CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Evolution Strategies as an Alternative to Reinforcement Learning in De Novo Molecular Generation

Evaluating Distribution Based Gaussian Evolution Strategies in REINVENT vs. Policy Based Reinforcement Learning on the Practical Molecular Optimization Benchmark

Master's thesis in Computer science and engineering

Malva Eklöv

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Evolution Strategies as an Alternative to Reinforcement Learning in De Novo Molecular Generation

Evaluating Distribution Based Gaussian Evolution Strategies in REINVENT vs. Policy Based Reinforcement Learning on the Practical Molecular Optimization Benchmark

Malva Eklöv

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Evolution Strategies as an Alternative to Reinforcement Learning in De Novo Molecular Generation
Evaluating Distribution Based Gaussian Evolution Strategies in REINVENT vs. Policy Based Reinforcement Learning on the Practical Molecular Optimization Benchmark
Malva Eklöv

Supervisors: Nicklas Österbacka, AstraZeneca
Ross Irwin, AstraZeneca & Chalmers Computer Science and Engineering
Examiner: Ola Engkvist, Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Evolution Strategies as an Alternative to Reinforcement Learning in De Novo Molecular Generation
Evaluating Distribution Based Gaussian Evolution Strategies in REINVENT vs. Policy Based Reinforcement Learning on the Practical Molecular Optimization Benchmark
Malva Eklöv
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

De novo molecular generation examines how computational methods can propose novel drug candidates by scoring generated molecules against desired properties and updating generation toward higher scoring regions. One approach trains a SMILES-based language model and fine tunes it toward such regions. Fine tuning is commonly performed with policy gradient reinforcement learning (RL), as in AstraZenecas highly optimized REINVENT platform. An alternative is evolutionary strategies (ES), where a population of models is created and their parameters are updated to bias generation toward higher scores. This thesis investigates how distribution-based ES compares with RL when implemented in REINVENT and evaluated on the Practical Molecular Optimization benchmark. OpenAI-ES shows near competitive performance on scoring and diversity metrics, whereas variance estimating Natural ES methods perform poorly. Variations of fixed variance ES are explored, and a novel sampling technique that biases generation toward high diversity shows promising performance across multiple metrics.

Keywords: Evolution Strategies, Reinforcement Learning, De Novo Molecular Generation, Molecular AI, Drug Development, Fine-Tuning

Acknowledgements

I would like to thank my supervisors, Ross and Nicklas, for their time, energy, enthusiasm, and patience. Nicklas, your encouragement to pursue this project in my own direction and on my own terms has meant a great deal. This spring has been challenging in many ways, and I am deeply grateful for your support. Your deep knowledge and many quick answers have made the work a lot easier. Ross, you have said many thought provoking and perspective bending ideas. My understand of this subject would be a lot less deep and nuance without your presence.

I would also like to thank Marten Winkler, who likewise completed a Masters thesis on REINVENT. Having a colleague working on related topics was reassuring, and I am grateful for our discussions on project approaches. I am also thankful to Anders Källberg for many stimulating conversations from which new ideas and perspectives emerged.

Finally, my warmest thanks go to my partner, Cecilia. Your steadfast support this spring has been invaluable, and I am grateful for your company and for your keen eye in identifying my spelling mistakes.

Malva Eklöv, Gothenburg, 2026-07-04

Nomenclature

Functions

$\mathcal{F}(\psi)$	Fisher information matrix
$\mathcal{O}(m)$	Scoring function
$\mathrm{AUC}_f(\mathrm{t})$	Area under the curve f
$\mathrm{top}_k(t)$	Mean score of the k best molecules
$\phi(x)$	Activation function
$\mathrm{sim}_x(m_i, m_j)$	Tanimoto similarity using fingerprint x
$D_{\mathrm{KL}}(P \mid\mid Q)$	Kullback-Leibler divergence
$J(\theta)$ / $J(\psi)$	Expected reward (RL) / Expected score (ES)

Operators

$\odot$	Element-wise multiplication
$\tilde{\nabla}$	Natural gradient

Parameters

α	Learning rate
$\alpha_{\log\sigma}$	Learning rate for variance updates
β_1	Decay rate of first moment for Adam
β_2	Decay rate of second moment for Adam
λ	Similarity penalty on oversampling
σ_{es}	Strength of sampling noise = Variance of distribution in ES
σ_{rl}	Weight of scoring function in REINVENT
N_{os}	Molecules initially sampled on oversampling
N_b	Batch size
N_p	Population size
T	Sampling temperature

Sets

$\mathcal{M}$	Space or set of molecules

Variables

ψ	Parameters of distribution on θ
θ	Trainable model parameters
ε	Added sampling noise
m	Molecule (element of $\mathcal{M}$)

Contents

List of Figures

List of Tables

1

Introduction

Developing new drugs is expensive, with estimates from 1.611×10^8 to 4.54×10^9 US dollars in 2019 [1], and time consuming, with discovery to approval taking 12 years on average in the US [2]. Drugs must be sufficiently easy to synthesize, exhibit suitable pharmacokinetics, be safe at therapeutic doses, and show potent, selective target engagement, while meeting practical constraints like stability and solubility. The number of drug like molecules is estimated to be 10^{24} to 10^{60} [3], [4] which makes searching the full space impossible. Consequently, early discovery increasingly relies on generative models and sample efficient optimization to navigate the vast chemical space under realistic evaluation budgets [5].

1.1 Drug Discovery

Drug discovery is a long process which usually begins by identifying a condition with nonexistent or inadequate treatment [6]. The next step is to identify a target, such as a protein, gene, or RNA fragments. This can be done by studying protein or mRNA levels, studying genetic polymorphism, phenotypic screening and a wide range of other approaches. The targets activity is then validated, which can be done *in vitro*, in animal models, or on patients.

The next step is to find hits: molecules that show the desired activity against the target [6]. Approaches include high-throughput screening, where large collections of existing compounds are tested in automated assays; fragment screening, where very small chemical pieces that bind weakly are identified and then combined into larger and more potent molecules; and structure aided drug design, where the three-dimensional structure of the binding pocket guides the design and testing of more suitable compounds. Computer-based approaches are also used, such as virtual screening where an existing library of molecules are tested to find molecules with desired properties, or this thesis' focus: *de novo* generation where novel molecules are created who satisfy the the desired properties.

Promising hits are then iteratively improved and studied in increasing detail [6]. The goal is to increase potency at the target, reduce off-target effects, and improve basic developability (solubility, permeability, and metabolic stability). These leads are refied until sufficient performance in all relevant metrics are reach or it is deemed they can not reach it. Out of potentially hundreds of molecules, only a handful end

as preclinical candidates. Only a small fraction of projects reach this point, with roughly one in ten advancing to a nominated preclinical candidate.

Preclinical candidates advance into formal preclinical and clinical development, a long and costly path [7]. Preclinical work establishes detailed pharmacology and safety profiles, including mandated animal studies (acute and repeat-dose toxicity, safety pharmacology, later carcinogenicity and reproductive toxicology), alongside formulation and manufacturing readiness. If the risk-benefit is deemed acceptable, clinical trials begin. In phase I, healthy volunteers are studied to assess safety, tolerability and human pharmacokinetics; in phase II, patients are enrolled to establish initial efficacy and refine dose-response; in phase III, large trials confirm efficacy and safety across broader, more diverse populations. Only about one in ten candidates that enter the clinical testing reaches approval, with success rates varying by therapeutic area.

1.2 *De Novo* Design

De novo molecular generation is a computational method to identify hits in early stage drug development. Instead of starting from an existing library of molecules, novel molecules are generated and then their desired properties are estimated computationally. Information about the performance of previously generated molecules informs the creation of new ones, guiding the search toward progressively higher scoring molecules. Compared to virtual screening, it allows targeted exploration of relevant regions and is not limited to what is present in a predefined library.

All *de novo* approaches require both a system to represent molecules and a molecule generator capable of iteratively proposing improved candidates. Molecular representations can be text based, leveraging advances in natural language processing, or graph based, which more naturally encode molecular topology [8]. Representations may operate at the atom level, at the fragment level, or via synthetic routes (e.g., reaction based encodings). A variety of generators have been explored, including genetic algorithms [9], variational autoencoders [10], generative adversarial networks [11], normalizing flows [12], diffusion models [13], and reinforcement learning driven policies [14].

This thesis builds on REINVENT [14], which uses a neural network to generate text based molecules in the SMILES format. The network is updated via policy based reinforcement learning to bias generation toward high performing regions of chemical space.

1.3 Evolution Strategies

Generative neural networks are commonly fine tuned with gradient based methods like policy based reinforcement learning. These use the likelihoods (probabilities) of generated strings, in this context text based representations of molecules, to estimate the direction in which the model will perform better.

Evolution Strategies (ES) offer a complementary, likelihood free alternative that treats the model as a black box, which has seen some applications on neural networks [15], [16], [17], [18]. They maintain a population of parameter settings, either as an explicit set of models or implicitly as a distribution over the parameter space, and iteratively update this population based on observed scores so that it shifts toward higher performing models over successive rounds. ES thus act on the parameter space in contrast to gradient based methods which act on action space of adding new tokens to a string.

Applied to neural generators, ES avoid backpropagation through decoding and external evaluators, tolerates discontinuous and sparse rewards, and promotes exploration through population diversity. While ES can require larger evaluation budgets and careful variance control, it provides a robust and practical option when gradients are noisy, likelihoods are unavailable, or policy gradient fine tuning is unstable.

1.4 Research Questions

Evolution Strategies offer a likelihood-free alternative for fine tuning generators in de novo molecular design. Unlike policy-gradient RL, which uses gradients of the log-likelihood of generated token sequences with respect to model parameters, ES estimate search directions from score differences across populations of perturbed parameter vectors. ES have been applied to train neural networks in other domains, but remain underexplored for *de novo* language-model-based generators

This motivate the following research questions:

- Can ES achieve comparable performance to RL as implemented on REINVENT4 on the PMO benchmark?
- Can ES produce more diverse molecules than RL as implemented on REINVENT4?
- Are variance estimating ES useful in the context of fine tuning language models for *de novo* molecular generation?

1.5 Risk Analysis and Ethical Considerations

This project focuses on foundational methodology development for molecular generation with practical applications being far removed. Nevertheless, risks arise from dual use potential and downstream clinical impact.

- Dual use: The generality of REINVENT makes it possible to use it for generation of molecules for chemical warfare and chemical terrorism.
- Patient safety: If model changes bias generation toward compounds with properties that reduce the probability of reaching market or increase the risk of adverse effects, downstream users are harmed.

Since these questions are general for the REINVENT platform, care will be taken to follow AstraZenecas existing policies.

2

Theory

In early drug discovery one often wants to find one or a set of molecules with some desired properties such as activity at a relevant site, low human toxicity, high synthesizability, and high stability. These properties are often approximated by an *scoring function* which for a given molecule m returns a real number, which in this context is constrained to $[0, 1]$.

The central problem of finding a good candidate molecule m^*, part of the space of valid molecules $\mathcal{M}$, is then defined by:

$$m^* \approx \arg\max_{m \in \mathcal{M}} \mathcal{O}(m). \tag{2.1}$$

for a real valued scoring function $\mathcal{O}(\cdot)$. This function is often an arithmetic or geometric mean of several scoring functions [19]:

$$\mathcal{O}(m) = \sum_{i=1}^{N} \omega_i \mathcal{O}_i(m), \quad \sum_{i=1}^{N} \omega_i = 1 \tag{2.2}$$

$$\text{or } \mathcal{O}(m) = \prod_{i=1}^{N} \mathcal{O}_i(m)^{\omega_i}, \quad \sum_{i=1}^{N} \omega_i = 1, \quad \omega_i \geq 0. \tag{2.3}$$

In order to algorithmically solve Eq. 2.1 one needs a computer readable representation to describe molecules, a model to generate candidate molecules, and an update rule to improve the model so it more likely generates good molecules.

2.1 Molecular Representations

The Simplified Molecular Input Line Entry System (SMILES) is a line notation to encode molecular structures in a computer and human readable format [20]. Molecules are represented as single line strings using a compact alphabet and a small set of grammar rules. A SMILES is obtained by linearizing the molecular graph, where atoms are nodes and bonds are edges, via a depth-first traversal.

Elements are described by their atomic letters in square brackets with charge included. For some common organic atoms (B, C, N, O, P, S, F, Cl, Br, I) brackets may be omitted if they have their default valence. Hydrogen atoms are in standard situations not shown. Bonded atoms are normally displayed next to each other, but

can be broken up by branches which are separated by parentheses. Single bonds use no symbol or `-`. Double and triple bond are shown by `=` and `#` respectively. Ring closures are indicated by matching digits placed at two bonded atoms, effectively linearizing the ring. Atoms in an aromatic ring may be written with lower case letters or using `:` for bonds, but aromaticity can in most cases be inferred. Most

Table 2.1: Some example SMILES

Molecule	SMILES
Carbon dioxide	`O=C=O`
Isobutyric acid	`CC(C)C(=O)O`
Cyclohexane	`C1CCCCC1`
Caffeine	`Cn1c(=O)c2c(ncn2C)n(C)c1=OC`
Chemically invalid	`[Na]=C`
Grammatically invalid	`C=O=`

molecules have multiple grammatically valid SMILES owing to how the molecular graph is traversed, including choices of starting atom, ring closure positions, and branch designation. To address this, SMILES can be canonicalized to a standard form using a defined scheme that imposes a deterministic atom ordering and traversal, so that all valid SMILES for the same molecule map to a single canonical string [21].

The validity of a SMILES can be considered on two levels: grammatical validity, whether the string conforms to the SMILES syntax, and chemical validity, whether the encoded structure is chemically reasonable. Chemical validity is often checked by rules based approaches which enforce valence, aromaticity, and stereochemistry heuristics tuned to standard organic chemistry. Rules based approaches therefore mark nonstandard chemistries (e.g., hypervalent maingroup species, organometallic/coordination complexes) as invalid [22].

2.1.1 SMARTS

SMARTS is a system for describing molecular substructures, useful for identifying if a molecule contains a specified motif [23]. SMARTS is based on SMILES but extends it with syntax for wildcard bonds, property qualifiers, and logical operators, enabling a SMARTS to work as a flexible search query.

Table 2.2: Example SMARTS evaluated against 4-chlorophenol (`Oc1ccc(Cl)cc1`)

Substructure	SMARTS	Match (T/F)
Phenol substructure	`Oc1ccccc1`	T
Any halogen	`[F,Cl,Br,I]`	T
Aliphatic carbon with two H	`[CH2]`	F
Aromatic nitrogen with one H	`[nH]`	F
Any bond between any atoms	`[*]~[*]`	T

2.2 Generative Models

Sequences of discrete tokens, such as SMILES or natural written language, can be generated with an *autoregressive generative model*. With a model that provides a conditional distribution $\Pr(t_t|t_{1:t-1})$ for each token t_t given the sequence of previous tokens $t_{1:t-1}$, one can generate a sequence by iteratively sampling the next token from this distribution and appending it to the prefix. Note that the first token is generated from the unconditional distribution $\Pr(t_1)$

The total probability of a sequence $t_{1:\ell}$ can then be written as:

$$\Pr(T) = \Pr(t_1|\emptyset) \prod_{t=2}^{\ell} \Pr(t_t|t_{1:t-1}). \tag{2.4}$$

2.2.1 Tokenization, Vocabulary and Embedding

For a type of data to be generated by an autoregressive generative model the data needs to be tokenized to a finitely sized set of tokens called a *vocabulary* [24]. For written natural language a vocabulary can be made out of words, parts of words or single letters [25].

Models used to build modern language processing systems use vectors as input. For a sequence of tokens, it is common to use a *one-hot encoding*, where each token is represented as a binary vector of the same length as the vocabulary. If a token has index k, its one-hot vector has a 1 in position k and 0 elsewhere. See Fig. 2.1 for an example.

$$x_1 = \begin{matrix} \texttt{C} \\ \texttt{O} \\ \texttt{Cl} \\ \texttt{1} \\ \texttt{(} \\ \texttt{)} \end{matrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, x_2 = \begin{matrix} \texttt{C} \\ \\ \\ \\ \\ \end{matrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, x_3 = \begin{matrix} \texttt{(} \\ \\ \\ \\ \\ \end{matrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, x_4 = \begin{matrix} \texttt{C} \\ \\ \\ \\ \\ \end{matrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, x_5 = \begin{matrix} \texttt{)} \\ \\ \\ \\ \\ \end{matrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, x_6 = \begin{matrix} \texttt{Cl} \\ \\ \\ \\ \\ \end{matrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Figure 2.1: One-hot encoding of 2-chloropropane (`CC(C)Cl`) on the vocabulary `C`, `O`, `Cl`, `1`, `(`, `)`, `Cl`

The one-hot encoding is then in many cases embedded in a higher dimensional space by a linear transformation [26]. The combination of one-hot encoding and embedding can be viewed as mapping each token to a specific learned vector.

2.2.2 Neural Networks

A neural network is a model which transfer information through a set of neurons. A neuron typically takes an input vector x, applies a linear transformation and a non

linear activation function and outputs the resulting scalar value [27]

$$y = \phi(Wx + b). \tag{2.5}$$

The non linear activation function enables neural networks to approximate non linear functions. Neural networks are generally connected in layers so that the outputs of neurons from one layers is the input for all the neurons in the next layer.

Some common activation functions are sigmoid functions (typically the logistic function), the hyperbolic tangent and the Rectified linear unit, see Fig. 2.2, and are all applied element-wise.

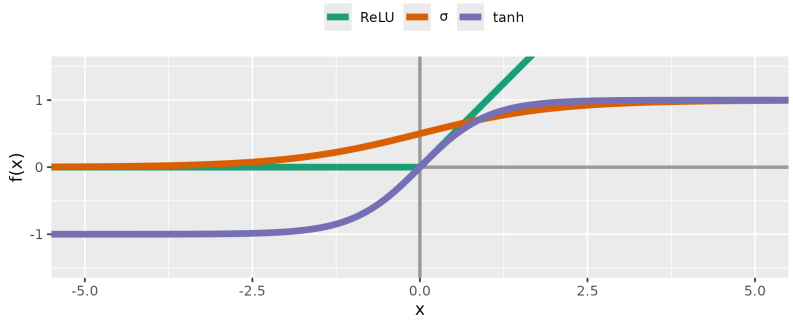

Figure 2.2: Graph of the activation functions $\mathrm{ReLU}(x) = \max(0, x)$, standard logistic function $\sigma(x) = 1/(1 + e^{-x})$ and tanh

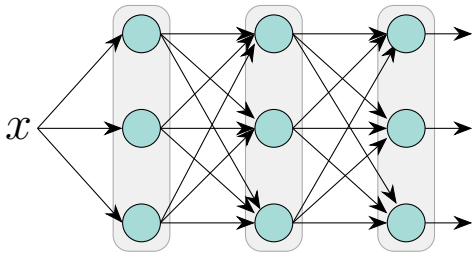

Figure 2.3: A 3 layer (feed-forward) network with 3 neurons in each layer

2.2.2.1 Recurrent Neural Networks

A recurrent neural network (RNN) are built to handle a sequence of vectors $x_1, x_2, \ldots$, where earlier vectors contain relevant information for later ones [27]. This is done be feeding the vectors one by one to the network. Information about earlier vectors are kept in the network by hidden states h according the following equation.

$$\begin{aligned} h_t &= \phi(W_{xh}x_t + W_{hh}h_{t-1} + b_h), \\ y_t &= W_{ho}h_t + b_o \end{aligned} \tag{2.6}$$

This means the network needs an initial hidden state h_0 which is commonly set to a zero vector.

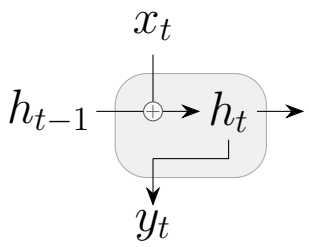

Figure 2.4: Structure of a vanilla RNN layer with input x_t and output y_t as defined by Eq. 2.6

2.2.2.2 Long Short-Term Memory

Vanilla RNNs suffer from vanishing gradients which limits information transfer over long time frames and interferes with model training using gradient based approaches [27]. Long short-term memory (LSTM) networks address this by introducing a dedicated memory cell, c_t, and gating mechanisms that regulate information flow. These gates enable the network to retain or discard information over extended horizons, mitigating vanishing gradients and facilitating stable learning [28].

The memory-cell update is a linear combination of the previous cell state, filtered by a forget gate f_t to keep relevant information, and a candidate cell state $\tilde{c}_t$, scaled by an input gate i_t. The hidden state h_t is then computed via the output gate and used as the layers output.

In the LSTM equations below, σ denotes the logistic sigmoid $1/(1+e^{-x})$ applied element wise and $\odot$ denotes element wise multiplication.

$$i_t = \sigma(W_t x_t + U_t h_{t-1} + b_t) \tag{2.7}$$
$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2.8}$$
$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{2.9}$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2.10}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{2.11}$$
$$h_t = o_t \odot \tanh(c_t) \tag{2.12}$$

LSTMs require initial states h_0 and c_0 which are commonly set to zero vectors [29].

2.2.2.3 Sampling and decoding

The output from one last layer of a neural network consist of a vector $y = (y_1, y_2, \ldots, y_N)$. The output can then be transformed to a distribution over the vocabulary using the softmax function [27]:

$$p_n = \text{softmax}_T(y_n) = \frac{\exp(y_n/T)}{\sum_{k=1}^{N} \exp(y_k/T)} \quad \text{for } n \in 1, \ldots, N \tag{2.13}$$

where T is a parameter called temperature. A low temperature increase the probability assigned to the highest output and in the limit $T \to 0^+$ assigns all probability to the highest output.

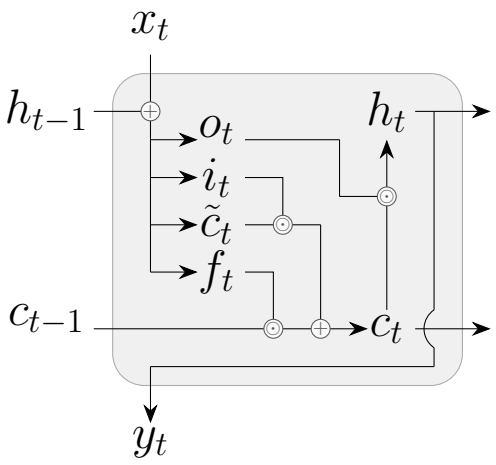

Figure 2.5: Structure of a LSTM layer where the hidden layer is used as output $h_t = y_t$

2.2.2.4 Pretraining

In autoregressive modeling, a network is trained on a large corpus to approximate the data distribution [24]. Ofthen the model maximizes the conditional likelihood of each next token given the observed prefix which is called teacher forcing. For a model with parameters θ and a data set $\mathcal{D}$, the best performing parameter set follows:

$$\theta^* \in \arg\max_{\theta} \mathbb{E}_{t_{1:\ell}\sim\mathcal{D}} \left[\Pr(t_1|\emptyset) \prod_{t=2}^{\ell} \Pr(t_t|t_{1:t-1}) \right]. \tag{2.14}$$

In practice this is done with stochastic gradient ascent. The resulting model from pretraining is called a prior and will with fine tuning be biased, as to more likely generate sequences with desired properties.

2.3 Fine Tuning Methods

This section discusses how to update the parameters θ of a generative model so that the sequences it generate score high against a specified scoring function. It starts with introducing how to optimize a general function $J(\theta)$ using gradients. Then two different frameworks (reinforcement learning and evolution strategies) are introduced to formulate the wanted problem as maximizing a function $J(\theta)$. Expressions for approximating the gradient $\nabla_\theta J(\theta)$ meaning the optimization methods can be applied.

2.3.1 Gradient Ascent

Consider the unconstrained problem:

$$\max_{\theta\in\mathbb{R}^n} J(\theta), \tag{2.15}$$

where J is continuous and its gradient is available. *Gradient ascent* iteratively seeks highervalue points by moving in the direction of steepest ascent of J [30]. Starting

from a initial iterate θ_0, the updates are:

$$\theta_{i+1} \leftarrow \theta_i + \alpha \nabla J(\theta_i) \tag{2.16}$$

where $\alpha \in \mathbb{R}^+$ is the step size.

When exact gradients are unavailable, one can use a gradient estimator with the correct mean [31]. This is called *stochastic gradient ascent* and follows the same update step as standard gradient ascent. While the added noise can encourage broader exploration by helping iterates escape shallow maxima and saddle neighborhoods, it also introduces variance that can slow convergence and make stepsize tuning delicate.

2.3.1.1 Adaptive Moment Estimation (Adam)

Adam can be viewed as a development of standard stochastic gradient ascent that leverages historical gradient information to stabilize and accelerate learning[32]. When J is sufficiently smooth and or estimated $\nabla J(\theta)$ is sufficiently noisy, earlier estimated gradients contain useful information about the ascent direction. Adam incorporates this information by storing first and second moments of the gradient and updating them at each step using weight decay . Alg. 1 illustrate the complete algorithm. The moments are initialized with zeros which introduce a bias by $1 - \beta_1^t$ and $1 - \beta_2^t$ respectively which are corrected in the algorithm.

Algorithm 1 Adam

Require: gradient estimate $\nabla_\theta J(\theta)$, initial parameters θ_0, step size α, decay rates $\beta_1, \beta_2 \in [0, 1)$, stabilizer ϵ

$m_0 \leftarrow 0$, $v_0 \leftarrow 0$
while Not terminated **do**
  $g_t \leftarrow \nabla_\theta J(\theta)(\theta_{t-1})$
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2)\,(g_t \odot g_t)$
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$
  $\theta_t \leftarrow \theta_{t-1} + \alpha\, \hat{m}_t \oslash \left(\sqrt{\hat{v}_t} + \epsilon\right)$
end while

2.3.1.2 Natural Gradient

In optimization over probability distributions, the same family can be written with very different parameterizations. A step that looks small in one set of parameters can correspond to a large change in the actual distribution, and vice versa. As a result, plain Euclidean gradient ascent can behave erratically: it may zigzag along narrow ridges, take tiny steps on flat plateaus, or depend heavily on how the parameters are encoded rather than on how the distribution itself changes. To solve these problems this section introduces a update step which is independent of the distribution's parametrization and show that it can be implemented by only chaining the standard gradient to the *natural gradient* [33].

Steepest ascent with the standard (Euclidean) gradient can be viewed as finding the point on the sphere of radius α in parameter space that maximizes a first order approximation of $J(\psi)$:

$$\max_{\delta\psi} \; J(\psi + \delta\psi) \approx J(\psi) + \delta\psi^\top \nabla_\psi J(\psi), \quad \text{s.t. } \|\delta\psi\|_2 = \alpha. \tag{2.17}$$

This reveals the implicit use of the Euclidean distance and raises the question of whether a different metric might be more suitable. When ψ parametrizes a probability distribution p_ψ over x, a commonly used statistical distance is the *Kullback-Leibler (KL) divergence*:

$$D_{\mathrm{KL}}(P\|Q) = \int P(x) \, \ln\!\left(\frac{P(x)}{Q(x)}\right) dx, \tag{2.18}$$

where P and Q are distributions on the same support [34]. Intuitively, $D_{KL}(P||Q)$ measures the information lost when using P to approximate Q. Using a constraint on the change in KL divergence, we consider:

$$\max_{\delta\psi} \; J(\psi + \delta\psi) \approx J(\psi) + \delta\psi^\top \nabla_\psi J(\psi), \quad \text{s.t. } D_{\mathrm{KL}}(p_{\psi+\delta\psi}\|p_\psi) = \alpha. \tag{2.19}$$

A key advantage is that the resulting steps are invariant to smooth reparameterizations of ψ [33].

For small steps ($\delta\psi \to 0$) and under regularity conditions:

$$D_{\mathrm{KL}}(p_{\psi+\delta\psi}\|p_\psi) = \tfrac{1}{2} \, \delta\psi^\top \mathcal{F}(\psi) \, \delta\psi \; + \; o(\|\delta\psi\|_2^2), \tag{2.20}$$

where the *Fisher information matrix* is:

$$\mathcal{F}(\psi) := \mathbb{E}_{x\sim p_\psi}\Big[\nabla_\psi \ln p_\psi(x) \, \nabla_\psi \ln p_\psi(x)^\top\Big]. \tag{2.21}$$

Using Lagrange multipliers, the first order optimality condition is [35]:

$$\mathcal{F}(\psi) \, \delta\psi = \beta \, \nabla_\psi J(\psi), \tag{2.22}$$

for some scalar $\beta > 0$ set by the KL constraint. If $\mathcal{F}(\psi)$ is invertible, the *natural gradient* is:

$$\tilde{\nabla}_\psi J(\psi) = \mathcal{F}(\psi)^{-1} \, \nabla_\psi J(\psi), \tag{2.23}$$

and steepest ascent with direction $\tilde{\nabla}_\psi J(\psi)$ solves the KL constrained problem above instead of the Euclidean distance constrained one.

2.3.2 Reinforcement Learning

Reinforcement learning (RL) is a training paradigm where an agent learns by interacting with an environment. The problems it tries to solve can often be modeled as a Markov decision process (MDP) [36]. This section first defines the MDP and then uses it to formulate the specific fine tuning method REINVENT.

2.3.2.1 Markov Decision Process

A discrete Markov decision process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where [36], [37]:

- $\mathcal{S}$ is a finite set of states called the state space.
- $\mathcal{A}$ is a finite set of actions called the action space.
- $\mathcal{P}$ is the transition kernel between states such that for current state s, next state s', and action a: $\mathcal{P}(s' \mid s, a) = \Pr[S_{t+1} = s' \mid S_t = s, A_t = a]$.
- $\mathcal{R}$ is a reward kernel that for each (s, s', a) specifies a distribution $R_{t+1} \sim \mathcal{R}(\cdot \mid s, s', a)$.
- $\gamma \in [0, 1]$ is a discount factor, which for terminating tasks can also take the value 1.

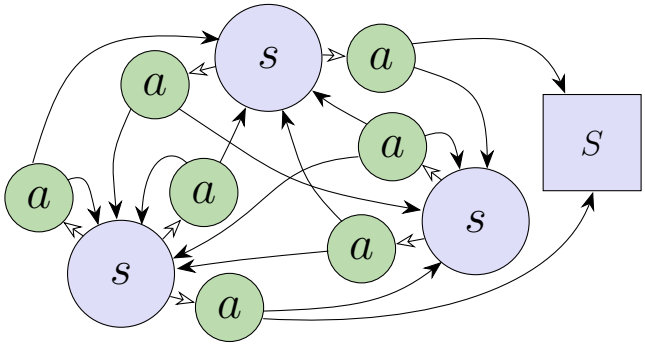

Figure 2.6: State-action graph for a finite MDP with states S (of which one is terminal), actions a, decisions (white arrows) and transition probabilities (black arrows).

An *agent*, acting on the MDP, is described by a policy π which is a distribution over actions given states:

$$\pi(a \mid s) = \Pr[A_t = a \mid S_t = s]. \tag{2.24}$$

At each time step i the agent observes the current environment state $S_t \in \mathcal{S}$ and selects an action $A_t \in \mathcal{A}$ by sampling from $\pi(\cdot \mid S_t)$. The choice of action changes the environment to S_{t+1} by sampling from the transition kernel $\mathcal{P}(\cdot \mid S_t, A_t)$.

An optimal policy maximizes the expected discounted reward starting from the system's initial state s_0:

$$J(\pi) := \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s_0\Big] \quad \text{while following } \pi \tag{2.25}$$

2.3.2.2 Policy Gradients

Finding a good policy π can be done in many ways. Policy gradients use a parameterized stochastic policy π_θ with parameters θ and improve the policy using gradients of the expected reward $J(\theta)$ [38].

A trajectory $\mathcal{T} = (s_0, a_0, s_1, a_1, \ldots)$ is a possible path through the MDP, with probability $p_\theta(\mathcal{T})$ under θ. The expected reward can then precisely be defined, with starting position s_0 implied, as:

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right] = \int \left(\sum_{t=0}^{\infty} \gamma^t R_t(\mathcal{T}_t)\right) p_\theta(\mathcal{T}) \, \mathrm{d}\mathcal{T}. \tag{2.26}$$

where $R_t(\mathcal{T})$ is the reward at step t of following the trajectory $\mathcal{T}$. Using some regularity conditions [39] the gradient is expressed as:

$$\nabla_\theta J(\theta) = \int \left(\sum_{t=0}^{\infty} \gamma^t R_t(\mathcal{T}_t)\right) \nabla_\theta p_\theta(\mathcal{T}) \, \mathrm{d}\mathcal{T} \tag{2.27}$$

$$= \int \left(\sum_{t=0}^{\infty} \gamma^t R_t(\mathcal{T}_t)\right) p_\theta(\mathcal{T}) \nabla_\theta \ln p_\theta(\mathcal{T}) \, \mathrm{d}\mathcal{T}. \tag{2.28}$$

The trajectory density can be factorized as $p_\theta(\mathcal{T}) = p(s_0) \prod_{t=0}^{\infty} \pi_\theta(a_t|s_t) \mathcal{P}(s_{t+1}|s_t, a_t)$. Since the environment and initial state is independent of θ one can simplify $\nabla_\theta \ln p_\theta(\mathcal{T}) = \sum_{t=0}^{\infty} \nabla_\theta \ln \pi_\theta(a_t|s_t)$ and thus:

$$\nabla_\theta J(\theta) = \int \left(\sum_{t=0}^{\infty} \gamma^t R_t(\mathcal{T}_t)\right) p_\theta(\mathcal{T}) \left(\sum_{t=0}^{\infty} \nabla_\theta \ln \pi_\theta(a_t|s_t)\right) \mathrm{d}\mathcal{T} \tag{2.29}$$

$$= \mathbb{E}\left[\left(\sum_{t=0}^{\infty} \gamma^t R_t\right) \left(\sum_{t=0}^{\infty} \nabla_\theta \ln \pi_\theta(a_t|s_t)\right)\right]. \tag{2.30}$$

By using Monte Carlo simulation the gradient can be estimated by a batch of N_b trajectories as:

$$\nabla_\theta J(\theta) \approx \frac{1}{N_b} \sum_{\mathrm{i}=1}^{N_b} \left(\sum_{t=0}^{\infty} \gamma^t R_t^{(i)}\right) \left(\sum_{t=0}^{\infty} \nabla_\theta \ln \pi_\theta(a_t^{(i)}|s_t^{(i)})\right). \tag{2.31}$$

With this, gradients methods such as steepest ascent can be used to optimize θ as to maximize $J(\theta)$.

2.3.2.3 REINVENT

The likelihood of a sequence $t_{1:\ell}$ of tokens $(t_1, t_2, \ldots, t_\ell)$ from a auto regressive generative model is:

$$\Pr(T) = \Pr(t_1|\emptyset) \prod_{t=2}^{\ell} \Pr(t_t|t_{1:t-1}). \tag{2.32}$$

The setup can be modeled as a MDP. The state space comprises all partial and complete sequences that the model can generate. The action space is the set of available tokens. The transition dynamics are deterministic, as the selected token is appended to the current sequence at each step.

Upon termination, a completed sequence $t_{1:\ell}$ is evaluated by a scoring function $\mathcal{O}(T)$ that maps to the interval $[0, 1]$. A reward function can be defined arbitrarily, but a natural choice is to assign zero reward at non-terminal steps and use $\mathcal{O}(T)$ as

the terminal reward. This structure justifies setting the discount factor $= 1$, since rewards only occur at termination.

Reinvent replaces the terminal reward $R_\ell = \mathcal{O}(T)$ with:

$$R_\ell = \frac{\left(\sigma_{\text{rl}}\mathcal{O}(T) + \ln \Pr_P(T) - \ln \Pr_A(T)\right)^2}{\ln \Pr_A(T)} \tag{2.33}$$

where $\sigma_{\text{rl}} \in \mathbb{R}^+$ is a scaling constant, $\Pr_A(T)$ is the likelihood under the agent, and $\Pr_P(T)$ is the likelihood under the untrained prior[14]. Since the prior is pretrained on a realistic data set its inclusion in the numerator rewards "realistic" sequences. Other similar rewards with a stronger theoretical motivation such as $R_\ell = (\mathcal{O}(T) + \sigma_{\text{rl}} \ln \Pr(T)_P)^2$ have been proposed and show similar performance in practice [40].

Using the reward from (2.33) the gradient of episodic reward using $\gamma = 1$ can be formulated as:

$$\nabla J(\theta) \approx \frac{1}{N_b} \sum_{i=1}^{N_b} \frac{\left(\sigma_{\text{rl}}\mathcal{O}(T_i) + \ln \Pr_P(T_i) - \ln \Pr_A(T_i)\right)^2}{\ln \Pr_A(T_i)} \nabla \ln \Pr_A(T_i) \tag{2.34}$$

$$. \tag{2.35}$$

$\ln \Pr_A(T_i)$ is computed using back-propagation on the generator.

2.3.3 Evolution Strategies

Evolution Strategies (ES) maximize a real-valued objective $R(x)$ by maintaining a population over the search variables x and iteratively shifting that population toward higher scoring regions of the search space [41].

A population can be represented either explicitly, by a set of individuals that reproduce and vary, or implicitly [41], by a parameterized distribution whose parameters are updated over time [33]. Many ES algorithms can be formulated from both perspectives. In this thesis, we adopt the distributional view.

For a set of continuous variable x a population on it is distributed according to $p_\psi(x)$ where ψ are the parameters of the distribution [16]. With a specified reward function $R(x)$ the goal is the maximize the expected score:

$$J(\psi) = \mathbb{E}_{x \sim p_\psi}[R(x)]. \tag{2.36}$$

The gradient of can be expressed as:

$$\nabla_\psi J(\psi) = \nabla_\psi \int R(x) p_\psi(x)\, \mathrm{d}x \tag{2.37}$$

$$= \int R(x) \nabla_\psi p_\psi(x)\, \mathrm{d}x \tag{2.38}$$

$$= \int R(x) p_\psi(x) \nabla_\psi \ln p_\psi(x)\, \mathrm{d}x \tag{2.39}$$

$$= \mathbb{E}_{x \sim p_\psi}[R(x) \nabla_\psi \ln p_\psi(x)]. \tag{2.40}$$

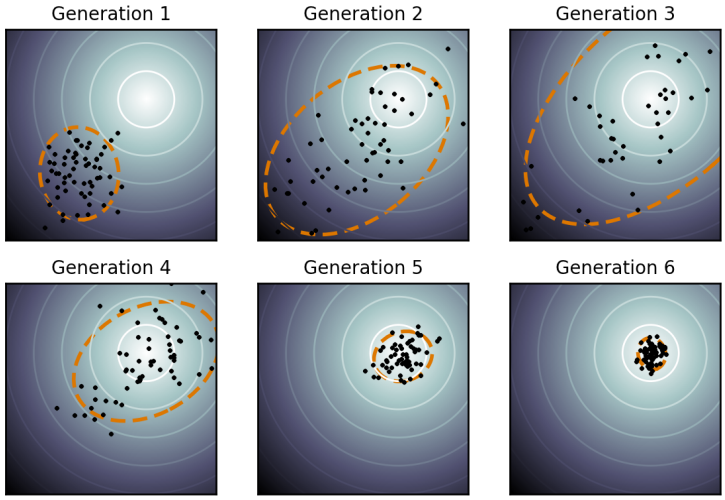

Figure 2.7: Illustration of an Evolution Strategy (CMA-ES) on a simple 2D objective. Contours show equal objective values; points show the population. The distribution tightens and moves toward the global optimum over a few generations. Taken from [42]

where some regularity conditions are needed [39].

$$\nabla_\psi J(\psi) = \mathbb{E}_{x\sim\psi}[R(x)\nabla_\psi \ln p_\psi(x)] \tag{2.41}$$

An evolution strategy can be applied to a generative model by taking the models parameters θ as the search variables. The objective function $R(\theta)$ should measure how well a parameter setting performs under the scoring, i.e., it should approximate the scorings expected score over samples drawn from the model parameterized by θ:

$$R(\theta) = \mathbb{E}_{T\sim p_\theta}\left[\mathcal{O}(T)\right], \tag{2.42}$$

where p_θ denotes the generative distribution and $t_{1:\ell}$ are samples produced by the model with parameters θ [16]. In practice, this expectation can be approximated by Monte Carlo:

$$R(\theta) \approx \frac{1}{N_b}\sum_{i=1}^{N}\mathcal{O}(T_i), \quad \text{with } T_i \sim p_\theta \text{ i.i.d.} \tag{2.43}$$

To apply a evolution strategy one needs to decide which distribution to use. The following sections introduce three different Gaussian versions and derive the needed expressions for it.

2.3.3.1 Isotropic Gaussian ES with fixed variance (F-ES)

The simplest reasonable distribution for p_ψ is a isotropic multivariate Gaussian $p_\mu(x) \sim \mathcal{N}(\mu, \sigma_{\text{es}}^2 I)$ with mean μ and fixed covariance $\sigma_{\text{es}}^2 I$ and is a commonly used approach on neural networks [15], [16].

The log gradient is:

$$\nabla_\mu \ln p_\mu(x) = (x-\mu)/\sigma_{\text{es}}^2 = \varepsilon/\sigma_{\text{es}}, \tag{2.44}$$

where $\varepsilon = (x-\mu)/\sigma_{\text{es}} \sim \mathcal{N}(0,1)$. The update rule then become:

$$\nabla_\mu J(\mu) = \frac{1}{\sigma_{\text{es}}} \mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}[R(\mu + \sigma_{\text{es}}\varepsilon) \odot \varepsilon]. \tag{2.45}$$

Note the Fisher information matrix has entries:

$$\mathcal{F}_{\mu_i,\mu_j} = \mathbb{E}\left[\frac{\varepsilon_i}{\sigma_{\text{es},i}} \frac{\varepsilon_j}{\sigma_{\text{es},j}}\right] = \delta_{i,j} \frac{1}{\sigma_{\text{es},i}\sigma_{\text{es},j}} = \delta_{i,j} \frac{1}{\sigma_{\text{es},i}^2}, \tag{2.46}$$

and thus its inverse is:

$$\mathcal{F}^{-1}(\mu) = \sigma_{\text{es}} I. \tag{2.47}$$

A natural gradient in this setting reduces to a component-wise linear rescaling of the Euclidean gradient by the fixed covariance. Because step size in steepest ascent is a user chosen scalar, this rescaling can be absorbed into the step size, so switching to the natural gradient yields the same update direction and an equivalent method under an appropriate learning rate choice.

2.3.3.2 Diagonal Gaussian ES (D-ES)

Using a diagonal multivariate Gaussian $p_{\mu,\rho}(x) \sim \mathcal{N}(\mu, \text{diag}(e^{2\rho}))$ where $\mu, \rho \in \mathbb{R}^d$, information about spread can be learned. Using $e^\rho = \sigma_{\text{es}}$ can help with stability.

The partial log derivatives can be calculated as:

$$\frac{\partial}{\partial_{\mu_i}} \ln p_{\mu,\rho}(x) = \frac{x_i - \mu_i}{e^{2\rho_i}} = \frac{\varepsilon_i}{e^{\rho_i}} \tag{2.48}$$

$$\frac{\partial}{\partial_{\rho_i}} \ln p_{\mu,\rho}(x) = \frac{(\mu_i - x_i)^2}{e^{2\rho_i}} - 1 = \varepsilon^2 - 1. \tag{2.49}$$

where $\varepsilon_i = (x_i - \mu_i)/e^{2\rho_i} \sim \mathcal{N}(0,1)$. The update rule then become:

$$\nabla_\mu J(\mu,\rho) = \mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}[R(\mu + e^\rho \odot \varepsilon) \odot \varepsilon] \odot \text{diag}(e^{-\rho}) \tag{2.50}$$

$$\nabla_\rho J(\mu,\rho) = \mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}[R(\mu + e^\rho \odot \varepsilon)(\varepsilon^2 - 1)]. \tag{2.51}$$

Since the problem is about optimizing the parameters of a distribution the framework of natural gradients is useful, see Section 2.3.1.2. This method is closely related to the Separable Natural Evolution Strategy [33]. To derive the natural gradients one needs the Fisher information matrix, which has entries:

$$\mathcal{F}_{\mu_i,\mu_j} = \mathbb{E}\left[\frac{\varepsilon_i}{e^{\rho_i}} \frac{\varepsilon_j}{e^{\rho_j}}\right] = \delta_{i,j} e^{-\rho_i\rho_j} = \delta_{i,j} e^{-2\rho_i} \tag{2.52}$$

$$\mathcal{F}_{\rho_i,\rho_j} = \mathbb{E}\left[(\varepsilon_i^2 - 1)(\varepsilon_j^2 - 1)\right] = \delta_{i,j} 2 \tag{2.53}$$

$$\mathcal{F}_{\mu_i,\rho_j} = \mathbb{E}\left[\frac{\varepsilon_i}{e^{\rho_i}}(\varepsilon_j^2 - 1)\right] = 0. \tag{2.54}$$

And thus the inverse is:

$$\mathcal{F}^{-1}(\mu, \rho) = \begin{pmatrix} \text{diag}(e^{2\rho}) & 0 \\ 0 & \frac{1}{2}I \end{pmatrix}. \tag{2.55}$$

The natural gradient then become:

$$\tilde{\nabla}_{\mu_i} J(\mu, \rho) = \text{diag}(e^{2\rho}) \odot \nabla_\mu J(\mu, \rho) \tag{2.56}$$
$$\tilde{\nabla}_\rho J(\mu, \rho) = 2\nabla_\rho J(\mu, \rho). \tag{2.57}$$

2.3.3.3 Isotropic Gaussian ES with varying variance (S-ES)

From the diagonal Gaussian ES versions with simpler variance structures can be derived. When using a single variance for all parameters, $p_{\mu,\rho}(x) \sim \mathcal{N}(\mu, e^\rho I)$ where $\mu \in \mathbb{R}$ and $\rho \in \mathbb{R}$, the update rule can be expressed as:

$$\nabla_\mu J(\mu, \rho) = e^{-\rho} \mathbb{E}_{\varepsilon \sim \mathcal{N}(0,I)}[R(\mu + e^\rho \odot \varepsilon)] \tag{2.58}$$
$$\nabla_\rho J(\mu, \rho) = \mathbb{E}_{\varepsilon \sim \mathcal{N}(0,I)}[R(\mu + e^\rho \odot \varepsilon)(\varepsilon^2 - 1)]. \tag{2.59}$$

The natural gradient become:

$$\tilde{\nabla}_{\mu_i} J(\mu, \rho) = e^{2\rho} \nabla_\mu J(\mu, \rho) \tag{2.60}$$
$$\tilde{\nabla}_\rho J(\mu, \rho) = 2\nabla_\rho J(\mu, \rho). \tag{2.61}$$

2.4 Evaluation Metrics

This section derives metrics to evaluate the performance of a fine tuning on a scoring function.

2.4.1 Metrics against Oracle

Finding molecules which score high against the scoring function is desirable. This can be quantified with $\text{top}_k(t)$ which is the average score of the k highest scoring molecules found at or before time point t.

It is also desirable that high scoring molecules can be found early which can be quantified by $\text{AUC}_{\text{top}_\text{k}}(t)$ where the area under the curve for a function $f(x)$ is defined by:

$$\text{AUC}_f = \frac{1}{t} \int_0^\tau f(\tau)\, \mathrm{d}\tau \tag{2.62}$$

2.4.2 Diversity

A good set of high scoring molecules is diverse, as they are likely less correlated in properties not captured by the scoring function (such as synthesizability, manufacturability & human toxicity). There is a great number of ways to measure chemical diversity of which this thesis use a few simple ones.

2.4.2.1 Internal Diversity

Internal diversity characterizes how varied a set of molecules is with respect to a chosen similarity metric. For a pairwise similarity metric $\text{sim}(\cdot, \cdot)$ and a set of molecules M the internal diversity is defined as:

$$\text{Internal diversity}(\mathcal{M}) = \frac{1}{|\mathcal{M}|} \sum_{1 \leq i < j \leq \mathcal{M}} 1 - \text{sim}(m_i, m_j). \tag{2.63}$$

The pairwise similarity between m_1, m_2, each with a set of features F_1, F_2, can be described by the Jaccard index (in computational chemistry often called Tanimoto similarity) [43]:

$$J(F_1, F_2) := \frac{|F_1 \cap F_2|}{|F_1 \cup F_2|}. \tag{2.64}$$

Features represent substructures of the molecule in some way [43]. For example the features of Extended-Connectivity Fingerprints (ECFP) are atoms, atoms including its neighbors and continuing up to a specified radius [44], see Fig 2.8. Commonly used are ECF4 and ECF6 which use radii 2 and 3 respectively (diameter 4 and 6).

In practice each feature is converted into a integer using hashing and the integers are placed into a binary vector called a fingerprints. By storing fingerprints internal diversity can be quickly computed.

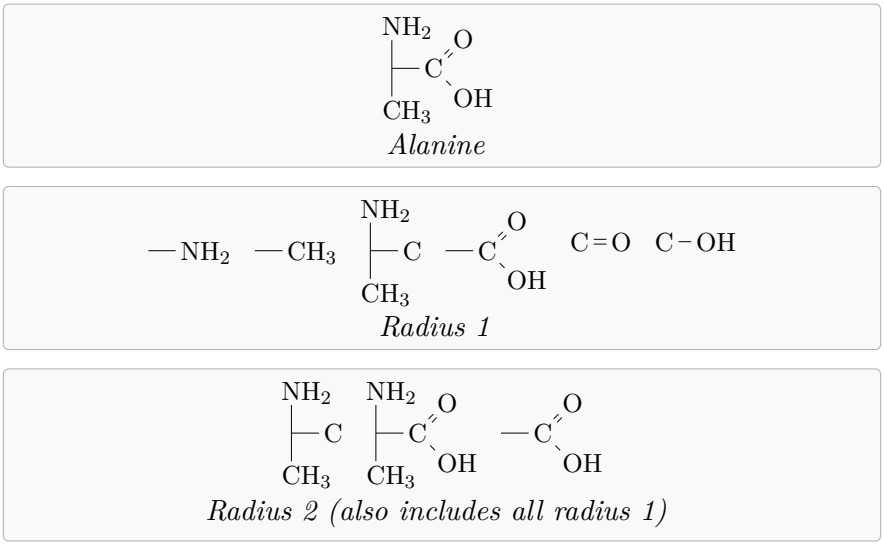

Figure 2.8: Radius 1 and 2 features of Alanine used by ECFP to produce fingerprints

2.4.2.2 Scaffold Diversity

Scaffold diversity counts the number of unique scaffolds in a set of molecules. Scaffolds are the main structure of a molecule that remain after pruning away peripheral substituents, providing a consistent core for grouping related compounds. One common scaffold type is the Murcko (Bemis-Murcko) scaffold, where the molecule is

reduced to its ring systems and the linkers connecting them, with side chains removed while preserving atom identities and bond orders in the core [45]. Under this definition, molecules that share the same heteroatom pattern, ring fusion, and linker chemistry map to the same scaffold.

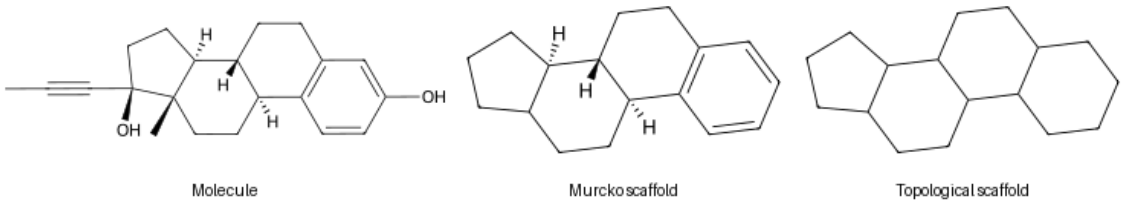

Figure 2.9: Murcko and topological Scaffolds of Estrogen

A topological scaffold can be based on a Murcko scaffold where all information about atom and bond type is removed.

2.4.3 Similarity to Prior

A prior is typical trained on a real set of molecules so for a fined tuned model to be similar to its prior is desirable. This can be measured by the *Kullback-Leibler (KL) divergence*, see Section 2.3.1.2 for definition, on a set of sampled molecules.

3

Methods

This section presents the setup of the system, specifics of fine tuning methods and how they where tested.

3.1 Model Setup and Structure

3.1.1 Vocabulary

In this project, SMILES are split into tokens according to Tab. 3.1 where only the relevant subset of available SMILES symbols are included [46]. The specific tokens are a consequence of the used prior. The tokens are the common symbols found in the training data.

Table 3.1: Vocabulary used for SMILES tokens

Category	Tokens
Start/Stop	`^`, `$`
Bonds	`=`, `-`, `#`
Ring closures	`1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `%10`
Elements (aliphatic)	`C`, `N`, `O`, `S`, `F`, `Cl`, `Br`
Aromatics	`c`, `n`, `o`, `s`
Bracketed ions/atoms	`[N+]`, `[N-]`, `[O-]`, `[S+]`, `[n+]`, `[nH]`
Parentheses	`(`, `)`

3.1.2 Structure of Generative Model

The generative model used in this project follows the structure of Fig. 3.1 [46]. A 34 dimensional one-hot encoding of the this far generated SMILES string is expanded to 256 dimension by a linear projection, which is then sent through three LSTM layers. The resulting vector is linearly down-projected back to 34 dimensions and then transformed to probabilities by a softmax function. The probability distribution is sampled once and the corresponding token is added to the end of the generated SMILES string. The process is repeated until a stop token is sampled. The model has thus in total 5 805 602 trainable parameters.

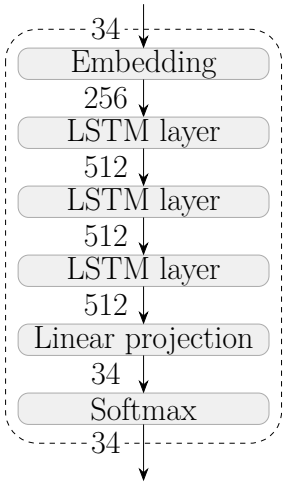

Figure 3.1: Structure of a complete generative model with the dimensionality of vectors between each layer shown. Input is a one-hot-encoded token and output is a probability distribution for the next token. The LSTM layers maintain context from earlier tokens, enabling long-range information retention.

3.1.3 Pretraining

The pre-trained model from REINVENT2 was used in this thesis [46].

3.2 Reinforcement Learning

The existing REINENT4 codebase was used for reinforcement learning. The core of the algorithm is shown in Alg. 2. The only significant modification to the code was temperature regulation and the ability to do over-sampling.

Algorithm 2 REINVENT

Require: Policy model with parameters θ_0, scoring function $\mathcal{O}(\cdot)$, total iterations T, batch size N_b, learning rate α, stabilizer ϵ

1: **for** $t = 1$ to T **do**
2: Sample SMILES $\{T_i\}_i^{N_b} \sim p(\cdot|\theta_{t-1})$ (gets prior and model likelihoods)
3: Compute scores $O_i \leftarrow \mathcal{O}(T_{(i)})$
4: Estimate gradient $\nabla_\theta J \leftarrow \frac{1}{N_b} \sum_{i=1}^{N_b} \frac{\left(\sigma_{\text{rl}}\mathcal{O}(T_i) + \ln \Pr_P(T_i) - \ln \Pr_A(T_i)\right)^2}{\ln \Pr_A(T_i)} \nabla \ln \Pr_A(T_i)$
5: **end for**

The algorithm always use Adam with standard PyTorch settings $\beta_1 = 0.9, \beta_2 = 0.999$ [47].

The REINVENT4 codebase have multiple variations which are often beneficial, mainly diversity filters which lessen the reward for new SMILES similar to already sampled ones and replay memory which add highly scoring SMILES to the scoring set as a cheap off-policy modification [14]. These where not tested in this project.

3.3 Evolution Strategy

The three evolution strategies introduced in Sec. 2.3.3 were implemented on the REINVENT4 codebase. Their structures are shown in Alg. 3, 4 and 5 respectively. All three normalize the scores by setting mean to 0 and standard deviation to 1, following the F-ES algorithm from [15].

Some variations which can be applied to any of the three algorithms are introduced in the following subsections.

Algorithm 3 F-ES Algorithm with Score Normalization

Require: Prior model with parameters θ_0, scoring function $\mathcal{O}(\cdot)$, total iterations T, population size N_p, batch size N_b, noise scale σ_{es}, learning rate α, stabilizer ϵ

for $t = 1$ to T **do**
  for $n = 1$ to N_p **do**
    Sample noise $\varepsilon_n \sim \mathcal{N}(0, I)$
    Sample SMILES $\{T_i\}_i^{N_b} \sim p(\cdot|\theta_{t-1} + \sigma_{\text{es}}\varepsilon)$
    Compute score $R_n \leftarrow \frac{1}{N_b}\sum_{i=1}^{N_b}\mathcal{O}(T_i)$
  end for
  Normalize scores $R_n \leftarrow \dfrac{R_n - \text{mean}(R)}{\text{std}(R) + \epsilon} \quad \forall n$
  Estimate gradient $\nabla_\theta J \leftarrow \frac{1}{N_p\sigma_{\text{es}}}\sum_{n=1}^{N} S_n\,\varepsilon_n$
  Update model parameters $\theta_t \leftarrow \theta_{t-1} + \alpha\nabla_\theta J$
end for

Algorithm 4 Natural S-ES Algorithm with Score Normalization

Require: Prior model with parameters θ_0, scoring function $\mathcal{O}(\cdot)$, total iterations T, population size N_p, batch size N_b, initial noise scale σ_{es}, learning rate α, variance learning rate $\alpha_{\log\sigma}$ stabilizer ϵ

$\sigma \leftarrow \sigma_{\text{es}}$
for $t = 1$ to T **do**
  for $n = 1$ to N_p **do**
    Sample noise $\varepsilon_n \sim \mathcal{N}(0, 1)$
    Sample SMILES $\{T_i\}_i^{N_b} \sim p(\cdot|\theta_{t-1} + \sigma\varepsilon)$
    Compute score $R_n \leftarrow \frac{1}{N_b}\sum_{i=1}^{N_b}\mathcal{O}(T_i)$
  end for
  Normalize scores $R_n \leftarrow \dfrac{R_n - \text{mean}(R)}{\text{std}(R) + \epsilon} \quad \forall n$
  Estimate gradient $\begin{cases}\tilde{\nabla}_\theta J \leftarrow \frac{\sigma}{N_p}\sum_{n=1}^{N} S_n\,\varepsilon_n \\ \tilde{\nabla}_\rho J \leftarrow \frac{2}{N_p}\sum_{n=1}^{N} S_n\,(\varepsilon_n^2 - 1)\end{cases}$
  Update model parameters $\begin{cases}\theta_t \leftarrow \theta_{t-1} + \alpha\tilde{\nabla}_\theta J \\ \sigma \leftarrow \sigma\exp(\alpha_{\log\sigma}\tilde{\nabla}_\rho J)\end{cases}$
end for

Algorithm 5 Natural D-ES Algorithm with Score Normalization

Require: Prior model with parameters θ_0, scoring function $\mathcal{O}(\cdot)$, total iterations T, population size N_p, batch size N_b, initial noise scale σ_{es}, learning rate α, variance learning rate $\alpha_{\log\sigma}$ stabilizer ϵ

$\sigma_i \leftarrow \sigma_{\text{es}} \quad \forall i \in$ model parameters
for $t = 1$ to T **do**
  for $n = 1$ to N_p **do**
    Sample noise $\varepsilon_n \sim \mathcal{N}(0, I)$
    Sample SMILES $\{T_i\}_i^{N_b} \sim p(\cdot|\theta_{t-1} + \text{diag}(\sigma) \odot \varepsilon)$
    Compute score $R_n \leftarrow \frac{1}{N_b} \sum_{i=1}^{N_b} \mathcal{O}(T_i)$
  end for
  Normalize scores $R_n \leftarrow \dfrac{R_n - \text{mean}(R)}{\text{std}(R) + \epsilon} \quad \forall n$
  Estimate gradient $\begin{cases} \tilde{\nabla}_\theta J \leftarrow \frac{1}{N_p} \text{diag}(\sigma) \odot \sum_{n=1}^{N} S_n \, \varepsilon_n \\ \tilde{\nabla}_\rho J \leftarrow \frac{2}{N_p} \sum_{n=1}^{N} S_n \, (\varepsilon_n^2 - 1) \end{cases}$
  Update model parameters $\begin{cases} \theta_t \leftarrow \theta_{t-1} + \alpha \tilde{\nabla}_\theta J \\ \sigma \leftarrow \sigma \exp(\alpha_{\log\sigma} \tilde{\nabla}_\rho J) \end{cases}$
end for

3.3.1 Antithetic Sampling

Antithetic sampling may be applied to model perturbation in a evolution strategy so that for noise ε both $\theta + \varepsilon$ and $\theta - \varepsilon$ is tested. The intent is to reduce variance of the Monte Carlo estimation.

3.3.2 Rank Transformation

Instead of oracle score results $\{R_1, R_2, \ldots, R_{N_p}\}$ one may use their ranks instead. The intent is the reduce the effect outliers and a skewed distribution have on the gradient estimate.

3.3.3 Adam

The Adam optimizer may be used instead of a standard stochastic gradient ascent. For S-ES and D-ES ρ a indepen det Adam otptimizer may be used.

3.3.4 Oversampling

A novel approach to encourage diversity of selected SMILES was developed and implemented. At each sampling step N_{os} SMILES were sampled at standard temperature $T = 1$. From this candidate set, one sample at a time was selected using reweight probabilities defined by:

$$p_{\text{reweight}}(m) = \text{softmax}_T \left(p(m)(1 - M(m)^\lambda \right), \tag{3.1}$$

where $M(\cdot)$ is a similarity metric defined by:

$$M(m_i) = \max_{m_j \in \mathcal{M}_{\text{selected}}} \text{sim}_{\text{ECF4}}(m_i, m_j) \tag{3.2}$$

on the set of already selected molecules $\mathcal{M}_{\text{selected}}$, and λ is a penalty parameter for similarity. The selection is repeated until the wanted batch size N_b is selected. Alg. 6 defines the complete algorithm.

Algorithm 6 Batch selection with NLL prioritization under Tanimoto similarity control

Require: Set of candidates $C = \{c_1, \ldots, c_N\}$ with attributes SMILES(c) and nll(c); threshold thr $\in [0, 1]$, batch size $N_b \in \mathbb{N}^+$, temperature $T \in \mathbb{R}^+$, similarity penalty $\lambda \in \mathbb{R}^+$

Ensure: $|C| \geq \text{N}_\text{b}$

$S \leftarrow \emptyset$ ▷ selected candidates
for all $c \in C$ **do**
  $M(c) \leftarrow 0.0$ ▷ max seen Tanimoto similarity for c
end for
while $|S| < \text{N}_\text{b}$ **and** $C \neq \emptyset$ **do**
  for all $c \in C$ **do**
    $p(c) \leftarrow \frac{\exp(-\text{nll}(c)/T)(1-M(c)^\lambda}{\sum_{c \in C} \exp(-\text{nll}(c)/T)(1-M(c)^\lambda}$ ▷ reweight probabilities
  end for
  $s \sim \text{Categorical}(p(\cdot))$ over C ▷ draw one candidate
  $S \leftarrow S \cup \{s\}$
  $C \leftarrow C \setminus \{s\}$
  for all $c \in C$ **do**
    $M(c) \leftarrow \max(M(c),\ \text{sim}_{\text{ECF4}}(\text{SMILES}(s), \text{SMILES}(c)))$
  end for
end while
return S

3.4 Evaluation of Fine Tuning

The PMO benchmark was used to compare performance which measures area under the curve of top10 scoring molecules over 10000 unique oracle calls for a specified set of oracles [19]. Most oracles are from Guacamol [8] of which nearly all measure similarity to a specific molecule. Outside of Guacamol DRD2 [48], GSk3β [49] and JNK3 [49] which estimate binding to a specific site are also included in PMO. The definitions of all oracle function are found in Appendix A.

4

Results

4.1 Complementary Study of Parameters and Model Assumptions for Evolution Strategies

The equal variance gaussian distributed assumption on a isotropic gaussian ES was investigated on the prior. Fig. 4.1 illustrates how the weights from each layer is distributed. The standard deviation is in the range 1.26 to 0.37 and the shapes are close to a Gaussian distribution, except for the embedding weights.

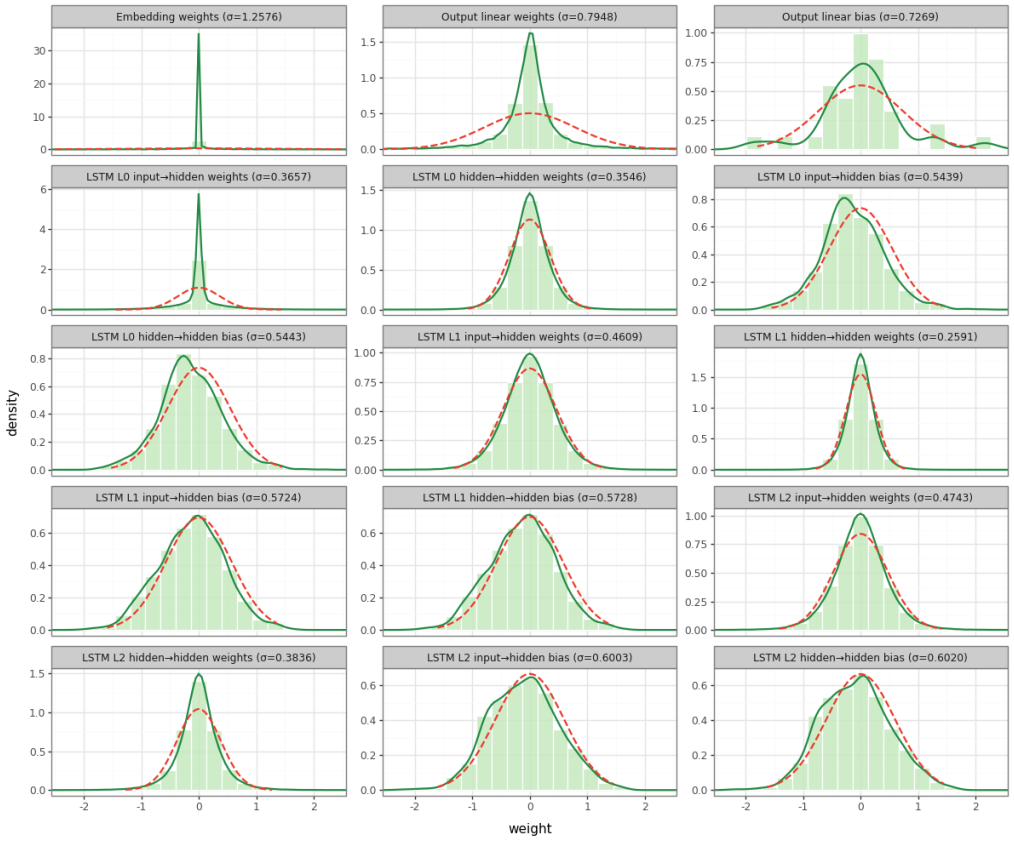

Figure 4.1: Weight distributions of the prior per layer. Each facet shows a histogram of weights, an overlaid kernel density estimate (green), and a dashed Normal(0, σ) curve using the layers empirically estimated σ.

The effect of perturbing the prior with $\mathcal{N}(0, \sigma_{\text{es}}^2)$ distributed noise was tested for varying σ_{es} and sampling temperatures. 10^4 SMILES were independently generated for each setup condition. Fig. 4.2 show a clear change in kl divergence at $\sigma_{\text{es}} \approx 0.1$ independent of temperature. The overlap of SMILES found against a set of molecules generated from a unperturbed model depend greatly on temperature with $0.001 < \sigma_{\text{es}} < 0.03$ being range of large change. These results motivate the hyperparameter tuning range $\sigma_{\text{es}} = [0.001, 0.01, 0.1]$.

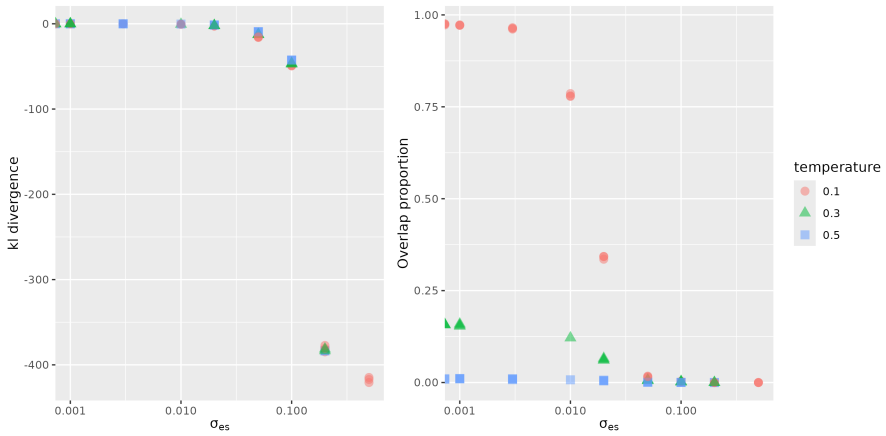

Figure 4.2: Test of how perturbing the prior with σ_{es} changes the distribution by measuring kl divergence against the prior (left) and sample weighted proportion of SMILES generated by prior and perturbed model (right). Each SMILES is sampled from an independently perturbed model

4.2 Hyperparameter Tuning

The limited application of ES on *de novo* molecular generation with neural networks means suitable ranges for hyperparameters and their importance are largely unknown. This section performs hyperparameter tuning and investigate how different hyperparameters affect performance.

The PMO benchmark does hyperparameter tuning on the scoring functions Perindopril and Zaleplon using three runs each and then selects the hyperparameter set which performs best on average on the two [19]. Both Perindopril and Zaleplon are similarity based scoring function. To check if performance changes when tuning against a non-similarity based oracle hyperparameter tuning was also done against JNK3.

4.2.1 Reinforcement Learning

Hyperparameter tuning was done on hyperparameters batch size N_b, sampling temperature T, learning rate α and weight of scoring function σ_{rl}. The results are summarized in Fig. 4.3.

The ideal temperature is around REINVENTS default value 1.0 for all three oracles.

The other hyperparameters seem less impactful except on JNK3 where high α and low σ_{es} is beneficial.

The highest performing hyperparameter sets are shown in Fig. 4.1 and are similar on the three oracles.

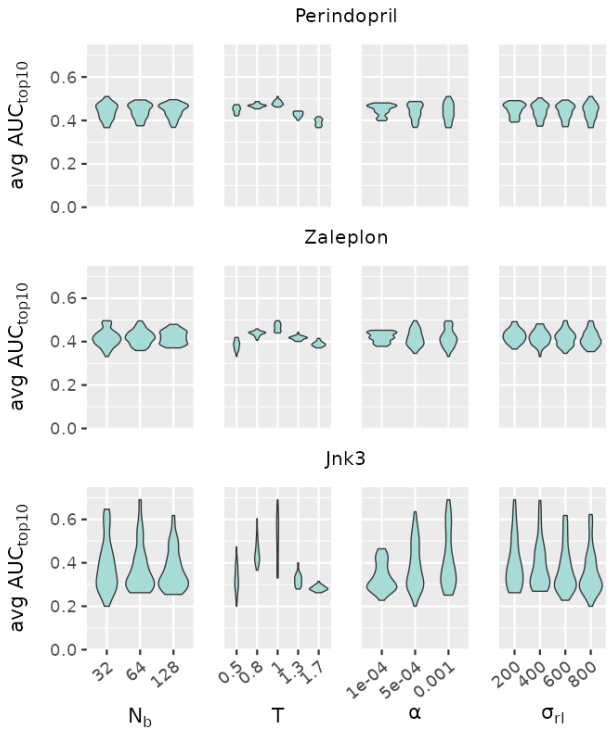

Figure 4.3: RL average performance over three runs for hyperparameters batch size N_b, sampling temperature T, learning rate α and weight of scoring function σ_{rl} on the scoring functions Perindopril, Zaleplon and Jnk3.

Table 4.1: Best hyperparameters for RL

Setup	N_b	T	α	σ_{rl}	$\text{AUC}_{\text{top10}}$
Perindopril	32	1	1×10^{-3}	600	0.510
Zaleplon	32	1	5×10^{-4}	600	0.496
Avg Per. Zal.	32	1	1×10^{-3}	800	
Jnk3	64	1	1×10^{-3}	200	0.691

4.2.2 Evolution Strategy

F-ES was tested with hyperparameters batch size N_b, sampling temperature T, population size N_p, sampling noise σ_{es} and learning rate α. It was also tested if doing antithetic sampling and or rank transformation were beneficial. The results are summarized in Fig. 4.4.

Most hyperparameters have small impact on on performance. Jnk3 seem most sensitive to hyperparameter settings. A small batch size seem slightly beneficial on all

three scoring functions. The middle values for temperature, population size sampling noise and step length was best on Jnk3. Not doing antithetic sampling or rank transformation is good on Jnk3 and not important on Perindopril and Zaleplon.

The highest performing hyperparameter sets are shown in Fig. 4.2.

Table 4.2: Best hyperparameters for F-ES

Setup	N_b	T	N_p	σ_{es}	α	Anithetic	Rank trans.	$\text{AUC}_{\text{top10}}$
Perindopril	4	0.1	30	0.01	5×10^{-4}	FALSE	TRUE	0.501
Zaleplon	4	0.3	30	0.01	5×10^{-4}	TRUE	TRUE	0.449
Avg Per. Zal.	4	0.1	30	0.01	5×10^{-4}	FALSE	TRUE	
JNK3	4	0.3	30	0.01	5×10^{-4}	FALSE	FALSE	0.510

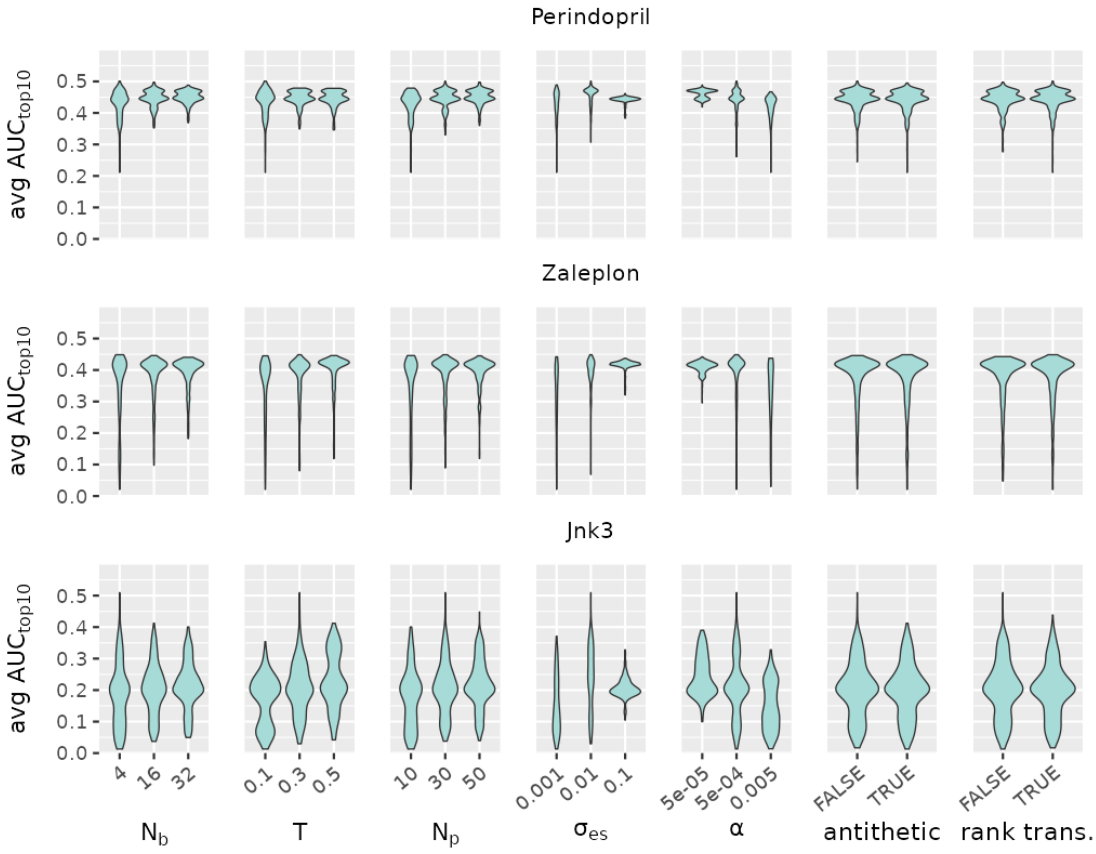

Figure 4.4: F-ES average performance over three runs for hyperparameters batch size N_b, sampling temperature T, learning rate α and weight of scoring function σ_{rl} on the scoring functions Perindopril, Zaleplon and JNK3 .

4.2.2.1 F-ES with Adam

Adam introduces two new parameters when used on F-ES. The decay rate of first moment β_1 which was set to the commonly used value 0.9 [47] and decay rate of

second moment β_2 where a hyperparameter sweep was done. Batch size N_b, temperature T and testing rank transformations were also added to the hyperparmeter sweep. Population size and sampling noise were fixed to the best values from F-Es without Adam beeing 30 and 0.01 respectively. Antithetic sampling was turned off. The results are summarized in Fig. 4.5.

The values of β_2, in the tested range 0.9 to 0.999, were unimportant. The best values for learning rate increased by a magnitude compared to F-ES over all oracles and higher temperatures seem overall beneficial. The best performing models achieved 0.520, 0.447 and 0.418

The highest performing hyperparameter sets are shown in Fig. 4.3.

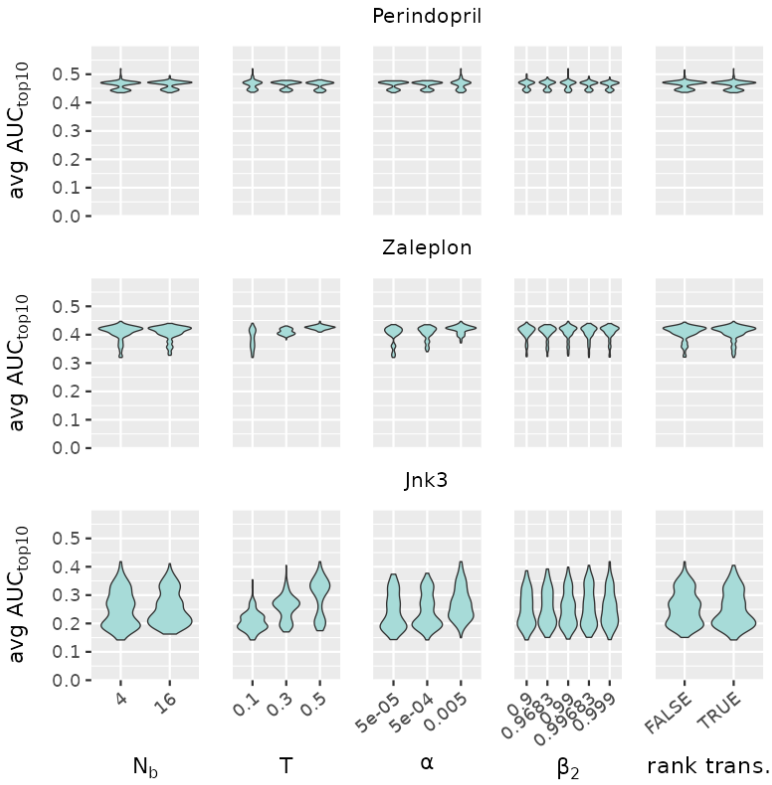

Figure 4.5: F-ES using Adam average performance over three runs for hyperparameters batch size N_b, sampling temperature T, learning rate α and decay rate of second moment for Adam β_2 on the scoring functions Perindopril, Zaleplon and JNK3.

Table 4.3: Best hyperparameters for F-ES with Adam

Setup	N_b	T	α	β_2	Rank trans.	AUC$_{top10}$
Perindopril	4	0.1	0.005	0.99	TRUE	0.520
Zaleplon	4	0.5	0.005	0.99	TRUE	0.447
Avg Per. Zal.	4	0.1	0.005	0.99	TRUE	
JNK3	4	0.5	0.005	0.999	FALSE	0.418

4.2.2.2 F-ES with Over-sampling

Over-sampling introduces the hyperparameter over-sampling count N_{os} and similarity penalty λ which were tested in a hyperparameter sweep together with batch size N_b and sampling temperature T. Similar to F-ES with Adam, population size and sampling noise were fixed to 30 and 0.01 respectively. Antithetic sampling was turned off and rank transformation were on. The results are summarized in Fig. 4.6.

Perindopril and Zaleplon had the highest performance with no over-sampling and got worse AUC_{top10} with higher over-sampling. Jnk3 got better with a larger over-sampling count and larger batch size. A second hyperparameter sweep was done on only over-sampling count and batch size on Jnk3 which showed larger batch size then 32 was not beneficial and larger over-sampling count was beneficial. These results are shown in Fig. 4.6.

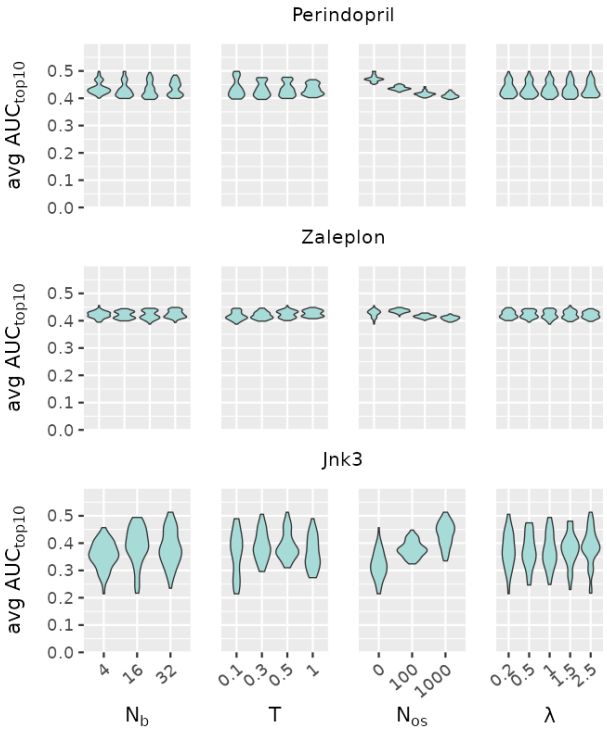

Figure 4.6: F-ES using over-sampling average performance over three runs for hyperparameters batch size N_b, sampling temperature T, over-sampling count N_{os} and similarity penalty λ on the scoring functions Perindopril, Zaleplon and JNK3.

Table 4.4: Best hyperparameters for F-ES with oversampling

Setup	N_b	T	N_{os}	λ	AUC_{top10}
Perindopril	4	0.1	0	2.5	0.498
Zaleplon	4	0.5	0	1.5	0.456
JNK3	32	0.5	1000	2.5	0.513

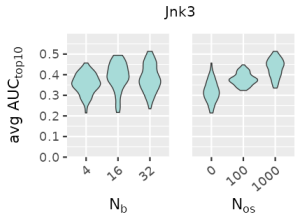

Figure 4.7: F-ES using over-sampling average performance over three runs for hyperparamters batch size N_b and over-sampling count N_{os} on the scoring functions Jnk3.

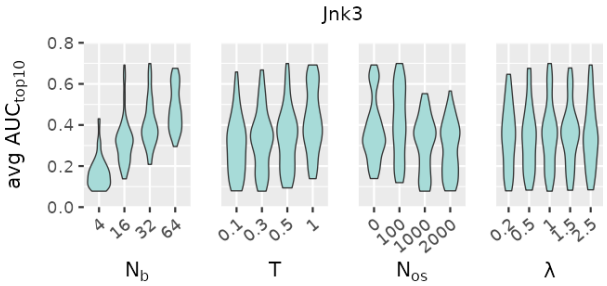

Figure 4.8: RL using over-sampling average performance over three runs for hyperparameters batch size N_b, sampling temperature T, over-sampling count N_{os} and similarity penalty λ on the scoring function JNK3.

4.2.2.3 F-ES with Adam and Over-sampling

F-ES with Adam and over-sampling was tested on a hyperparameter sweep with their respective unique hyperparameters, batch size N_b and temperature T. Other parameter were fixed to the best ones found for F-ES ($N_p = 30, \sigma_{\text{es}} = 001, \alpha = 0.005$, no antithetic sampling, with rank transformations) and β_1 was fixed to the standard value 0.9 [47].

Perindopril and Zaleplon were still most performant without over-sampling and got worse with a larger N_{os}. Jnk3 got better with larger N_{os} but not with larger batch size, unlike for F-ES with Adam. The best performing settings are shown in Tab. 4.5.

Table 4.5: Best hyperparameters for F-ES with Adam and oversampling

Setup	N_b	T	N_{os}	λ	β_2	$\text{AUC}_{\text{top10}}$
Perindopril	16	0.5	0	1.5	0.999	0.476
Zaleplon	64	0.5	100	1.5	0.9	0.445
JNK3	4	0.3	1000	1.0	0.9	0.544

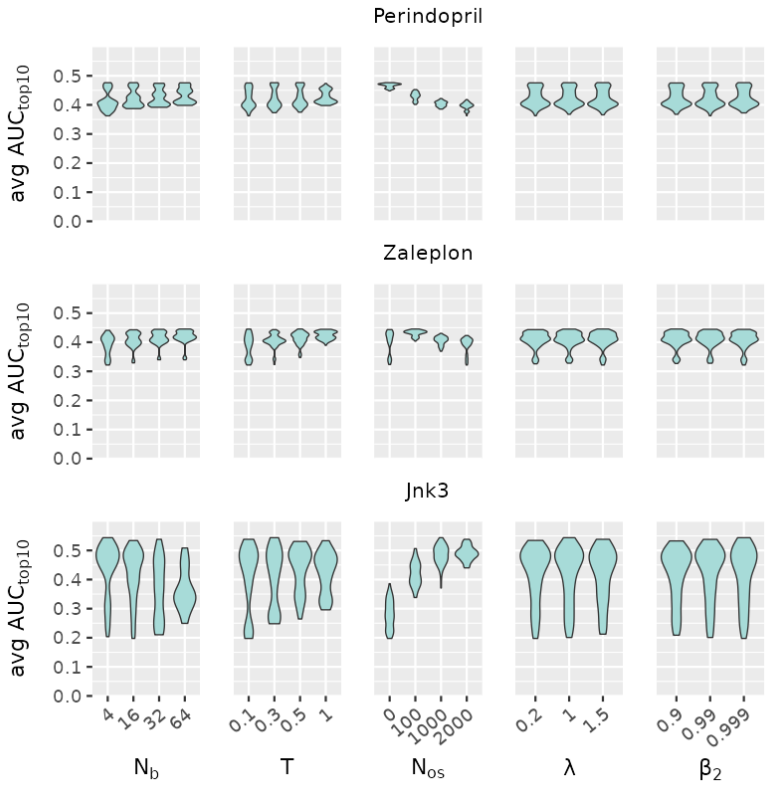

Figure 4.9: F-ES using Adam and over-sampling average performance over three runs for hyperparameters batch size N_b, sampling temperature T, over-sampling count N_{os} and similarity penalty λ and decay rate of second moment for Adam β_2 on the scoring functions Perindopril, Zaleplon and JNK3.

4.2.2.4 S-ES and D-ES

S-ES and D-ES was tested on JNK3 using the best hyperparameters for F-ES ($N_{\text{b}} = 16, T = 0.5, N_{\text{p}} = 30, \sigma_{\text{es}} = 0.001$, no antithetic sampling, using rank transformation) with varying learning rate α and learning rate for log standard deviation $\alpha_{\log\sigma}$. The avg AUC$_{\text{top10}}$ for the best hyperparmeter setups were 0.199 for S-ES and 0.366 for D-ES, which can be compared to 0.501 for F-ES and 0.510 for RL. Their poor performance meant no more testing was done on them.

4.3 Performance on the Benchmark Suite PMO

The PMO benchmark selected the hyperparameters set whose average performance was best on Zaleplon and Perindopril combined. To check if other oracles might be better we also tested the best hyperparmeters from Jnk3. Since Adam had a negativ effect on Jnk3 only the Perindopril and Zaleplon variant was tested for Adam. The reverse was true from oversampling motivating only testing the Jnk3 version.

The scoring function scaffold_hop crashed when running it and thus was excluded from all results.

Performance between the main metrics top10 (Fig. 4.10) and AUC_{top10} (Fig. 4.11) were similar. Near perfect top10 were achieved on DRD2, isomers_c9h10n2o2pf2cl, Mestranol_similarity and Albuterol_similarity. The most difficult scoring function was valsartan_smarts, on which all models struggled greatly. It computes the geometric mean of three physicochemical property terms, but sets the score to zero if a specific SMARTS substructure is not present.

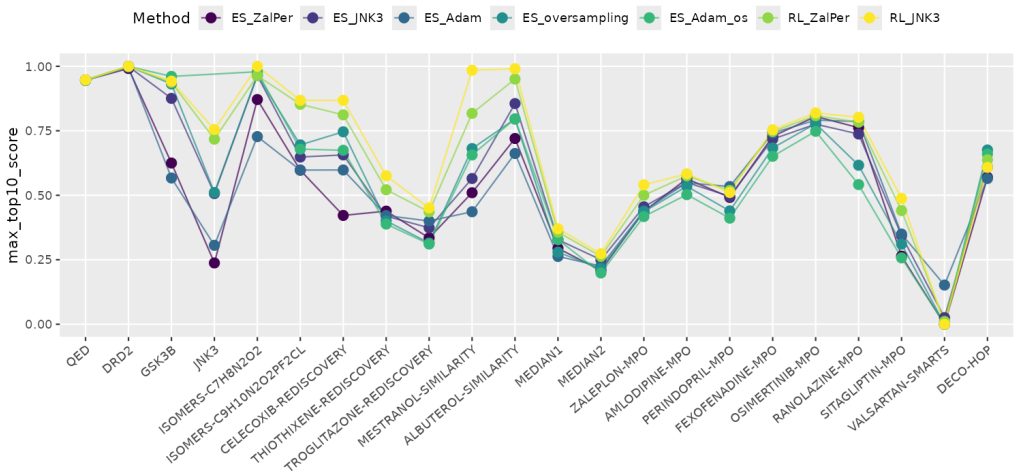

Figure 4.10: Max top10 on the PMO benchmark

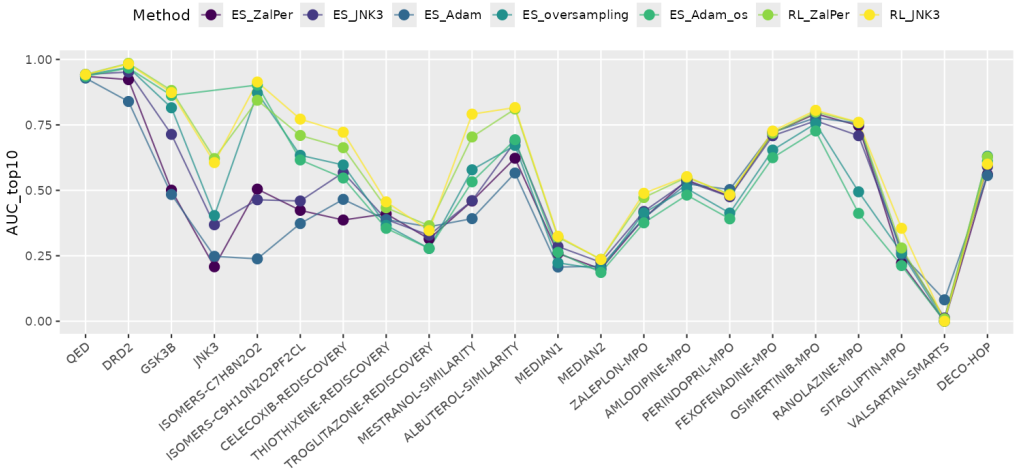

Figure 4.11: AUC_{top10} on the PMO benchmark

Over the complete benchmark both RL versions are the most strongly performant with average AUC_{top10} of 0.66 with hyperparameters from Zaleplon and Perindopril and 0.69 with hyperparameters from JNK3. ES with hyperparameters from Jnk3 often perform better then from Zaleplon and Perindopril, especially on GSK3β and Jnk3. Their respective average AUC_{top10} were 0.60 and 0.54 respectively.

The benefit for oversampling on ES varied a lot, with a positive effect on the non-similarity based scoring function GSK3β, isomers_c7h8n2o2 and isomers_c9h10n2o2pf2cl

where other ES versions struggle, and a negative effect on Perindopril, Fexofenadine and especially Randolazine which are similarity based scoring functions with some additional parts, and on which all other methods perform well. It had a average AUC_{top10} of 0.59.

ES with Adam had a weak performance except on valsartan_smarts were it outperforms all other models. Its average AUC_{top10} was 0.55. ES with Adam and oversampling performed similar but slightly worse then ES with only oversampling on all metrics.

The summarized results in more metrics are shown in Fig 4.12, with non-summarized in Appendix B. ES with oversampling stands out a lot with consistent high internal diversity, consistent low proportion of of samples belonging to the five most common scaffolds for both Murcko and topological scaffolds and somewhat high and consistent number of unique scaffolds, and highly correlated proportion of unique scaffolds, for both Murcko and topological scaffolds. ES with oversampling has consistent high diversity on all metrics. It does also have a consistently have a high mean QED with a mean of 0.67. For comparison, the mean QED sampled from 10 000 SMILES from the prior was 0.53 at $T = 0.5$, 0.45 at $T = 0.3$ and 0.37 at $T = 0.38$.

ES without Adam or oversampling outperforms RL on internal diversity, and under performs on number of unique scaffolds and proportion unique scaffolds on both Murcko and topological scaffolds. ES with hyperparameters from Jnk3 perform better then RL on proportion taken up by five most common Murcko scaffolds but not topological scaffolds.

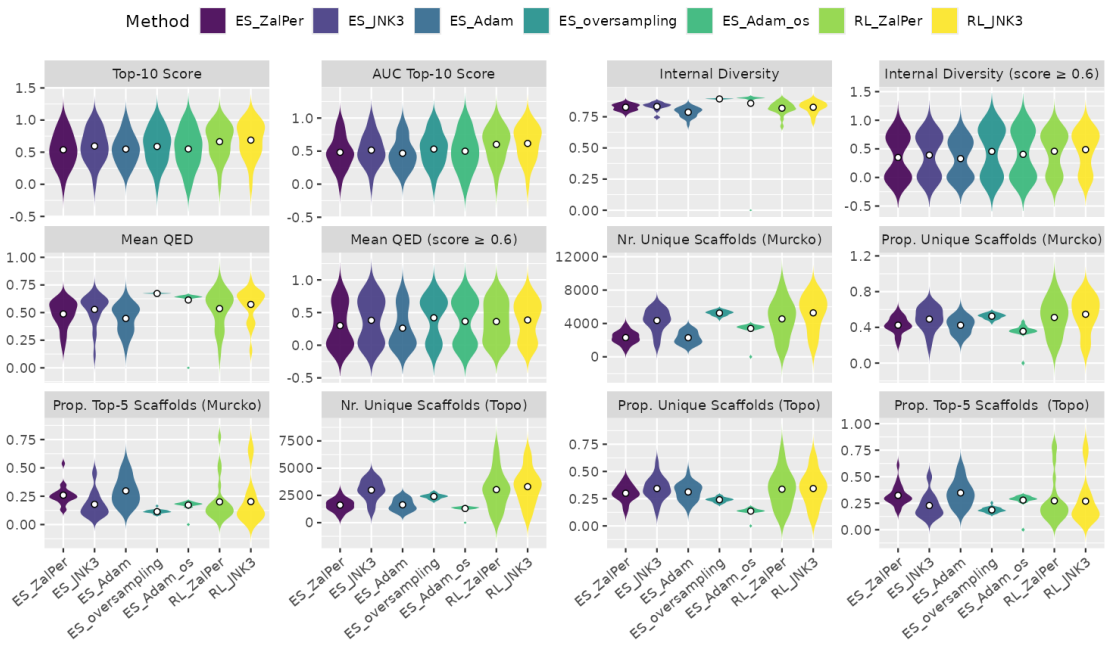

Figure 4.12: Distribution of performance on all scoring functions on PMO benchmark for varying metrics on tested methods. White dots show the mean.

The number of unique high scores count the number of unique SMILES which score

$\geq$ 0.6. RL strongly outperform ES on this metric with ES with hyperparameters from JNK3 being the least weak ES variant. The full results on this is shown in Fig. 4.13. On five scoring functions (median1, median2, perindopril and sitagliptin_mpo) no method generated any SMILES which scored 0.6 or higher. The high threshold of high scoring SMILES mean the internal diversity and mean QED metrics on the high scoring subsets are hard to analyze.

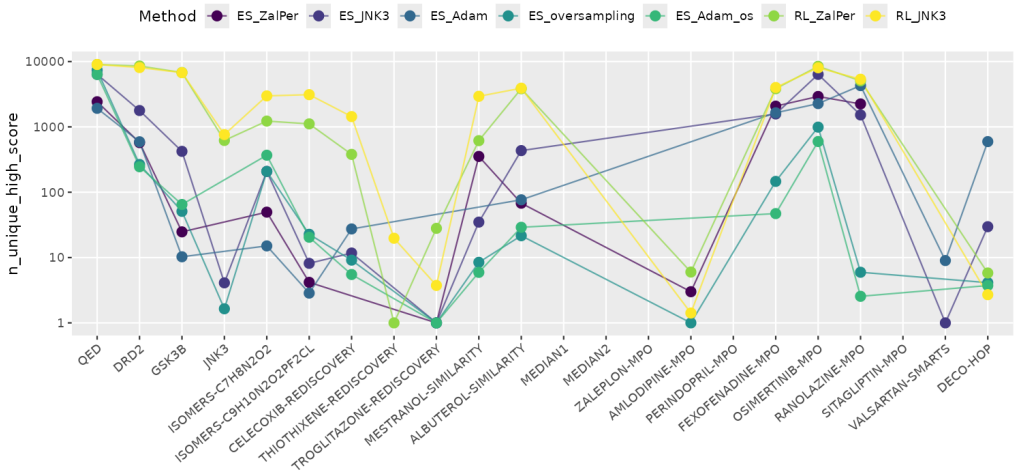

Figure 4.13: Number of unique SMILES scoring $\geq$ 0.6 on the PMO benchmark on a log scale.

The number of unique calls to the scoring function (Fig. 4.14) varied between methods. The experiment was stopped if: the top10 was not increased in a span of 200 epochs, where a epoch contain one parameter update, the allocated 30 hours ran out or the complete budget of 10 000 calls were used. The allocated time was sufficient for most scoring functions and methods, except Deco_hop and ES with Adam and oversampling. Deco_hop stopped 5 times for ES_ZalPer, 4 times for ES_Jnk3, 3 for ES_Adam, 0 for ES_oversampling, 0 for ES_Adam_os, 2 for RL_ZalPer and 1 for RL_Jnk3. ES_Adam_os ran out of times on 43 of the total 110 runs. RL_Jnk3 crashed once on DRD2 during a run. Not using the full number of calls to the scoring function can thus be attributed to methods plateauing.

RL generally ran for longer then ES versions except against ES with over-sampling which always used all calls.

The allocated time was sufficient for most scoring functions and methods, except Deco_hop and ES with Adam and oversampling. Deco_hop evaluates the presence of three SMARTS patterns and awards one third of the score for each pattern detected. Under this scoring function, early stopping occurred 5 times for ES_ZalPer, 4 times for ES_Jnk3, 3 for ES_Adam, 0 for ES_oversampling, 0 for ES_Adam_os, 2 for RL_ZalPer and 1 for RL_Jnk3. ES_Adam_os exceeded the time limit in 43 of the 110 total runs. Additionally, RL_Jnk3 crashed once on DRD2 during a run.

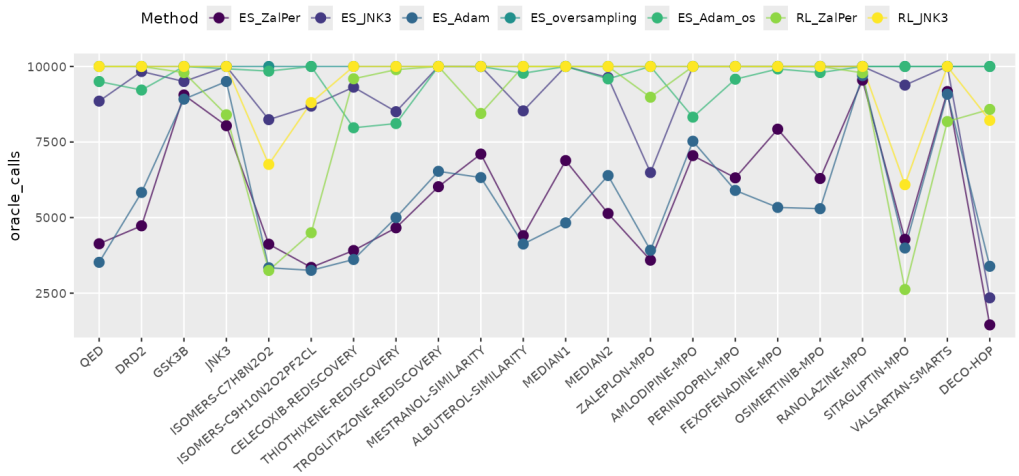

Figure 4.14: Number of unique calls to the oracle function used on the PMO benchmark

5

Discussion

RL performed on average better in both the top-10 score and AUC_{top10} metrics and yeilded on average more high scoring (score ≥ 0.6) molecules than tested F-ES variants. F-ES with the novel over-sampling procedure was the most diverse, and stably so, in all metrics. It achieved the highest average AUC_{top10} and second highest average top-10 score of all F-ES variants but produced the lowest number of high-scoring molecules. Excluding oversampling, diversity was similar for all other methods, with ES outperforming on internal diversity and RL outperforming on the number of unique scaffolds. It should be noted that REINVENT offers several mechanisms to adjust and improve performance, most notably diversity filters and replay memory that stores high-performing SMILES for off-policy use, which often enhance both optimization and diversity; these were not enabled in our tests.

Not all runs used the full budget of 10 000 unique calls to the scoring function. This was primarily due to early stopping, triggered when the top-10 score failed to improve for 200 epochs. In practice, early stopping indicates the method has reached a local maximum it cannot escape. ES typically consumes fewer scoring calls and achieves lower final scores, which suggests it is less effective at escaping local maxima than RL in this context.

To improve escapes from local maxima, two complementary strategies can be considered. First, introduce more sampling noise to increase the probability of stochastic jumps out of local maxima. Second, broaden SMILES sampling to cover a wider region of chemical space, increasing the likelihood of discovering higher scoring points outside the current basin. For this second strategy to be effective, the algorithm must leverage off-basin discoveries in its update step (i.e., ensure those points significantly influence parameter updates). If such points are not efficiently incorporated, broader sampling may merely prolong the run without enabling genuine escape from the local maximum.

A possible approach for the first strategy is to adaptively increase how noisy gradient estimates are when the top-10 score has not improved for an extended period using for example a smaller population size.

Over-sampling consistently uses the full scoring budget and explicitly encourages more diverse sampling, aligning with the second strategy above. Its slightly better performance than other ES variants suggests it is might be improving exploration and the chance of locating superior optima.

5.1 The Role of Noise

Noise in gradient estimates can be both positive and negative. Noisy gradients mean that traversal in the true gradients direction is slower as gradient estimates point less in the true gradients direction. They can also aid in stochastically escaping local maxima or travel over valleys.

Most parameters for ES have a clear effect on how noisy gradient estimates are. Larger population and batch size increase the accuracy of their respective Monte Carlo estimations, defined by Eq. (2.40) and Eq. (2.43), at the cost of more calls to the scoring function. Reducing the step size similarly mitigates the impact of noisy estimates by taking more incremental updates, again at the cost of additional scoring calls. Antithetic sampling aims to reduce the variance of the outer Monte Carlo estimation, defined by Eq. (2.40), at the cost of less parameter space explored. Theoretically determining these trade-offs is difficult and has thus, in this thesis, been done with extensive hyperparameter tuning.

Adam is designed to reduce the impact of noisy gradient estimates by using momentum and adaptive learning rates, effectively smoothing updates based on information from previous steps. In this study, when applied to F-ES, Adam negatively affected performance. A possible hypothesis is that Adams smoothing may dampen the stochastic perturbations that help F-ES escape local maxima, particularly when noise levels are moderate.

It would be informative to evaluate F-ES with Adam under deliberately noisier settings (smaller population sizes, smaller batch sizes, and smaller step sizes) where Adams variance reduction might be more beneficial. In the hyperparameter tuning conducted here, the population size for Adam was fixed, and batch size was evaluated at only two values which yielded similar performance, limiting our ability to draw conclusions about Adams behavior in noisy environments.

5.2 Variance Estimating ES

Prior work on distribution based ES for large neural networks has predominantly employed fixed variance and isotropic search distributions [15], [16]. This thesis evaluates that assumption by testing a diagonal variance ES (D-ES) and a single common variance ES (S-ES) on Jnk3. Both variants underperformed relative to all other methods. A possible factor is parameter burden: D-ES requires estimating a separate variance per dimension, effectively doubling the number of parameters compared with the fixed variance ES (F-ES), which can be difficult to estimate reliably and may explain its weaker results. Even more notably, S-ES, which introduces only a single additional variance parameter, performed worst overall. Because S-ES was initialized with the best performing σ_{es} from F-ES, its subsequent adaptation appears to have pushed σ_{es} in a counterproductive direction.

The implemented D-ES is closely related to the separable Natural Evolution Strategy (SNES), differing only in how it transforms scores before computing updates [33]. In

SNES, scores are mapped to rank based utilities that make the worst performing runs equally bad, down-weigh the influence of very good scores, and enforce a mean-zero normalization. Wierstra argues that the choice is not important as long as it is monotonic and based on ranks[33]. Nevertheless, evaluating the impact of this score-shaping choice across all ES variants in our setting would be informative.

5.3 Over-sampling

One of the more surprising results concerns the novel over-sampling procedure. It originated as a pragmatic alternative to beam search, a deterministic sampling method that expands the k most probable tokens at each step and returns the most likely completed sequence. Beam search would eliminate sampling stochasticity (each model parameter set allways produce the same SMILES) and effectively set the batch size to one, reducing oracle calls to implement, but has a high memory requirement. Instead of implementing beam search, a precursor of over-sampling was built where a larger number of SMILES than needed where sampled and those with highest likelihood was selected. The initial prototype produced highly similar SMILES and performed poorly. Batch size one was not tested. The low diversity of this version motivated the current implementation.

F-ES with oversampling has high ECF4 internal diversity, which is expected since that is exactly what it encourages in the sampling. It performs well in all other tested diversity metrics, but more interestingly, it has extremely low variance. No good explanation has been found for this. F-ES with oversampling also had a very high and very stable QED, for which no explanation has been found.

The current implementation encourages diversity using the maximum ECF4 similarity against all already selected SMILES, for each candidate. Other diversity metrics should be tested here to see which encourage exploration and diversity the best.

Over-sampling, as implemented in this thesis, uses the likelihood of generated SMILES on the current model, which ES otherwise does not need and is one of its benefits. This reliance is not intrinsic: if the initial sampling is performed at the desired temperature, all sampled SMILES can be treated as equally likely during the selection step. By doing initial sampling at a high temperature, generating the same SMILES is less likely. This reduces the amount of initial sampling needed to achieve the same number of unique SMILES, making the algorithm faster, given likelihoods are available. So, over-sampling can be applied to generators that do not provide likelihoods, particularly when sample generation is relatively inexpensive.

5.4 Future Work

This project has multiple directions in which to continue the work, some of which are presented below.

- Evaluate alternative rank based score transformations for variance estimating ES (e.g., NES utilities) to assess whether score shaping improves stability and

performance.

- Introduce an adaptive sampling noise schedule in F-ES that increases how noisy gradient estimates are only when the top-10 score stagnates, to aid escapes from local maxima without degrading normal progress.
- Assess beam search or other deterministic sampling for F-ES to remove one Monte Carlo component and potentially reduce gradient variance, comparing compute/memory trade-offs.
- Extend over-sampling by testing additional diversity metrics beyond ECF4 to identify which best promote exploration and robust performance.

5.5 Conclusions

The evolutionary strategies tested in this thesis could not achieve the same performance as the baseline from REINVENT4. These results are consistent with the limited use of ES for fine-tuning large language models in the broader literature and suggest that, under the tested settings, RL remains the stronger choice for direct performance optimization.

At the same time, the study surfaced areas where ES may offer distinctive value. First, ES methods that do not rely on model likelihoods can be advantageous for generators where likelihoods are unavailable or impractical to computes. Second, the novel over-sampling procedure, conceptually aligned with widening exploration in chemical space, showed encouraging signs. It reliably used the evaluation budget, improved exploration, and delivered slightly better outcomes than other ES variants with notably high and stable diversity on tested metrics as well as high and stable QED.

Bibliography

[1] M. Schlander, K. Hernandez-Villafuerte, C.-Y. Cheng, J. Mestre-Ferrandiz, and M. Baumann, "How Much Does It Cost to Research and Develop a New Drug? A Systematic Review and Assessment," en, *PharmacoEconomics*, vol. 39, no. 11, pp. 1243–1269, Nov. 2021, ISSN: 1179-2027. DOI: 10.1007/s40273-021-01065-y. Accessed: May 26, 2026. [Online]. Available: https://doi.org/10.1007/s40273-021-01065-y.

[2] G. A. Van Norman, "Drugs, Devices, and the FDA: Part 1: An Overview of Approval Processes for Drugs," *JACC: Basic to Translational Science*, vol. 1, no. 3, pp. 170–179, Apr. 2016, ISSN: 2452-302X. DOI: 10.1016/j.jacbts.2016.03.002. Accessed: Jun. 22, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2452302X1600036X.

[3] *Virtual Chemical Libraries | Journal of Medicinal Chemistry*. Accessed: May 26, 2026. [Online]. Available: https://pubs.acs.org/doi/10.1021/acs.jmedchem.8b01048.

[4] L. Ruddigkeit, R. Van Deursen, L. C. Blum, and J.-L. Reymond, "Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17," en, *Journal of Chemical Information and Modeling*, vol. 52, no. 11, pp. 2864–2875, Nov. 2012, ISSN: 1549-9596, 1549-960X. DOI: 10.1021/ci300415d. Accessed: May 26, 2026. [Online]. Available: https://pubs.acs.org/doi/10.1021/ci300415d.

[5] R. L. van den Broek, S. Patel, G. J. P. van Westen, W. Jespers, and W. Sherman, "In Search of Beautiful Molecules: A Perspective on Generative Modeling for Drug Design," *Journal of Chemical Information and Modeling*, vol. 65, no. 18, pp. 9383–9397, Sep. 2025, Publisher: American Chemical Society, ISSN: 1549-9596. DOI: 10.1021/acs.jcim.5c01203. Accessed: May 27, 2026. [Online]. Available: https://doi.org/10.1021/acs.jcim.5c01203.

[6] J. Hughes, S. Rees, S. Kalindjian, and K. Philpott, "Principles of early drug discovery," en,

[7] N. A. Tamimi and P. Ellis, "Drug Development: From Concept to Marketing!" *Nephron Clinical Practice*, vol. 113, no. 3, pp. c125–c131, Aug. 2009, ISSN: 1660-2110. DOI: 10.1159/000232592. Accessed: May 26, 2026. [Online]. Available: https://doi.org/10.1159/000232592.

[8] N. Brown, M. Fiscato, M. H. Segler, and A. C. Vaucher, "GuacaMol: Benchmarking Models for de Novo Molecular Design," en, *Journal of Chemical Information and Modeling*, vol. 59, no. 3, pp. 1096–1108, Mar. 2019, ISSN: 1549-9596,

1549-960X. DOI: 10.1021/acs.jcim.8b00839. Accessed: Mar. 26, 2026. [Online]. Available: https://pubs.acs.org/doi/10.1021/acs.jcim.8b00839.

[9] G. Lamanna et al., "GENERA: A Combined Genetic/Deep-Learning Algorithm for Multiobjective Target-Oriented De Novo Design," *Journal of Chemical Information and Modeling*, vol. 63, no. 16, pp. 5107–5119, Aug. 2023, Publisher: American Chemical Society, ISSN: 1549-9596. DOI: 10.1021/acs.jcim.3c00963. Accessed: May 27, 2026. [Online]. Available: https://doi.org/10.1021/acs.jcim.3c00963.

[10] Y. Matsukiyo, C. Yamanaka, and Y. Yamanishi, "De Novo Generation of Chemical Structures of Inhibitor and Activator Candidates for Therapeutic Target Proteins by a Transformer-Based Variational Autoencoder and Bayesian Optimization," *Journal of Chemical Information and Modeling*, vol. 64, no. 7, pp. 2345–2355, Apr. 2024, Publisher: American Chemical Society, ISSN: 1549-9596. DOI: 10.1021/acs.jcim.3c00824. Accessed: May 27, 2026. [Online]. Available: https://doi.org/10.1021/acs.jcim.3c00824.

[11] E. Lin, C.-H. Lin, and H.-Y. Lane, "De Novo Peptide and Protein Design Using Generative Adversarial Networks: An Update," *Journal of Chemical Information and Modeling*, vol. 62, no. 4, pp. 761–774, Feb. 2022, Publisher: American Chemical Society, ISSN: 1549-9596. DOI: 10.1021/acs.jcim.1c01361. Accessed: May 27, 2026. [Online]. Available: https://doi.org/10.1021/acs.jcim.1c01361.

[12] S. Zhai et al., *Normalizing Flows are Capable Generative Models*, arXiv:2412.06329 [cs.CV], Jun. 2025. DOI: 10.48550/arXiv.2412.06329. Accessed: May 27, 2026. [Online]. Available: http://arxiv.org/abs/2412.06329.

[13] A. Alakhdar, B. Poczos, and N. Washburn, "Diffusion Models in De Novo Drug Design," *Journal of Chemical Information and Modeling*, vol. 64, no. 19, pp. 7238–7256, Oct. 2024, Publisher: American Chemical Society, ISSN: 1549-9596. DOI: 10.1021/acs.jcim.4c01107. Accessed: May 27, 2026. [Online]. Available: https://doi.org/10.1021/acs.jcim.4c01107.

[14] H. H. Loeffler et al., "Reinvent 4: Modern AIdriven generative molecule design," en, *Journal of Cheminformatics*, vol. 16, no. 1, p. 20, Feb. 2024, ISSN: 1758-2946. DOI: 10.1186/s13321-024-00812-5. Accessed: Feb. 11, 2026. [Online]. Available: https://doi.org/10.1186/s13321-024-00812-5.

[15] X. Qiu et al., *Evolution Strategies at Scale: LLM Fine-Tuning Beyond Reinforcement Learning*, Feb. 2026. DOI: 10.48550/arXiv.2509.24372. [Online]. Available: https://doi.org/10.48550/arXiv.2509.24372.

[16] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, *Evolution Strategies as a Scalable Alternative to Reinforcement Learning*, arXiv:1703.03864 [stat], Sep. 2017. DOI: 10.48550/arXiv.1703.03864. Accessed: Feb. 9, 2026. [Online]. Available: http://arxiv.org/abs/1703.03864.

[17] A. Y. Majid, S. Saaybi, V. Francois-Lavet, R. V. Prasad, and C. Verhoeven, "Deep Reinforcement Learning Versus Evolution Strategies: A Comparative Survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 9, pp. 11 939–11 957, Sep. 2024, ISSN: 2162-2388. DOI: 10.1109/TNNLS.2023.3264540. Accessed: Feb. 9, 2026. [Online]. Available: https://ieeexplore.ieee.org/document/10114063/.

[18] B. Sarkar et al., *Evolution Strategies at the Hyperscale*, arXiv:2511.16652 [cs], Nov. 2025. DOI: 10.48550/arXiv.2511.16652. Accessed: Feb. 9, 2026. [Online]. Available: http://arxiv.org/abs/2511.16652.

[19] W. Gao, T. Fu, J. Sun, and C. W. Coley, *Sample Efficiency Matters: A Benchmark for Practical Molecular Optimization*, arXiv:2206.12411 [cs], Oct. 2022. DOI: 10.48550/arXiv.2206.12411. Accessed: Feb. 9, 2026. [Online]. Available: http://arxiv.org/abs/2206.12411.

[20] D. Weininger, "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules," *Journal of Chemical Information and Computer Sciences*, vol. 28, no. 1, pp. 31–36, Feb. 1988, Publisher: American Chemical Society, ISSN: 0095-2338. DOI: 10.1021/ci00057a005. Accessed: Mar. 26, 2026. [Online]. Available: https://doi.org/10.1021/ci00057a005.

[21] D. Weininger, A. Weininger, and J. L. Weininger, "SMILES. 2. Algorithm for generation of unique SMILES notation," en, *Journal of Chemical Information and Computer Sciences*, vol. 29, no. 2, pp. 97–101, May 1989, ISSN: 0095-2338, 1520-5142. DOI: 10.1021/ci00062a008. Accessed: Apr. 8, 2026. [Online]. Available: https://pubs.acs.org/doi/abs/10.1021/ci00062a008.

[22] G. Landrum, "RDKit Book," RDKit Project, Tech. Rep., 2025. [Online]. Available: https://www.rdkit.org/docs/RDKit_Book.html.

[23] *Daylight Theory: SMARTS - A Language for Describing Molecular Patterns*. Accessed: Apr. 8, 2026. [Online]. Available: https://www.daylight.com/dayhtml/doc/theory/theory.smarts.html.

[24] I. Sutskever, J. Martens, and G. Hinton, "Generating Text with Recurrent Neural Networks," en,

[25] T. Kudo and J. Richardson, *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*, arXiv:1808.06226 [cs], Aug. 2018. DOI: 10.48550/arXiv.1808.06226. Accessed: May 12, 2026. [Online]. Available: http://arxiv.org/abs/1808.06226.

[26] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A Neural Probabilistic Language Model," en,

[27] B. Mehlig, *Machine learning with neural networks*, arXiv:1901.05639 [cs.LG] version: 3, Feb. 2021. DOI: 10.48550/arXiv.1901.05639. Accessed: May 26, 2026. [Online]. Available: http://arxiv.org/abs/1901.05639.

[28] H. Sak, A. Senior, and F. Beaufays, *Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition*, arXiv:1402.1128 [cs], Feb. 2014. DOI: 10.48550/arXiv.1402.1128. Accessed: Apr. 15, 2026. [Online]. Available: http://arxiv.org/abs/1402.1128.

[29] *LSTM PyTorch 2.11 documentation*, en. Accessed: Apr. 15, 2026. [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.LSTM.html.

[30] N. Andreasson, A. Evgrafov, and M. Patriksson, "An Introduction to Continuous Optimization: Foundations and Fundamental Algorithms," en,

[31] H. Robbins and S. Monro, "A Stochastic Approximation Method," en, *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, Sep. 1951, Publisher: Institute of Mathematical Statistics, ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177729586. Accessed: May 6, 2026. [Online]. Avail-

able: https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-22/issue-3/A-Stochastic-Approximation-Method/10.1214/aoms/1177729586.full.

[32] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, arXiv:1412.6980 [cs], Jan. 2017. DOI: 10.48550/arXiv.1412.6980. Accessed: Feb. 11, 2026. [Online]. Available: http://arxiv.org/abs/1412.6980.

[33] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, *Natural Evolution Strategies*. DOI: 10.48550/arXiv.1106.4487. [Online]. Available: https://doi.org/10.48550/arXiv.1106.4487.

[34] S. Kullback and R. A. Leibler, "On Information and Sufficiency," en, *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, Mar. 1951, Publisher: Institute of Mathematical Statistics, ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177729694. Accessed: May 6, 2026. [Online]. Available: https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-22/issue-1/On-Information-and-Sufficiency/10.1214/aoms/1177729694.full.

[35] J. R. Peters, "Machine Learning of Motor Skills for Robotics," en,

[36] B. Mehlig, "Markov Decision Processes," in *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*, Cambridge: Cambridge University Press, 2021.

[37] B. Mehlig, "Value Functions," in *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*, Cambridge: Cambridge University Press, 2021.

[38] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in *Advances in Neural Information Processing Systems*, vol. 12, MIT Press, 1999. Accessed: May 26, 2026. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1999/hash/464d828b85b0bed98e80ade0a5c43b0f-Abstract.html.

[39] P. W. Glynn and M. Olvera-Cravioto, "Likelihood Ratio Gradient Estimation for Steady-State Parameters," en, *Stochastic Systems*, vol. 9, no. 2, pp. 83–100, Jun. 2019, ISSN: 1946-5238, 1946-5238. DOI: 10.1287/stsy.2018.0023. Accessed: May 7, 2026. [Online]. Available: https://pubsonline.informs.org/doi/10.1287/stsy.2018.0023.

[40] M. Thomas, A. Bou, J. C. Gómez-Tamayo, G. Tresadern, M. Ahmad, and G. De Fabritiis, "REINFORCE-ING Chemical Language Models for Drug Discovery," *Journal of Chemical Information and Modeling*, vol. 65, no. 23, pp. 12 752–12 763, Dec. 2025, Publisher: American Chemical Society, ISSN: 1549-9596. DOI: 10.1021/acs.jcim.5c02053. Accessed: Apr. 24, 2026. [Online]. Available: https://doi.org/10.1021/acs.jcim.5c02053.

[41] H.-G. Beyer and H.-P. Schwefel, "Evolution Strategies - A comprehensive introduction," en,

[42] Sentewolf, *Concept of directional optimization in CMA-ES algorithm*, Oct. 2008. Accessed: May 8, 2026. [Online]. Available: https://commons.wikimedia.org/wiki/File:Concept_of_directional_optimization_in_CMA-ES_algorithm.png.

[43] H. Safizadeh et al., "Improving Measures of Chemical Structural Similarity Using Machine Learning on ChemicalGenetic Interactions," *Journal of Chemical Information and Modeling*, vol. 61, no. 9, pp. 4156–4172, Sep. 2021, ISSN: 1549-9596. DOI: 10.1021/acs.jcim.0c00993. Accessed: May 8, 2026. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC8479812/.

[44] D. Rogers and M. Hahn, "Extended-Connectivity Fingerprints," *Journal of Chemical Information and Modeling*, vol. 50, no. 5, pp. 742–754, May 2010, Publisher: American Chemical Society, ISSN: 1549-9596. DOI: 10.1021/ci100050t. Accessed: Apr. 28, 2026. [Online]. Available: https://doi.org/10.1021/ci100050t.

[45] G. W. Bemis and M. A. Murcko, "The Properties of Known Drugs. 1. Molecular Frameworks," *Journal of Medicinal Chemistry*, vol. 39, no. 15, pp. 2887–2893, Jan. 1996, Publisher: American Chemical Society, ISSN: 0022-2623. DOI: 10.1021/jm9602928. Accessed: May 8, 2026. [Online]. Available: https://doi.org/10.1021/jm9602928.

[46] T. Blaschke et al., "REINVENT 2.0: An AI Tool for De Novo Drug Design," *Journal of Chemical Information and Modeling*, vol. 60, no. 12, pp. 5918–5922, Dec. 2020, Publisher: American Chemical Society, ISSN: 1549-9596. DOI: 10.1021/acs.jcim.0c00915. Accessed: May 29, 2026. [Online]. Available: https://doi.org/10.1021/acs.jcim.0c00915.

[47] *Adam PyTorch main documentation*. Accessed: May 22, 2026. [Online]. Available: https://docs.pytorch.org/docs/main/generated/torch.optim.adam.Adam_class.html.

[48] M. Olivecrona, T. Blaschke, O. Engkvist, and H. Chen, "Molecular de-novo design through deep reinforcement learning," en, *Journal of Cheminformatics*, vol. 9, no. 1, p. 48, Sep. 2017, ISSN: 1758-2946. DOI: 10.1186/s13321-017-0235-x. Accessed: Feb. 9, 2026. [Online]. Available: https://doi.org/10.1186/s13321-017-0235-x.

[49] Y. Li, L. Zhang, and Z. Liu, "Multi-objective de novo drug design with conditional graph generative model," en, *Journal of Cheminformatics*, vol. 10, no. 1, p. 33, Jul. 2018, ISSN: 1758-2946. DOI: 10.1186/s13321-018-0287-6. Accessed: Mar. 30, 2026. [Online]. Available: https://doi.org/10.1186/s13321-018-0287-6.

[50] S. H. Bertz, "The first general index of molecular complexity," *Journal of the American Chemical Society*, vol. 103, no. 12, pp. 3599–3601, Jun. 1981, Publisher: American Chemical Society, ISSN: 0002-7863. DOI: 10.1021/ja00402a071. Accessed: Jun. 22, 2026. [Online]. Available: https://doi.org/10.1021/ja00402a071.

[51] G. R. Bickerton, G. V. Paolini, J. Besnard, S. Muresan, and A. L. Hopkins, "Quantifying the chemical beauty of drugs," en, *Nature Chemistry*, vol. 4, no. 2, pp. 90–98, Feb. 2012, Publisher: Nature Publishing Group, ISSN: 1755-4349. DOI: 10.1038/nchem.1243. Accessed: Mar. 30, 2026. [Online]. Available: https://www.nature.com/articles/nchem.1243.

A

Oracle functions in the Practical Molecular Benchmark

The oracle functions from the Practical molecular benchmark are found in Tab. A.1. The table use $\mathrm{sim}_{\mathrm{x}}(m_1, m_2)$ to notate the Tanimoto using the fingerprint x. The function bertz is a measure of molecular complexity [50].

Transformations in the table are defined by:

$$\mathrm{G}_{\sigma,\mu}(x) = \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{A.1}$$

$$\mathrm{MaxG}_{\sigma,\mu}(x) = \begin{cases} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), & \text{if } x > \mu \\ 1, & \text{if } x \leq \mu \end{cases} \tag{A.2}$$

$$\mathrm{MinG}_{\sigma,\mu}(x) = \begin{cases} 1, & \text{if } x \geq \mu \\ \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), & \text{if } x < \mu \end{cases} \tag{A.3}$$

$$T_\tau(x) = \begin{cases} x & \text{if } x \geq \tau \\ 0 & \text{if } x < \tau \end{cases} \tag{A.4}$$

SMILES and SMARTS used in the table are defined by:

$\mathrm{m}_1 =$ `CCCOc1cc2ncnc(Nc3ccc4ncsc4c3)c2cc1S(=O)(=O)C(C)(C)C`,
$s_1 =$ `[#7]-c1n[c;h1]nc2[c;h1]c(-[#8])[c;h0][c;h1]-c12`,
$s_2 =$ `[#7]-c1ccc2ncsc2c1`,
$s_3 =$ `CS([#6])(=O)=O`,
$s_4 =$ `[#6]-[#6]-[#6]-[#8]-[#6]~[#6]~[#6]~[#6]~[#6]-[#7]-c1ccc2ncsc2c1`.

Table A.1: Oracle functions found in the Practical molecular benchmark [19]

Benchmark	**Scoring Function**
QED	Quantitative estimate of drug-likeness [51]
DRD2	Predicted activity against Dopamine D2 receptor [48]
GSK3β	Predicted activity against GSK3β kinase [49]
JNK3	Predicted activity against JNK3 kinase [49]
isomers_c7h8n2o2	$\mathbb{1}[m$ is isomer to $\mathrm{C_7H_8N_2O_2}]$
isomers_c9h10n2o2pf2cl	$\mathbb{1}[m$ is isomer to $\mathrm{C_9H_{10}N_2O_2PF_2Cl}]$
celecoxib_rediscovery	$\mathrm{sim}_{\mathrm{ECFC4}}(m, \mathrm{celecoxib})$
thiothixene_rediscovery	$\mathrm{sim}_{\mathrm{ECFC4}}(m, \mathrm{thiothixene})$
troglitazone_rediscovery	$\mathrm{sim}_{\mathrm{ECFC4}}(m, \mathrm{troglitazone})$
mestranol_similarity	$\mathrm{T}_{0.75}(\mathrm{sim}_{\mathrm{AP}}(m, \mathrm{mestranol}))$
albuterol_similarity	$\mathrm{T}_{0.75}(\mathrm{sim}_{\mathrm{FCFC4}}(m, \mathrm{albuterol}))$
median1	$(\mathrm{sim}_{\mathrm{ECFC4}}(m, \mathrm{camphor}) \times \mathrm{sim}_{\mathrm{ECFC4}}(m, \mathrm{menthol}))^{1/2}$
median2	$(\mathrm{sim}_{\mathrm{ECFC6}}(m, \mathrm{tadalafil}) \times \mathrm{sim}_{\mathrm{ECFC6}}(m, \mathrm{sildenafil}))^{1/2}$
zaleplon_mpo	$(\mathrm{sim}_{\mathrm{ECFC4}}(m, \mathrm{zaleplon}) \times \mathbb{1}[m$ is isomer to $\mathrm{C_{19}H_{17}N_3O_2}])^{1/2}$
amlodipine_mpo	$(\mathrm{sim}_{\mathrm{ECFC4}}(m, \mathrm{amlodipine}) \times G_{3,0.5}(\text{number of rings}(m)))^{1/2}$
perindopril_mpo	$(\mathrm{sim}_{\mathrm{ECFC4}}(m, \mathrm{perindopril})$ $\times G_{3,0.5}(\text{number of aromatic rings}(m)))^{1/2}$
fexofenadine_mpo	$(\mathrm{T}_{0.8}(\mathrm{sim}_{\mathrm{AP}}(m, \mathrm{fexofenadine}))$ $\times \mathrm{MaxG}_{90,2}(\mathrm{TPSA}(m))$ $\times \mathrm{MinG}_{4,2}(\log P(m)))^{1/3}$
osimertinib_mpo	$(\mathrm{T}_{0.8}(\mathrm{sim}_{\mathrm{FCFC4}}(m, \mathrm{osimertinib}))$ $\times \mathrm{MinG}_{0.85,2}(\mathrm{sim}_{\mathrm{ECFC6}}(m, \mathrm{osimertinib}))$ $\times \mathrm{G}_{100,2}(\mathrm{TPSA}(m))$ $\times \mathrm{MinG}_{1,2}(\log P(m)))^{1/4}$
ranolazine_mpo	$(\mathrm{T}_{0.7}(\mathrm{sim}_{\mathrm{AP}}(m, \mathrm{ranolazine}))$ $\times \mathrm{MaxG}_{7,1}(\log P(m))$ $\times \mathrm{MaxG}_{95,20}(\mathrm{TPSA}(m))$ $\times \mathrm{G}_{1,1}(\text{number of fluorine atoms}(m)))^{1/4}$
sitagliptin_mpo	$(G_{0,0.1}(\mathrm{sim}_{\mathrm{ECFC4}}(m, \mathrm{sitagliptin}))$ $\times G_{2.0165,0.2}(\log P(m))$ $\times G_{77.04,5}(\mathrm{TPSA}(m))$ $\times \mathbb{1}[m$ is isomer to $\mathrm{C_{16}H_{15}F_6N_5O}])^{1/4}$
valsartan_smarts	$(\mathbb{1}[s_1 \subseteq m]$ $\times G_{2.0165,0.2}(\log P(m))$ $\times G_{77.04,5}(\mathrm{TPSA}(m))$ $\times G_{896.38,30}(\mathrm{Bertz}(m)))^{1/4}$
deco_hop	$\mathrm{T}_{0.85}(\mathrm{sim}_{\mathrm{PHCO}}(m, \mathrm{m_1})) + \mathbb{1}[s_1 \subseteq m] + \mathbb{1}[s_2 \not\subseteq m] + \mathbb{1}[s_3 \not\subseteq m]$
scaffold_hop	$\mathrm{T}_{0.75}(\mathrm{sim}_{\mathrm{PHCO}}(m, \mathrm{m_1})) + \mathbb{1}[s_1 \not\subseteq m] + \mathbb{1}[s_4 \subseteq m]$

B

Additional figures from the PMO Benchmark

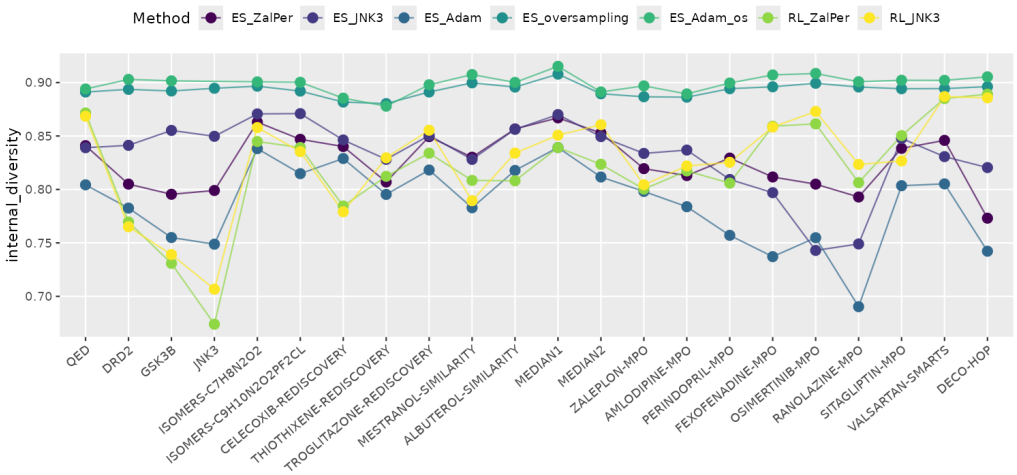

Figure B.1: Internal (ECF4) diversity on the PMO benchmark

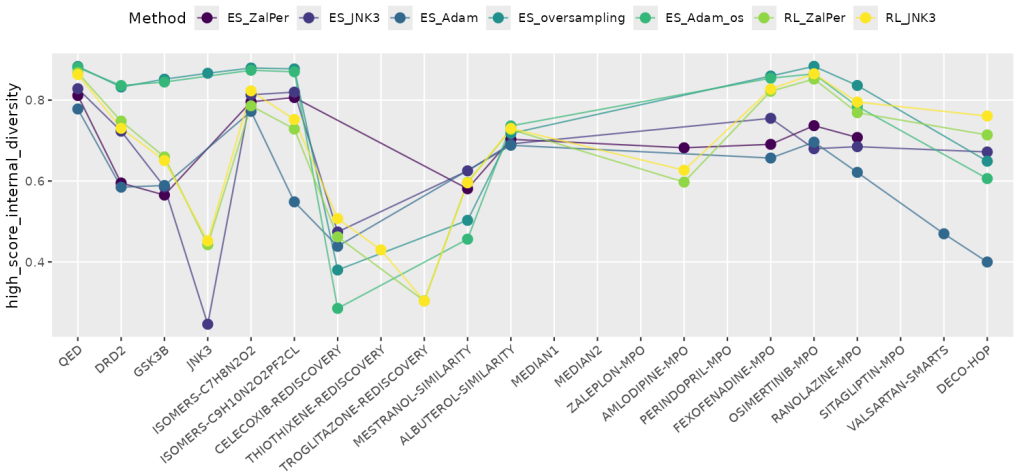

Figure B.2: Internal (ECF4) diversity on subset with score $\geq$ 0.6 on the PMO benchmark

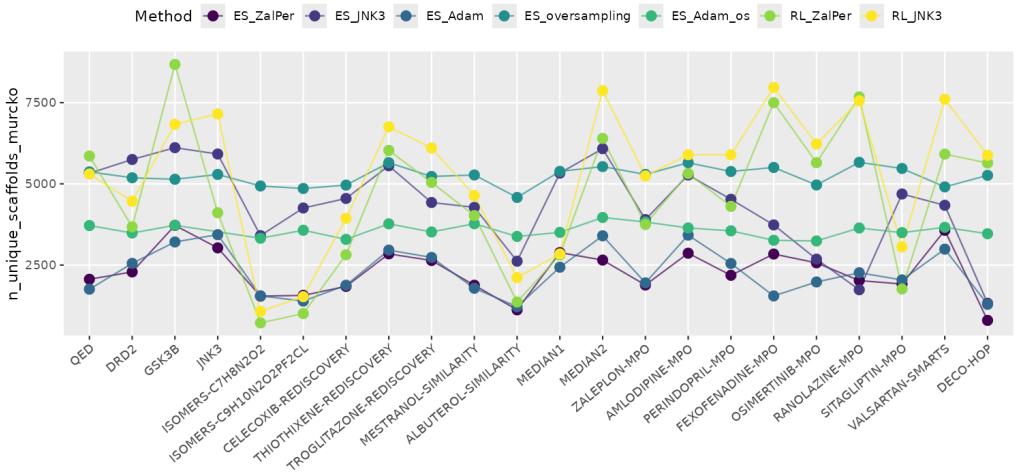

Figure B.3: Number of unique murcko scaffolds on the PMO benchmark

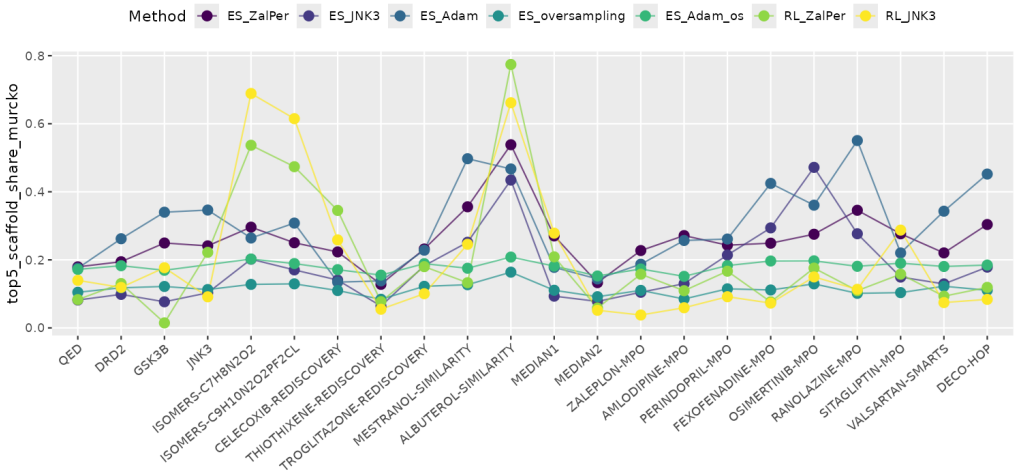

Figure B.4: Proportion of SMILES belonging to the five most common murcko scaffolds on the PMO benchmark

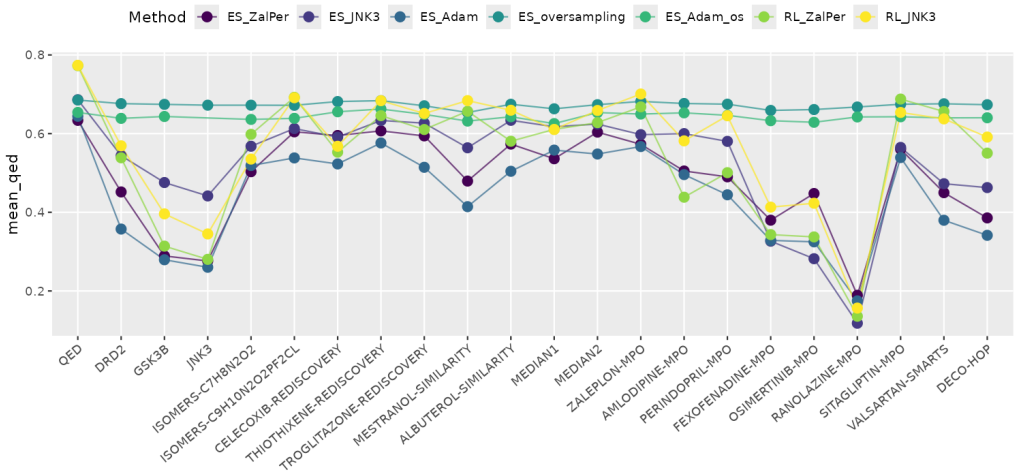

Figure B.5: Mean QED on the PMO benchmark

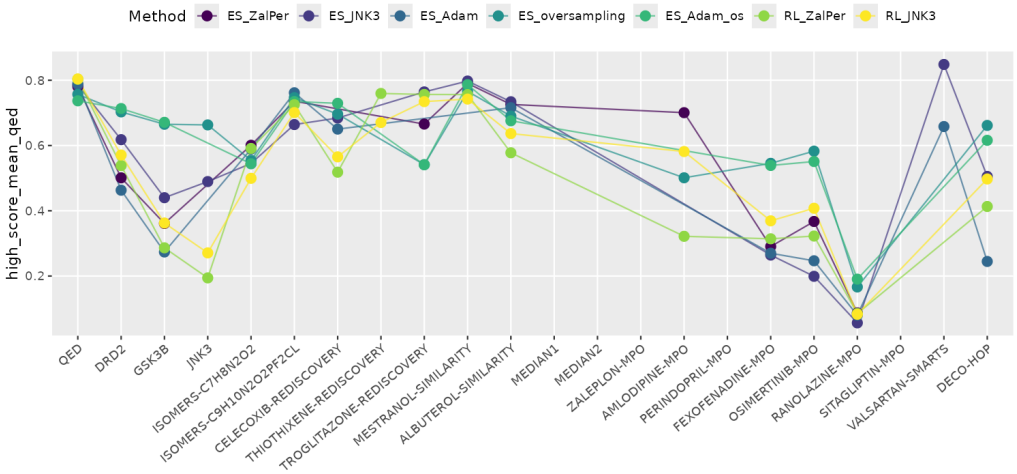

Figure B.6: Mean QED with score ≥ 0.5 on the PMO benchmark