules with optimal properties within the vast space of chemical compounds, sometimes requiring more than 100k trials Brown et al. (2019).

Reinforcement Learning (RL) (Sutton & Barto, 2018), a family of machine learning algorithms that use feedback as a learning signal, has emerged as a possible solution due to its ability to guide generative models to explore the vast space of chemical compounds in an efficient way (Olivecrona et al., 2017; Popova et al., 2018; Thomas et al., 2022b). Through a trial and error process, RL algorithms can adapt their search strategy to the specific characteristics of compounds associated with desir-

able properties, and propose novel molecules with similar characteristics. This is particularly useful when data isn't available for supervised learning, as is often the case in new drug discovery campaigns. In practice, these methods operate using a customized reward function that assigns values to molecules, tailored specifically for the application or research question being addressed. To navigate the molecular space and find the most highly rewarded molecules, RL agents alternate between two phases: a data collection phase and a learning phase. During the data collection, molecules are generated by the generative model and scored based on the reward function. In the agent learning phase, the collected data is utilized to enhance the agent's ability to generate interesting and relevant compounds. By leveraging this feedback loop, the generative model can be continually improved to generate compounds with increasingly desirable properties. However, despite promising results thus far, there is still an underutilized array of reinforcement learning techniques that have yet to be explored for the challenges of drug discovery.

Currently available implementations for drug discovery with RL rely heavily on custom code (Blaschke et al., 2020; Gao et al., 2022), which, unfortunately, hinders their overall impact. The limitation stems from the inherent nature of RL techniques, where agents are typically constructed using a combination of different interacting algorithmic components (such as actors, data collectors, and replay buffers), each with specific input and output data requirements. In modern RL, each agent component can come in various flavours with diverse method signatures and specific data requirements. To truly advance the field, it is essential that available implementations empower practitioners to explore all potential components and facilitate the development integration of novel ideas. However, straightforward implementations encounter a dilemma. They must either narrow down the available options, diminishing the code's utility, or increase complexity to accommodate a broader range of options while ensuring compatibility among components. This trade-off compromises readability, modifiability, and, ultimately, the flexibility to incorporate new ideas.

The RL community already provides elegant solutions to address this challenge. TorchRL, a comprehensive RL library (Bou et al., 2023), offers well-tested independent components and a robust data carrier known as TensorDict to ensure integration regardless of their input and output data signatures. In this work, we propose to adopt TorchRL's components as building blocks to assemble efficient and reliable drug discovery agents - a practice already successfully applied in diverse domains like drone control (Xu et al., 2024) and combinatorial optimization (Berto et al., 2023). TorchRL allows to consolidate the training logic into a single script, where components are created and combined in a cohesive training loop. This methodology enhances clarity and realiability compared to alternative implementations. This setup also facilitates straightforward replacements of components for experimenting with novel ideas.

AceGen therefore proposes to utilise this library to conduct and explore RL algorithms for generative chemical design. To showcase the advantages of our approach, we implement and evaluate well-known language-based algorithms for drug design. Language models can learn complex patterns in text and generate novel sequences, including molecular structures (Segler et al., 2018; Gómez-Bombarelli et al., 2018). Studies have focused on utilizing Simplified Molecular Input Line Entry System (SMILES) (Weininger, 1988), a string notation system used to represent chemical structures in a compact and standardized manner. Moreover, several prior studies have evidenced the benefits of combining generative language models and RL (Olivecrona et al., 2017; Fialková et al., 2021; Guo et al., 2022). Therefore, we re-implement two current state-of-the-art algorithms - Reinvent (Olivecrona et al., 2017) and AHC (Thomas et al., 2022b) - and also adapt general RL algorithms such as A2C (Mnih et al., 2016) and PPO (Schulman et al., 2017) to our problem setting. Moreover, we propose a novel algorithm inspired by the PPOD (Libardi et al., 2021) approach, but with our own modifications. This new algorithm exhibits superior sample efficiency compared to all previously mentioned algorithms, as demonstrated by its success on the MolOpt benchmark (Gao et al., 2022). Our code, accessible at `https://github.com/Acellera/acegen-open`, offers all of these implementations to generate compound libraries. More importantly, it serves as a toolkit to easily and efficiently implement new architectures, RL algorithms and new ideas.

1.1 Problem Definition

As previously mentioned, the process of sequentially designing molecules can be framed as a reinforcement learning (RL) problem. In this context, an RL agent interacts with an environment in a series of episodes to create compounds. Each episode begins with a single special start token and

can last for a varying number of steps. At each time step, the RL agent utilizes the representation of the partially completed molecule, referred as the current observation (s_t) in RL terms, to predict a probability distribution over all possible next tokens ($\pi(a_t|s_t)$), which represent the agent's available actions. The agent selects a token and provides it to the environment, which updates the molecule representation and provides the next observation to the agent. The episode ends when the agent chooses to append the stop token. After an episode is completed, a scalar reward value is assigned to the generated compound. The objective of the RL problem is for the agent to learn over time how to create molecules that result in high rewards.

1.2 Reinforcement learning environment

To effectively address the defined problem using TorchRL the main prerequisite is to define an environment component, a flexible interface that conveniently handles environment transitions and stores tensor batches in TensorDict's. It is the component that manages the segment of the RL loop responsible for providing observations in response to the agent's actions. This environment class inherits from the TorchRL base environment component *EnvBase*, providing a range of advantages that include input and output data transformations, compatibility with Gym-based (Brockman et al., 2016) APIs, efficient vectorized options (enabling the generation of multiple molecules in parallel), and the retrieval of clearly specified information attributes regarding expected input and output data. With this environment, all TorchRL components become available for creating potential solutions.

AceGen currently provides an environment for language model experimentation with SMILES representation, complemented by a user-friendly vocabulary class. However, extending the TorchRL *EnvBase* class is straightforward, and a similar strategy could be applied to craft environments tailored to various generative models, including 3D molecular models. Appendix A illustrates how the vocabulary and environment can be easily created and utilized for data generation.

1.3 Scoring and evaluation of molecules

To extend the implementation of RL to generative chemical models beyond the theoretical, practically relevant scoring functions and objective tasks must be designed to assign rewards to generated compounds. Scoring functions must reflect real-world drug design scenarios and be flexible enough to apply to a range of drug design challenges, yet easy enough to implement without needing to write any code. This challenging requirement has led to overly simplistic but irrelevant objective functions such as penalized logP Jin et al. (2018), or code that is too complex and fixed to one particular generative model Blaschke et al. (2020). Recently, MolScore Thomas et al. (2023) was proposed as a framework that offers all of the needed characteristics including a broad range of drug design relevant scoring functions, simple integration into any generative model, flexible objective design without any coding necessary, as well as distributed computing for more computationally expensive but more useful scoring functions such as molecular docking Thomas et al. (2021). The most recent version also contains a benchmarking mode which we use as a reimplementation of the MolOpt benchmark Gao et al. (2022). Therefore, by default, we delegate the scoring and evaluation of molecules to MolScore, which already satisfies all the necessary requirements. However, we also provide the option to bypass the library entirely and implement completely custom scoring functions.

1.4 Training implementations

We have implemented a suite of language-based algorithms for *de novo* drug discovery. Within these scripts, we've streamlined tasks such as data collection, experience replay, and loss computation by leveraging various TorchRL components. Notably, our design prioritizes easy customization, making it as straightforward as possible for practitioners to adapt them to their specific needs. The scripts are structured to facilitate sequential data processing using TensorDicts, with each component contributing to the training process's logic. Modifying the algorithms is easy; users can simply swap out specific components along the data processing pipeline with others. Despite their inherent flexibility, the scripts are user-friendly and don't require extensive RL expertise to utilize effectively. The suite includes scripts for training RL agents utilizing Reinvent Blaschke et al. (2020), AHC Thomas et al. (2022b), A2C Mnih et al. (2016) and PPO Schulman et al. (2017) algorithms, and an adaptation of the PPOD Libardi et al. (2021) algorithm. We deviate from the exact implementa-

tion of PPOD proposed in the original work, opting for modifications that better suit our problem. Specifically, we omit the use of an initial expert demonstration, refrain from replaying episodes with high-value predictions (only episodes with high reward are replayed), and employ a fixed amount of replay data per batch. Additionally, for A2C, PPO and PPOD we introduce an extra term to the loss function based on the Kullback-Leibler divergence (KL) between the actor policy and a prior policy. This term serves to penalize the policy for deviating too much from the prior. The incorporation of KL constraints is a common practice in research papers aligning language models with custom reward functions, as seen in examples utilizing human feedback Menick et al. (2022); Ouyang et al. (2022); Bai et al. (2022). Finally, we also provide a pre-training script for language models to learn to generate valid SMILES through the teacher enforcing method (Lamb et al., 2016), which can be used to train prior policies. Our code is engineered to adapt to the available computational resources, whether it's a single GPU or a distributed setup spanning multiple machines and GPUs. This adaptability allows to efficiently train on datasets of large size.

1.5 METHOD EXPLORATION

As previously mentioned, employing a highly modular coding approach and a training logic that combines independent components through a robust communication standard that ensures compatibility, presents a compelling advantage. Specifically, this approach allows the seamless replacement of specific components with others, facilitating the testing of different ideas without risking errors elsewhere in the code. In the context of the problem at hand, the generative policy emerges as a particularly crucial component. Exploring policy architectures holds significant importance for practitioners in the field. TorchRL streamlines this exploration by facilitating the encapsulation of PyTorch *nn.Modules* and ensuring automatic compatibility with TensorDict. A basic example illustrating this process can be found in Appendix B, where an RNN-based policy is adapted to smoothly generate actions based on the environment's input. While our repository currently provides methods to create LSTM (Hochreiter & Schmidhuber, 1997), GRU (Cho et al., 2014) and GPT2 Radford et al. (2019) policies, practitioners can similarly include any other architectures of their choice. Additionally, as detailed in subsection 1.3, both the availability of reference scoring benchmarks and the flexibility to define custom scoring functions is highly relevant to drug discovery research. Appendix C showcases how both options can be accommodated within our codebase. Additionally, the library contains step-by-step tutorials to assist users in implementing custom scoring functions and custom policies.

1.6 CONSTRAINED MOLECULAR GENERATION MODES

In drug discovery pipelines, merely generating molecules from scratch may not always suffice. To meet diverse requirements, AceGen solutions offer multiple sampling modes. By efficiently integrating PromptSMILES (Thomas et al., 2024), a simple method enabling constrained molecule generation using models pre-trained solely with teacher enforcement on full SMILES, it is possible to generate molecules while adhering to specific chemical sub-structures. Specifically, in addition to de novo generation, AceGen scripts allow for easy configuration of scaffold decoration and fragment linking molecule generation modes. Constrained sampling modifies only the data collection behavior, operating independently of all other agent components, in line with the TorchRL philosophy. Tutorials on performing constrained generation are provided within the repository.

1.7 DISTRIBUTION AND AVAILABLE RESOURCES

AceGen is open-sourced on GitHub under the MIT license. The main dependencies of AceGen are TorchRL (Bou et al., 2023), TensorDict (Bou et al., 2023), and, optionally, MolScore (Thomas et al., 2023) and PromptSMILES (Thomas et al., 2024). Additionally, within the repository, we currently provide pretrained weights for several policies. More specifically, we provide GRU policies pretrained on two datasets: ChEMBL (Gaulton et al., 2012) and ZINC Gómez-Bombarelli et al. (2018). We also provide weights for a LSTM policy pretrained on ChEMBL (Gaulton et al., 2012) and for a GPT2 (Radford et al., 2019) pretrained on the REAL Database REAL lead-like compounds (Shivanyuk et al., 2007).

It's important to highlight that TorchRL operates within the PyTorch ecosystem (Paszke et al., 2019), ensuring active collaboration for ongoing development. This facilitates the integration of new ideas

as new TorchRL components, thereby allowing future advancements in decision-making to seamlessly integrate into AceGen.

2 RESULTS

2.1 PRETRAINING

To demonstrate the integration of TorchRL with generative chemical models, we implement an RNN based on the SMILES representation of molecules. This architecture has consistently shown state-of-the-art results Brown et al. (2019); Polykovskiy et al. (2020); Cieplinski et al. (2020); Huang et al. (2021); Gao et al. (2022) even in light of more modern architectures such as graph neural networks (e.g., Mercado et al. (2021)). The model was trained on a subset of ChEMBL28 filtered to only include the most desirable chemistry from a drug design perspective. For further details on dataset and model training see Appendix D.

2.2 MOLOPT BENCHMARK RESULTS

To assess the algorithms, we undergo multiple training processes using the Practical Molecular Optimization (MolOpt) benchmark (Guo et al., 2022). This benchmark encompasses 23 distinct tasks, each associated with different targets. All algorithms use the same policy architecture and the weights pretrained as described in the previous section. In the case of Reinvent and AHC, we adopt the hyperparameters suggested by the respective authors in their original papers. Additionally, for the REINFORCE algorithm, we also conduct a test using the hyperparameters recommended in the MolOpt paper. As for A2C, PPO, and PPOD, we engage in manual hyperparameter tuning. The specific hyperparameter values are provided in our repository. We give a budget of 10K molecules per scoring function to each algorithm and task as per the original benchmark. Results for the AUC of the top 100 molecules, which captures optimization ability but specially sample efficiency, are shown in Table 1. Complementary metrics are included in Appendix E. We also conducted a sanity check by comparing our Reinvent implementation with the implementation used in the MolOpt paper. Our simplified re-implementation achieved better results with faster training. These results are shown in Appendix F.

Task	Reinvent (default)	Reinvent (MolOpt)	AHC	A2C	PPO	PPOD
Albuterol_similarity	0.569	0.865	0.640	0.760	0.911	**0.919**
Amlodipine_MPO	0.506	0.626	0.505	0.511	0.553	**0.656**
C7H8N2O2	0.615	0.871	0.563	0.737	0.864	**0.875**
C9H10N2O2PF2Cl	0.556	0.721	0.553	0.610	0.625	**0.756**
Celecoxxib_rediscovery	0.566	0.812	0.590	0.700	0.647	**0.888**
Deco_hop	0.602	**0.657**	0.616	0.605	0.601	0.646
Fexofenadine_MPO	0.668	**0.765**	0.680	0.663	0.687	0.747
Median_molecules_1	0.199	0.348	0.197	0.321	0.362	**0.363**
Median_molecules_2	0.195	0.270	0.208	0.224	0.236	**0.285**
Mestranol_similarity	0.454	0.821	0.514	0.645	0.728	**0.870**
Osimertinib_MPO	0.782	**0.837**	0.791	0.780	0.798	0.815
Perindopril_MPO	0.430	**0.516**	0.431	0.444	0.477	0.506
QED	0.922	0.931	0.925	0.927	**0.933**	**0.933**
Ranolazine_MPO	0.626	**0.721**	0.635	0.681	0.681	0.706
Scaffold_hop	0.758	**0.834**	0.772	0.764	0.761	0.808
Sitagliptin_MPO	0.226	0.356	0.219	0.272	0.295	**0.372**
Thiothixene_rediscovery	0.350	0.539	0.385	0.446	0.473	**0.570**
Troglitazone_rediscovery	0.256	0.447	0.282	0.305	0.449	0.511
Valsartan_smarts	0.012	0.014	0.011	0.010	**0.022**	**0.022**
Zaleplon_MPO	0.408	**0.496**	0.412	0.415	0.469	0.490
DRD2	0.907	**0.963**	0.906	0.942	**0.967**	0.963
GSK3B	0.738	0.890	0.719	0.781	0.863	**0.891**
JNK3	0.640	0.817	0.649	0.660	0.770	**0.842**
Total	11.985	15.118	12.205	13.203	14.170	**15.434**

Table 1: Algorithm comparison for the Area Under the Curve (AUC) of the top 100 molecules on MolOpt benchmark scoring functions. Each algorithm ran 5 times with different seeds, and results were averaged.

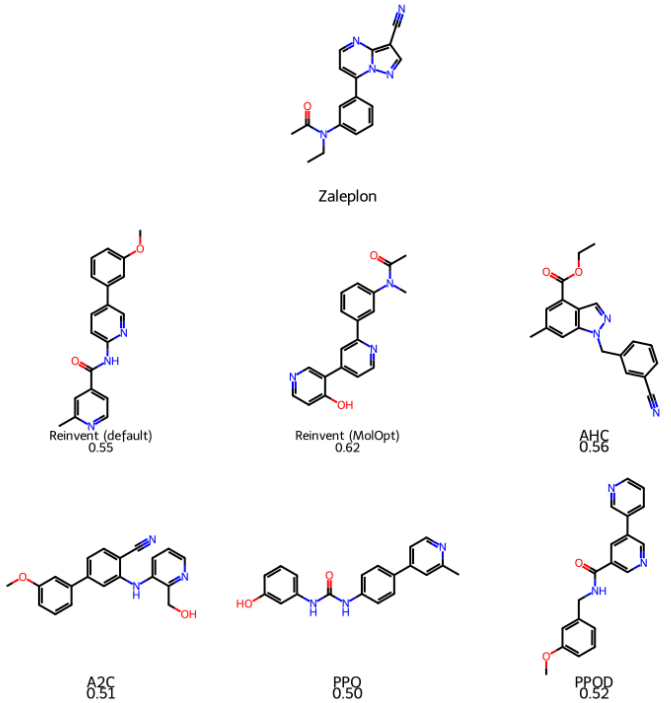

Figure 1: Demonstrative example of the top molecule on the Zaleplon MPO task by RL algorithm. Similar scores are achieved but different solutions reached based on RL algorithm.

3 Conclusion

This study proposes utilizing TorchRL to explore RL algorithms for generative chemical design, conveniently packaged into AceGen. We highlight the advantages of this approach in several ways. Firstly, we re-implement well-known language-based algorithms for generative drug discovery with a single logic script to demonstrate enhanced simplicity and readability. Secondly, we showcase modifiability by adapting other algorithms to the drug discovery setting. Additionally, we introduce and adaptation of the PPOD algorithm, which demonstrates superior sample efficiency compared to other tested algorithms on the MolOpt benchmark, contributing to the ongoing optimization of methodologies within the domain of drug discovery.

References

Josep Arús-Pous, Simon Viet Johansson, Oleksii Prykhodko, Esben Jannik Bjerrum, Christian Tyrchan, Jean-Louis Reymond, Hongming Chen, and Ola Engkvist. Randomized smiles strings improve the quality of molecular generative models. *Journal of cheminformatics*, 11(1):1–13, 2019.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Federico Berto, Chuanbo Hua, Junyoung Park, Minsu Kim, Hyeonah Kim, Jiwoo Son, Haeyeon Kim, Joungho Kim, and Jinkyoo Park. Rl4co: an extensive reinforcement learning for combinatorial optimization benchmark. *arXiv preprint arXiv:2306.17100*, 2023.

Thomas Blaschke, Josep Arús-Pous, Hongming Chen, Christian Margreitter, Christian Tyrchan, Ola Engkvist, Kostas Papadopoulos, and Atanas Patronov. Reinvent 2.0: an ai tool for de novo drug design. *Journal of chemical information and modeling*, 60(12):5918–5922, 2020.

Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. Torchrl: A data-driven decision-making library for pytorch, 2023.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Nathan Brown, Marco Fiscato, Marwin HS Segler, and Alain C Vaucher. Guacamol: benchmarking models for de novo molecular design. *Journal of chemical information and modeling*, 59(3): 1096–1108, 2019.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Tobiasz Cieplinski, Tomasz Danel, Sabina Podlewska, and Stanislaw Jastrzebski. We should at least be able to design molecules that dock well. *arXiv preprint arXiv:2006.16955*, 2020.

Daniel C. Elton, Zois Boukouvalas, Mark D. Fuge, and Peter W. Chung. Deep learning for molecular design—a review of the state of the art. *Mol. Syst. Des. Eng.*, 4:828–849, 2019.

Vendy Fialková, Jiaxi Zhao, Kostas Papadopoulos, Ola Engkvist, Esben Jannik Bjerrum, Thierry Kogej, and Atanas Patronov. Libinvent: reaction-based generative scaffold decoration for in silico library design. *Journal of Chemical Information and Modeling*, 62(9):2046–2063, 2021.

Wenhao Gao, Tianfan Fu, Jimeng Sun, and Connor Coley. Sample efficiency matters: a benchmark for practical molecular optimization. *Advances in Neural Information Processing Systems*, 35: 21342–21357, 2022.

Anna Gaulton, Louisa J Bellis, A Patricia Bento, Jon Chambers, Mark Davies, Anne Hersey, Yvonne Light, Shaun McGlinchey, David Michalovich, Bissan Al-Lazikani, et al. Chembl: a large-scale bioactivity database for drug discovery. *Nucleic acids research*, 40(D1):D1100–D1107, 2012.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

Jeff Guo, Vendy Fialková, Juan Diego Arango, Christian Margreitter, Jon Paul Janet, Kostas Papadopoulos, Ola Engkvist, and Atanas Patronov. Improving de novo molecular design with curriculum learning. *Nature Machine Intelligence*, 4(6):555–563, 2022.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Kexin Huang, Tianfan Fu, Wenhao Gao, Yue Zhao, Yusuf Roohani, Jure Leskovec, Connor W Coley, Cao Xiao, Jimeng Sun, and Marinka Zitnik. Therapeutics data commons: Machine learning datasets and tasks for drug discovery and development. *arXiv preprint arXiv:2102.09548*, 2021.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*, pp. 2323–2332. PMLR, 2018.

Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. *Advances in neural information processing systems*, 29, 2016.

Gabriele Libardi, Gianni De Fabritiis, and Sebastian Dittert. Guided exploration with proximal policy optimization using a single demonstration. In *International Conference on Machine Learning*, pp. 6611–6620. PMLR, 2021.

Jacob Menick, Maja Trebacz, Vladimir Mikulik, John Aslanides, Francis Song, Martin Chadwick, Mia Glaese, Susannah Young, Lucy Campbell-Gillingham, Geoffrey Irving, et al. Teaching language models to support answers with verified quotes. *arXiv preprint arXiv:2203.11147*, 2022.

Rocío Mercado, Tobias Rastemo, Edvard Lindelöf, Günter Klambauer, Ola Engkvist, Hongming Chen, and Esben Jannik Bjerrum. Graph networks for molecular design. *Machine Learning: Science and Technology*, 2(2):025023, 2021.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics*, 9(1):1–14, 2017.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Daniil Polykovskiy, Alexander Zhebrak, Benjamin Sanchez-Lengeling, Sergey Golovanov, Oktai Tatanov, Stanislav Belyaev, Rauf Kurbanov, Aleksey Artamonov, Vladimir Aladinskiy, Mark Veselov, et al. Molecular sets (moses): a benchmarking platform for molecular generation models. *Frontiers in pharmacology*, 11:565644, 2020.

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

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science*, 4(1): 120–131, 2018.

AN Shivanyuk, SV Ryabukhin, A Tolmachev, AV Bogolyubsky, DM Mykytenko, AA Chupryna, W Heilman, and AN Kostyuk. Enamine real database: Making chemical diversity real. *Chemistry today*, 25(6):58–59, 2007.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Morgan Thomas, Robert T Smith, Noel M O'Boyle, Chris de Graaf, and Andreas Bender. Comparison of structure-and ligand-based scoring functions for deep generative models: a gpcr case study. *Journal of cheminformatics*, 13(1):1–20, 2021.

Morgan Thomas, Andrew Boardman, Miguel Garcia-Ortegon, Hongbin Yang, Chris de Graaf, and Andreas Bender. Applications of artificial intelligence in drug design: opportunities and challenges. *Artificial Intelligence in Drug Design*, pp. 1–59, 2022a.

Morgan Thomas, Noel M O'Boyle, Andreas Bender, and Chris De Graaf. Augmented hill-climb increases reinforcement learning efficiency for language-based de novo molecule generation. *Journal of Cheminformatics*, 14(1):1–22, 2022b.

Morgan Thomas, Noel M O'Boyle, Andreas Bender, and Chris De Graaf. Molscore: A scoring and evaluation framework for de novo drug design. 2023.

Morgan Thomas, Mazen Ahmad, Gary Tresadern, and Gianni de Fabritiis. Promptsmiles: Prompting for scaffold decoration and fragment linking in chemical language models. 2024.

Mingyang Wang, Zhe Wang, Huiyong Sun, Jike Wang, Chao Shen, Gaoqi Weng, Xin Chai, Honglin Li, Dongsheng Cao, and Tingjun Hou. Deep learning approaches for de novo drug design: An overview. *Current Opinion in Structural Biology*, 72:135–144, 2022.

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.

Botian Xu, Feng Gao, Chao Yu, Ruize Zhang, Yi Wu, and Yu Wang. Omnidrones: An efficient and flexible platform for reinforcement learning in drone control. *IEEE Robotics and Automation Letters*, 2024.

APPENDIX

A REINFORCEMENT LEARNING ENVIRONMENT

```python
from acegen.rl_env import SMILESEnv
from acegen.vocabulary import SMILESVocabulary
from torchrl.collectors import RandomPolicy

# Create a vocabulary from a list of characters
chars = ["START", "END", "(", ")", "1", "=", "C", "N", "O"]
chars_dict = {char: index for index, char in enumerate(chars)}
vocab = SMILESVocabulary.create_from_dict(chars_dict, start_token="START", end_token="END")

# Create an environment from the vocabulary
env = SMILESEnv(
    start_token=vocab.start_token_index,
    end_token=vocab.end_token_index,
    length_vocabulary=len(vocab),
    batch_size=4, # Number of trajectories to collect in parallel
)

# Create a random policy
policy = RandomPolicy(env.full_action_spec)

# Take an environment step
step_tensordict = env.step(policy(env.reset()))
print(step_tensordict)
```

```
# Output:

TensorDict(
    fields={
        action: Tensor(shape=torch.Size([4]), device=cpu, dtype=torch.int32,
        done: Tensor(shape=torch.Size([4, 1]), device=cpu, dtype=torch.bool),
        next: TensorDict(
            fields={
                done: Tensor(shape=torch.Size([4, 1]), device=cpu, dtype=torch.bool),
                observation: Tensor(shape=torch.Size([4]), device=cpu, dtype=torch.int64),
                reward: Tensor(shape=torch.Size([4, 1]), device=cpu, dtype=torch.float32),
                terminated: Tensor(shape=torch.Size([4, 1]), device=cpu, dtype=torch.bool),
                truncated: Tensor(shape=torch.Size([4, 1]), device=cpu, dtype=torch.bool)},
            batch_size=torch.Size([4]),
            device=cpu,
            is_shared=False),
        observation: Tensor(shape=torch.Size([4]), device=cpu, dtype=torch.int32),
        terminated: Tensor(shape=torch.Size([4, 1]), device=cpu, dtype=torch.bool),
        truncated: Tensor(shape=torch.Size([4, 1]), device=cpu, dtype=torch.bool)},
    batch_size=torch.Size([4]),
    device=cpu,
    is_shared=False)
```

Figure 2: Code example that shows how to create a language-based reinforcement learning environment components for SMILES generation and how to take sampling steps on it with a random policy.

B DEFINE A CUSTOM GENERATIVE POLICY

```python
import torch.nn as nn
from tensordict.nn import TensorDictModule

class ExamplePolicy(nn.Module):
    """Example policy that takes an observation and returns an action."""

    def __init__(self):
        super(ExamplePolicy, self).__init__()

        self.embed = nn.Embedding(
            num_embeddings=len(vocab), embedding_dim=3)

        self.rnn = nn.LSTM(
            input_size=3, hidden_size=3, num_layers=1, batch_first=True)

        self.head = nn.Linear(in_features=3, out_features=10)

    def forward(self, x, recurrent_state=None):
        x = self.embed(x)
        x, recurrent_state = self.rnn(x, recurrent_state)
        x = self.head(x)
        return x, recurrent_state

policy = TensorDictModule(
    ExamplePolicy(),
    in_keys=["observation", "recurrent_state"],
    out_keys=["action"],
)

initial_tensordict = env.reset()
print(initial_tensordict)

updated_tensordict = policy(initial_tensordict)
print(updated_tensordict)
```

```
# Output 1:

TensorDict(
    fields={
        done: Tensor(shape=torch.Size([4, 1]), device=cpu, dtype=torch.bool),
        observation: Tensor(shape=torch.Size([4]), device=cpu, dtype=torch.int32),
        terminated: Tensor(shape=torch.Size([4, 1]), device=cpu, dtype=torch.bool),
        truncated: Tensor(shape=torch.Size([4, 1]), device=cpu, dtype=torch.bool)},
    batch_size=torch.Size([4]),
    device=cpu,
    is_shared=False)

# Output 2:

TensorDict(
    fields={
        action: Tensor(shape=torch.Size([4]), device=cpu, dtype=torch.float32),
        done: Tensor(shape=torch.Size([4, 1]), device=cpu, dtype=torch.bool),
        observation: Tensor(shape=torch.Size([4]), device=cpu, dtype=torch.int32),
        terminated: Tensor(shape=torch.Size([4, 1]), device=cpu, dtype=torch.bool),
        truncated: Tensor(shape=torch.Size([4, 1]), device=cpu, dtype=torch.bool)},
    batch_size=torch.Size([4]),
    device=cpu,
    is_shared=False)
```

Figure 3: Code example showing how to wrap a torch.nn.Module to make it TensorDict-compatible.

C DEFINE SCORING FUNCTIONS

```python
from molscore.manager import MolScore

task = MolScore(
    model_name="example",
    task_config=path/to/molscore/config/MolOpt/Albyterol_similarity.json,
    budget=10,
    output_dir="/tmp",
))

env =  SMILESEnv(
    start_token=vocab.start_token_index,
    end_token=vocab.end_token_index,
    length_vocabulary=len(vocab),
    batch_size=1,
)

data = env.rollout(max_steps=100)
smiles_str = [vocab.decode(smi.numpy()) for smi in data["action"]]
reward = task(smiles_str)
```

Figure 4: An example scoring function created using MolScore.

```python
from rdkit.Chem import AllChem, QED

def evaluate_mol(smiles: str):
    mol = AllChem.MolFromSmiles(smiles)
    if mol:
        return QED(mol)
    else:
        return 0.0

def evaluate_mols(smiles: list):
    return [evaluate_mol(smi) for smi in smiles]

env =  SMILESEnv(
    start_token=vocab.start_token_index,
    end_token=vocab.end_token_index,
    length_vocabulary=len(vocab),
    batch_size=1,
)

data = env.rollout(max_steps=100)
smiles_str = [vocab.decode(smi.numpy()) for smi in data["action"]]
reward = evaluate_mols(smiles_str)
```

Figure 5: An example of custom scoring function that returns a reward based on an input molecule. If the molecule is invalid, no reward is returned (i.e., 0.0).

D PRETRAINING

The training dataset was extracted from ChEMBL28 Gaulton et al. (2012) and filtered to only include molecules with a pChEMBL value greater than 6 from an assay with a confidence value of 8 or higher. This is a proxy for molecules that have likely undergone some medicinal chemistry design and optimisation. The molecules were further standardized, neutralized and filtered to ensure molecules had a logP less than or equal to 4.5, rotatable bond count less than or equal to 7, molecular weight in the range 150 to 650 Da and only contain atoms belonging to the following set $A \in \{C, S, O, N, H, F, Cl, Br\}$. Molecules violating substructure alerts as described in Polykovskiy et al. (2020) were removed. Lastly, to smooth the chemical space distribution, molecules were clustered based on scaffold similarity using ECFP4 fingerprints at a threshold of 0.8 Tanimoto similarity and only the centroids carried forward to the final training dataset of 189,238 unique molecules. This then underwent 10-fold restricted randomized augmentation Arús-Pous et al. (2019) resulting in a final dataset of 1,892,380.

The RNN consisted of an embedding layer of size and 256, 3 layers of GRU cells with a hidden dimension size of 512. The model was trained on the training dataset for 5 epochs using a batch size of 128 and Adam optimizer with a learning rate of 0.001.

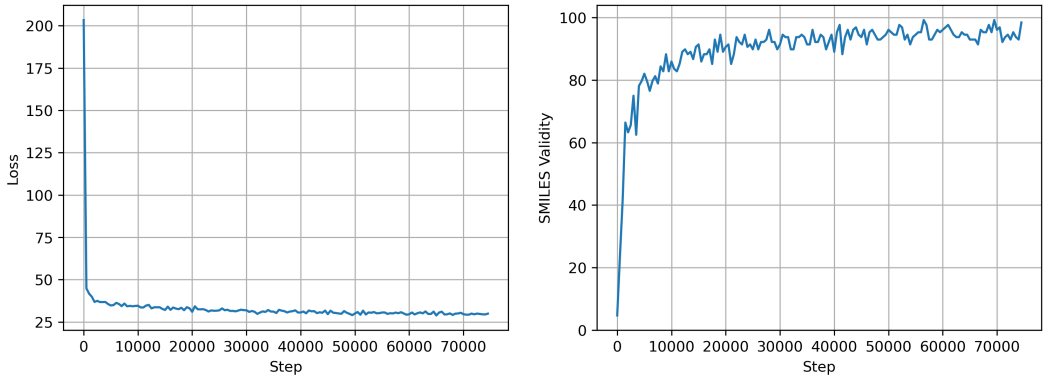

Figure 6: Training loss (i.e., negative log-likelihood) over 5 epochs and ratio of SMILES parsed as valid.

E MolOpt Benchmark

The MolOpt benchmark was run using the MolScore implementation for convenience as it automatically calculates the same metrics as in the original implementation. In addition, MolScore computes the metrics after the removal of undesirable chemistry via the use of a basic chemistry filter (BCF). This filter ensures molecules have a logP less than or equal to 4.5, rotatable bond count less than or equal to 7, molecular weight in the range 150 to 650 Da, only contain atoms belonging to the following set $A \in \{C, S, O, N, H, F, Cl, Br\}$, and do not violate the substructure alerts as described in Polykovskiy et al. (2020).

Metric	Reinvent (default)	Reinvent (MolOpt)	AHC	A2C	PPO	PPOD
Top-1 Avg Score	16.57	**17.73**	16.77	16.35	16.14	17.47
Top-10 Avg Score	15.96	**17.67**	16.13	15.90	15.83	17.36
Top-100 Avg Score	14.90	**17.55**	15.04	15.37	15.60	17.26
Top-1 AUC Score	14.92	16.28	15.06	15.21	15.39	**16.40**
Top-10 AUC Score	13.74	15.88	13.93	14.41	14.86	**16.05**
Top-100 AUC Score	11.98	15.12	12.20	13.20	14.17	**15.43**
B-CF Top-1 Avg Score	16.00	**16.91**	16.13	15.70	15.58	16.75
B-CF Top-10 Avg Score	15.38	**16.77**	15.52	15.19	15.24	16.63
B-CF Top-100 Avg Score	14.24	**16.50**	14.36	14.55	14.96	16.43
B-CF Top-1 AUC Score	14.74	16.00	14.90	14.94	15.03	**16.05**
B-CF Top-10 AUC Score	13.82	15.72	14.06	14.44	14.62	**15.81**
B-CF Top-100 AUC Score	12.44	15.19	12.74	13.68	14.11	**15.38**
B-CF	16.86	14.47	15.96	12.98	**17.52**	15.79

Table 2: Performance summary of algorithms for all metrics measured in this benchmark. Each algorithm ran 5 times with different seeds, and results were averaged over the sum of results per task. B-CF is the fraction of molecules that pass the basic chemistry filter summed over tasks and averaged over seeds.

F Comparison against an existing implementation

We conducted a comparative analysis between our Reinvent implementation and MolOpt's (Guo et al., 2022) implementation, which is accessible on their Github repository. To ensure a fair comparison, we replaced only the prior weights and utilized MolScore for computing scoring functions. Apart from these adjustments, we maintained the use of the original code. Both implementations use the same network architecture and hyperparameters.

Table 3 and Table 4 show that our code is faster and obtains higher values for all metrics with compared to that implementation. The observed performance gaps may stem from subtle idiosyncrasies within the codebase. In reinforcement learning, even minor variations can yield notable discrepancies in outcomes. This underscores the importance of employing standardized, rigorously tested components and foundational building blocks, which offer heightened reliability. Furthermore, we hypothesize the enhanced efficiency can be attributed to leveraging components from TorchRL, a part of the META ecosystem, which inherently provides better optimization and reliability compared to bespoke solutions.

Task	Reinvent MolOpt Code	Reinvent AceGen Code
Albuterol similarity	2:06 mins	**1:01 mins**
Amlodipine MPO	5:05 mins	**3:25 mins**
C7H8N2O2	1:44 mins	**0:42 mins**
C9H10N2O2PF2Cl	1:44 mins	**0:37 mins**

Table 3: Comparison of Reinvent MolOpt and Reinvent AceGen code on a machine with 32 CPUs and a NVIDIA GeForce RTX 4090 GPU for different scoring functions. In each run the algorithm generates and trains on 10K molecules.

Metric	Reinvent MolOpt Code	Reinvent AceGen Code
Avg top 10	15,833	**15,959**
Avg top 100	14,640	**14,896**
AUC top 10	13,597	**13,740**
AUC top 100	11,873	**11,985**
BCF Avg top 10	15,253	**15,380**
BCF Avg top 100	13,966	**14,237**
BCF AUC top 10	13,691	**13,821**
BCF AUC top 100	12,334	**12,436**

Table 4: Comparison of Reinvent MolOpt and Reinvent AceGen on the MolOpt benchmark scoring functions. The results are presented as the sum of all scoring function in the benchmark for different metrics. Each algorithm ran 5 times with different seeds, and results were averaged.