Digital Discovery

ROYAL SOCIETY OF CHEMISTRY

PAPER

Published on 22 July 2026

Cite this: DOI: 10.1039/d6dd00190d

Exhaustive molecular string enumeration for data augmentation and structure exploration

AkshatKumar Nigam,[a] Serhii Tretiakov[b] and Robert Pollice *[b]

Received 14th April 2026
Accepted 21st July 2026

DOI: 10.1039/d6dd00190d

rsc.li/digitaldiscovery

Molecular strings such as SMILES and SELFIES are linear representations of molecular graphs. Due to the inherent nonlinear structure of molecular graphs, *a priori*, there is no unique string representation for a given molecular graph but rather many alternative strings correspond to the same graph. This allows for the randomization of molecular strings to be used both as a means of data augmentation for training sequence-based deep learning models and, when applied to SELFIES strings, for the systematic exploration of chemical space through random string mutations and interpolations. However, established algorithms for the randomization of molecular strings only account for the randomization of the starting atom and the main and sub branches. They do not randomize the spanning trees of the molecular graphs. This results in a substantial number of overlooked molecular strings for structures with at least one ring. Herein, we present TYCHE, an algorithm for the exhaustive enumeration of all spanning trees of molecular graphs and, thus, for the generation of many more randomized molecular strings. TYCHE shows a systematic and robust performance increase when used for data augmentation, string mutation, and string interpolation. Several case studies showcase the large potential of TYCHE for impact in cheminformatics and molecular machine learning.

1 Introduction

Molecular graphs[1] are the basis for all well-established molecular string representations such as SMILES,[2,3] InChI,[4,5] and SELFIES.[6,7] Typical applications of these strings, especially SMILES and InChI, are the representation of structures on computers, particularly as (unique) identifiers in chemical databases.[3,4] In addition, with the rise of machine learning (ML) in the past decade,[8,9] they have been used as efficient sequence representations for molecules.[10] They can be used directly as input for native sequence-based deep learning (DL) architectures such as recurrent neural networks (RNNs)[11,12] and architectures that can be readily adapted to sequences like transformer networks.[13,14] SELFIES has been developed for ML applications, with its 100% robustness (*i.e.*, every random combination of SELFIES characters corresponds to a valid molecular graph) being ideal for both deep generative models and interpretability.[6]

In general, except for the InChI, there are multiple equally valid strings for the same molecular graph.[2,6] Whereas applying molecular strings as unique identifiers in databases is reliant on efficient canonicalization algorithms,[3,15,16] several other application areas actually benefit from utilizing synonymous strings. For instance, generative models profit from augmentation with synonymous strings,[17] especially with limited training data. Additionally, combining SELFIES with both the systematic generation of synonymous strings and random string mutation makes for a simple and powerful algorithm for molecular space exploration.[18]

For these applications, string randomization should provide both diverse and unbiased synonyms to maximize associated performance gains. Especially for structure exploration, the complete molecular space captured by SELFIES strings is only reachable if all unique string synonyms can be generated. A SMILES or SELFIES string arises from a molecular graph that has been reduced to a tree by disconnecting all ring closures. In mathematical terms, such a structure is called a spanning tree. Since a ring system can be broken in many equivalent ways, each choice defines a different spanning tree, and these alternative trees are one of the principal sources of the multitude of possible string representations. However, the current implementation of string randomization in RDKit,[19] which is the *de facto* standard, preserves the spanning tree of the source string, which limits the number of unique synonymous strings it can generate. Consequently, for structures with at least one ring, most of the synonymous strings are neglected. In this work, we implement TYCHE, an exhaustive and efficient algorithm for generating synonymous SMILES and SELFIES strings that accounts for all main randomization mechanisms (Fig. 1). We show the impact of TYCHE on data augmentation in DL and molecular space exploration through representative case

[a]*Department of Computer Science, Stanford University, USA*
[b]*Stratingh Institute for Chemistry, University of Groningen, Nijenborgh 3, 9747 AG Groningen, The Netherlands. E-mail: r.pollice@rug.nl*

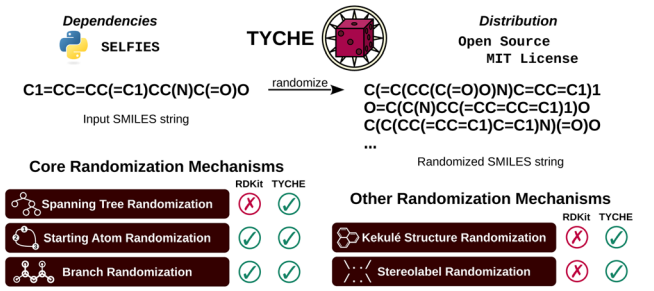

Fig. 1 Outline of this work. TYCHE is a Python package that implements all principal randomization mechanisms for SMILES strings.

studies. Overall, TYCHE provides a general framework for working with synonymous molecular string representations.

2 Methods

TYCHE is implemented as a Python package with SELFIES v2 (ref. 7 and 20) as its only external dependency. TYCHE requires a SMILES string as input and operates on both the string and the graph level. The output is a synonymous SMILES string, which can be converted to the corresponding SELFIES string.

2.1 Algorithmic overview

To illustrate the main mechanisms for randomizing molecular strings, it is instructive to inspect a general algorithm for converting molecular graphs to corresponding strings (Fig. 2):

1. Find a spanning tree.
2. Select a starting node.
3. Select a main branch and sub-branches.
4. Traverse the spanning tree and write down string.

This algorithm applies to both SMILES and SELFIES with the only difference in how strings are written down (step 4). The first three steps determine the string that will be obtained in step 4. Each of these three steps allows for many valid outcomes. Hence, they provide independent randomization mechanisms. TYCHE implements dedicated functions for each of them. Next to these main mechanisms, both kekulization and stereochemistry labels allow for additional randomization, which TYCHE supports. The following paragraphs sketch the core functionality of TYCHE.

2.1.1 Spanning tree randomization. Ring bonds are shuffled randomly within each ring. This is equivalent to randomizing the spanning tree and is not performed by RDKit.

2.1.2 Starting node randomization. The starting node is changed randomly to another node.

2.1.3 Branch randomization. The priorities of all branches are shuffled randomly. This is equivalent to randomizing both main and sub-branches.

2.1.4 Kekulé structure randomization. For each delocalized (aromatic) subgraph, we randomly permute the atom priorities, thereby effectively randomizing the Kekulé assignment of single and double bonds within that subgraph. This is not performed by RDKit.

2.1.5 Stereochemistry label randomization. Parity labels of stereogenic centers are corrected after branch randomization to avoid scrambling of stereochemistry. Corresponding double bond stereochemistry labels are inverted at random with preservation of the assigned configuration, which is not performed by RDKit.

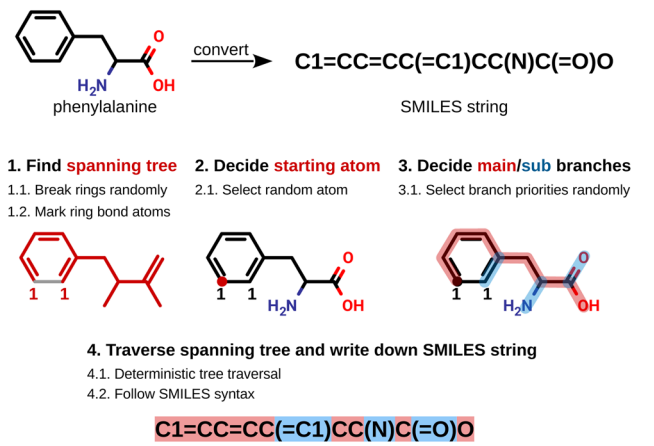

Fig. 2 Schematic outline of a general algorithm for the conversion of molecular graphs to molecular strings illustrated based on SMILES.

2.2 Algorithm testing

The algorithms implemented in TYCHE were tested for correctness on diverse molecule datasets by performing string randomization and checking whether the output still represents the same molecule as the input. For testing, we selected the ZINC-250k subset[21] of the ZINC dataset,[22] the CEP_{SUB} subset[23] of the Harvard Clean Energy Project dataset,[24] the $GDB\text{-}13_{SUB}$ subset[23] of the systematic small molecule dataset GDB-13,[25] the DTP_{SUB} subset[23] of the Developmental Therapeutics Program Open Compound Collection,[26–29] and the *syn*-sesquinorbornene dataset SNB-60K.[23] Across all these diverse datasets (839 027 SMILES in total), TYCHE randomized the strings without any errors or changes to molecular structure or configuration.

2.3 Application examples

We compared TYCHE to the SMILES string randomization as implemented in RDKit (version 2025.9.3).[19] The following subsections outline the specific methodology adopted for each of the application examples.

2.3.1 SMILES randomization. We selected 4 representative molecules from the ESOL dataset[30] with 0, 1, 2, and 3 rings, respectively, and started with their canonical SMILES strings. For each data point, we performed a fixed number of string randomization cycles with either RDKit or TYCHE. Each cycle could reproduce a previously seen string or generate a new, unique SMILES for that molecule. We then counted unique SMILES, and averaged the count over 5 independent runs. The results for each algorithm were fitted to an additive-inverted exponential decay equation as a guide to the eye. We repeated this procedure with 2 additional representative molecules, naphthalene and (*Z*)-stilbene, to investigate the contributions of

distinct randomization mechanisms to the total number of unique SMILES strings generated. Hence, we repeated the TYCHE randomizations multiple times and turned off specific randomization mechanisms. We also performed 1 000 000 string randomization cycles for each molecule from the ESOL, FreeSolv, and LIPO datasets with either RDKit or TYCHE, and counted the number of unique SMILES generated. The results were filtered for entries with at most 750 000 unique SMILES strings obtained *via* TYCHE to minimize skewing due to incomplete randomization. In practice, this discards molecules whose SMILES space is so large that it is not sufficiently explored within 1 000 000 cycles. For each subset of molecules with the same number of rings, we looked at the ratio between the total number of unique SMILES strings generated by TYCHE or RDKit. To benchmark the runtime of TYCHE *versus* RDKit, we timed 25 string randomization roundtrips with each algorithm on 1 CPU for all molecules in the DTP_{SUB} (105 338 structures) and GDB-13_{SUB} (398 453 structures) datasets.[23] For analysis, runtime values were divided by the lowest runtime observed in a systematic set. For the stress test with extreme molecules, we timed 100 string randomization roundtrips with each algorithm on 1 CPU and confirmed that randomization maintained the original structure faithfully.

2.3.2 Training data augmentation. The effect of training data augmentation *via* string randomization was demonstrated on the Freesolv,[31] ESOL,[32] and LIPO[30] datasets by adopting the following procedure for each dataset. We converted all molecules to canonical kekulized SMILES with RDKit.[19] For each string, we then repeatedly applied randomization with either RDKit or TYCHE in batches of 256 randomization cycles, collecting unique kekulized strings (including the original) until either 256 distinct strings were obtained or a maximum of 50 runs (12 800 randomization cycles in total) was reached.

In order to split the data, we clustered molecules by substructure using RDKit fingerprints (details in the SI Material), assigning each molecule to exactly one cluster. To balance ring-containing and non-ring structures, clusters were labeled as predominantly cyclic or acyclic, and train-validation splits were then created by random stratified splitting over cluster IDs within these two groups, yielding 10 distinct cluster-based splits. For each split, we evaluated five training augmentation settings: (i) original data with canonical kekulized SMILES only, (ii) fully augmented data with up to 256 unique SMILES per molecule, and (iii–v) random subsets of the augmented data capped at 4, 16, or 64 unique strings per molecule. Validation sets were either not augmented or augmented 256-fold with either TYCHE or RDKit. Therefore, we had 50 train-validation pairs in total.

We created 384-dimensional embeddings for all unique SMILES strings using the `DeepChem/ChemBERTa-77M-MLM` checkpoint[33] of the ChemBERTa model.[34] For each individual split and subset, we trained a fully connected feedforward neural network using the ChemBERTa embeddings as features. The Gibbs free energy of solvation was the target for Freesolv, the logarithm of the aqueous solubility for ESOL, and the logarithm of the *n*-octanol–water distribution coefficient for LIPO. The model hyperparameters are provided in the SI Material. Prediction performance was compared *via* mean absolute error (MAE), root mean square error (RMSE), and the coefficient of determination (R^2) on three validation sets for each split and subset: the matching validation dataset, its unaugmented, and its fully augmented version. Each performance metric is reported as the first, second, and third quartile across the 10 distinct training-validation splits.

2.3.3 SELFIES mutation. Starting from the SMILES of an input structure, we perform string randomization 1 million times with either RDKit or TYCHE and collect all unique synonymous SELFIES representations. In case 1 million randomization runs is insufficient for exhaustive string enumeration, we perform it 100 million times. Subsequently, following the literature,[18] we perform one, two, three, four, and five random point mutations on each unique SELFIES string to generate new structures. Structure similarities between input and output structures are evaluated with RDKit.

2.3.4 SELFIES chemical paths. Starting from the SMILES strings of two input structures, we perform string randomization of each multiple times with either RDKit or TYCHE and collect all unique alternative SELFIES per input. Subsequently, following the literature,[18] we create chemical paths interconverting the two structures progressively by randomly pairing SELFIES representations of the two input structures, defining one as start and the other as target. Then, we harmonize the string lengths through padding and compare the characters of the strings at each position. Subsequently, we randomly pick a position with differing characters and modify the start to equalize them, under the condition of strictly increasing extended-connectivity fingerprint similarity to the target.[35] This is repeated until the starting point is transformed into the target. All structures encountered therein form the chemical path.

For comparing randomization algorithms, we used the chemical paths between two reference molecules to generate median molecules, *i.e.*, structures resembling both references. Performance was assessed *via* the revised joint similarity metric penalizing asymmetric similarity to only one reference.[18] For each reference, 25 randomized SMILES were generated. All 625 string combinations between the references were used for bidirectional chemical path generation. For each combination, 25 random mutation sequences were followed in each direction, providing 31 250 chemical paths per reference pair. The top 100 structures with highest joint similarities were collected for evaluation.

2.3.5 GuacaMol benchmarks. We modified the existing implementation of STONED,[18] a simple algorithm for structure exploration and interpolation, by replacing RDKit randomization with TYCHE and carried out all benchmarks in GuacaMol following the literature.[36] For both TYCHE and RDKit randomization, 10 million randomized SMILES strings were generated for molecule generation within STONED. Each benchmark was performed once. As baselines, we took literature results for various inverse design algorithms.[36–40]

2.3.6 Tartarus benchmarks. We modified the existing implementation of JANUS,[39] the parallel-tempered genetic algorithm guided by deep neural networks (DNNs), by replacing

RDKit-based string randomization with TYCHE in the SELFIES-based genetic operators mutation and crossover. With this modified version of JANUS, we performed both the organic photovoltaics and the organic emitters design benchmarks from the TARTARUS suite following the literature procedure.[23] Each benchmark was repeated 5 times and the final results were obtained as mean and standard deviation from the individual results. As baselines, we took the literature results.[23]

3 Results

The following sections demonstrate the impact of TYCHE on both systematic benchmarks and representative case studies.

3.1 SMILES randomization

To illustrate the systematic differences of randomization with TYCHE and RDKit, we selected 4 molecules from the ESOL dataset[32] with 0 to 3 rings, respectively, and performed repeated randomization of their SMILES representations. Fig. 3A–D shows the number of unique isomeric SMILES generated as a function of the number of randomization cycles. For molecule **1**, which does not contain any rings, randomization dynamics of both TYCHE and RDKit are identical and both generate the same number of unique strings. In contrast, for molecules **2–4**, which contain 1–3 rings, respectively, TYCHE creates increasingly more unique SMILES strings relative to RDKit. Notably, the results for molecule **2** (*cf.* Fig. 3) provide an estimate for how much of the increased diversity obtained *via* TYCHE comes from spanning tree randomization relative to starting atom randomization and branch randomization. This is because molecule **2** does not benefit from either Kekulé structure randomization or stereolabel randomization. For molecule **4**, while 30 000 randomization cycles reach the limit of unique strings for RDKit, TYCHE is still in the linear regime with no signs of saturation. Notably, when we inspected the random SMILES strings for molecules **2** and **3**, we found that spanning tree randomization samples the distinct spanning trees approximately in a uniform manner (*cf.* SI Material).

To investigate the contribution of the distinct randomization mechanisms, we performed ablations for 2 selected molecules. Naphthalene (**5**) was selected for its contribution from Kekulé structure randomization and (*Z*)-stilbene (**6**) was selected for its contribution from stereochemistry label randomization. The corresponding results obtained from repeated SMILES string randomization are illustrated in Fig. 4. These results suggest that while spanning tree randomization contributes most significantly to the total number of unique SMILES strings

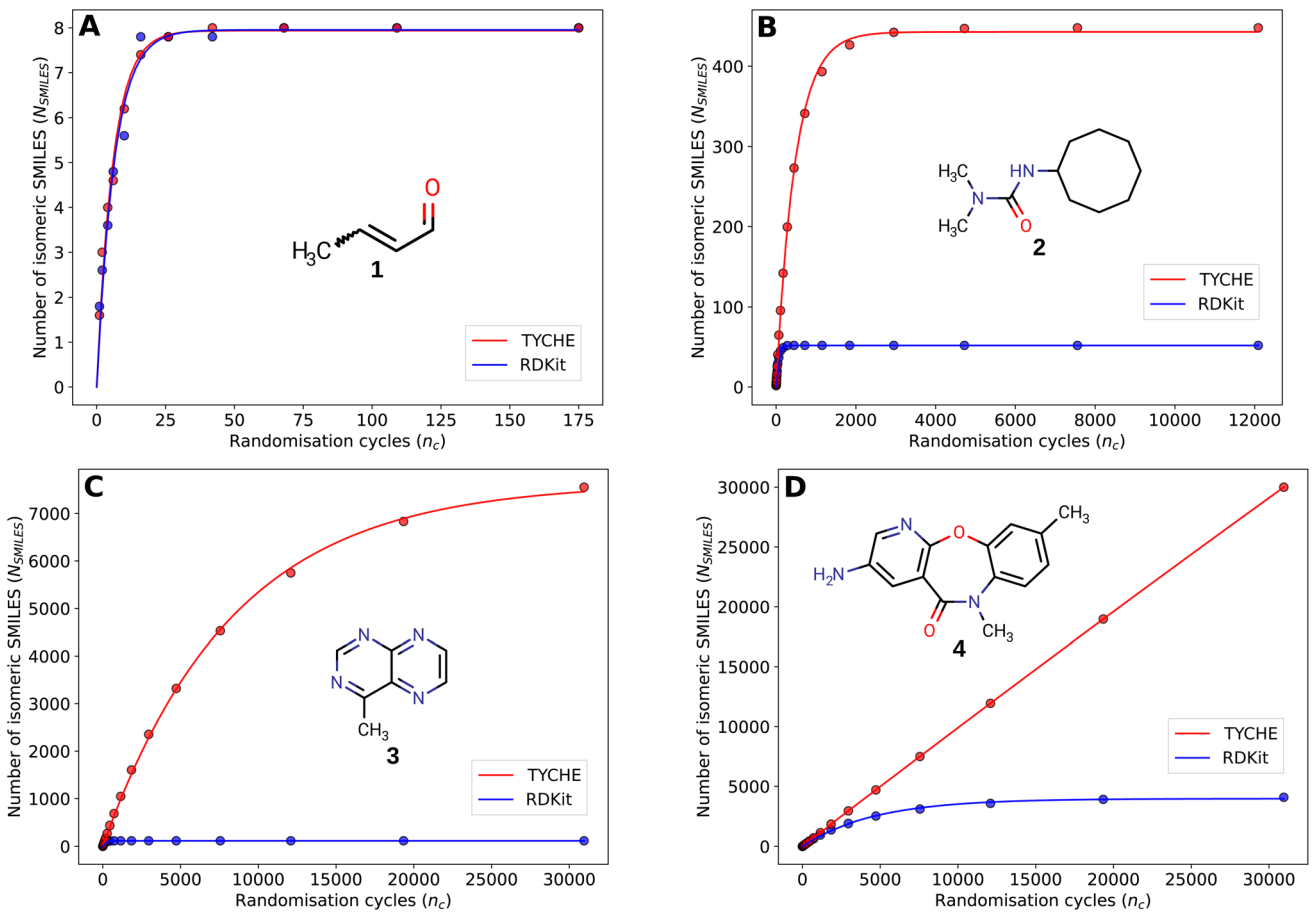

Fig. 3 Comparison of the number of unique SMILES strings generated *versus* the number of successive randomization cycles performed between RDKit and TYCHE for molecules with (A) zero, (B) one, (C) two, and (D) three rings. Fitted curves are added as a guide to the eye.

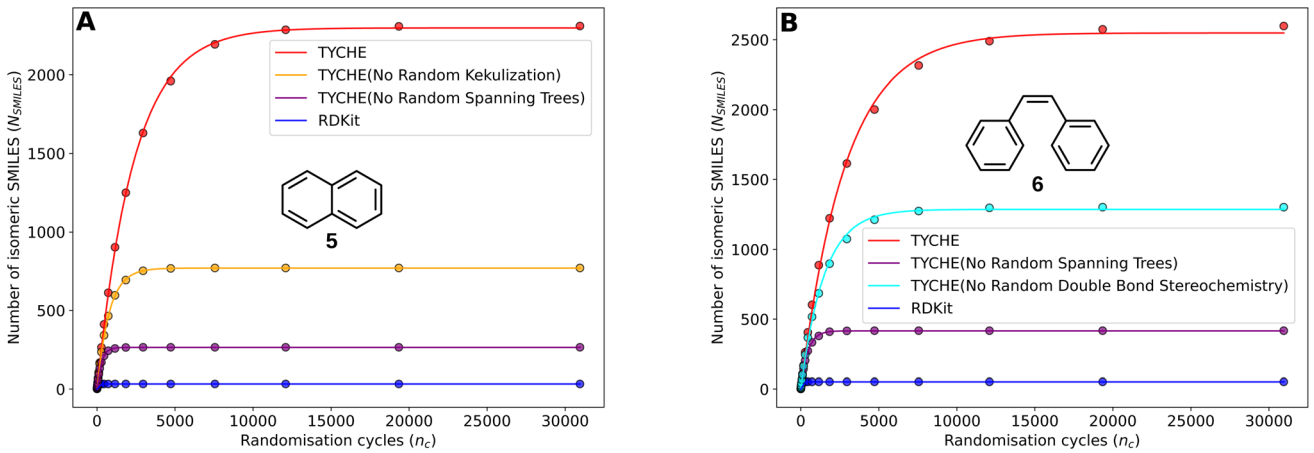

Fig. 4 Comparison of unique SMILES strings generated *versus* the number of randomization cycles performed between RDKit and TYCHE when specific randomization mechanisms are turned off for (A) naphthalene and (B) (*Z*)-stilbene. Fitted curves are added as a guide to the eye.

Table 1 Median ratios of unique SMILES strings generated by TYCHE relative to RDKit for molecules from the ESOL, FreeSolv, and LIPO datasets grouped by the ring count

Parameter	Values			
Number of rings	0	1	2	3
Ratio of unique strings	1.0	7.9	78.6	269.6

generated for these molecules, the contributions of either Kekulé structure randomization for naphthalene or stereochemistry label randomization for (*Z*)-stilbene are still considerable. Notably, for (*Z*)-stilbene, removing Kekulé structure randomization did not reduce the number of unique SMILES strings generated.

Table 1 summarizes the median ratios of unique SMILES representations generated by TYCHE relative to RDKit for molecules from the ESOL, FreeSolv, and LIPO datasets that were sufficiently close to saturation within 1 000 000 randomization cycles. All molecules with more than 3 rings would require substantially more randomization cycles to reach saturation. The results show that whereas both TYCHE and RDKit provide the same number of unique SMILES for molecules with no rings, the more rings in a molecule, the more unique strings are generated by TYCHE. In the SI Material, we included distributions for these three datasets illustrating the advantage of TYCHE over RDKit with respect to the number of unique SMILES strings generated.

We were also interested in comparing the runtime of TYCHE to RDKit. Hence, we timed the randomization of all molecules in the DTP_{SUB} dataset.[23] Fig. 5A depicts the scaling of the median runtime with the number of non-hydrogen atoms for both TYCHE and RDKit. The regression lines show good agreement with linear size scaling for both algorithms. The ratio of the corresponding slopes provide an asymptotic estimate of 1.78 for the relative timing of TYCHE compared to RDKit. Fig. 5B shows the ratio of the median runtime of TYCHE randomization relative to RDKit depending on the number of

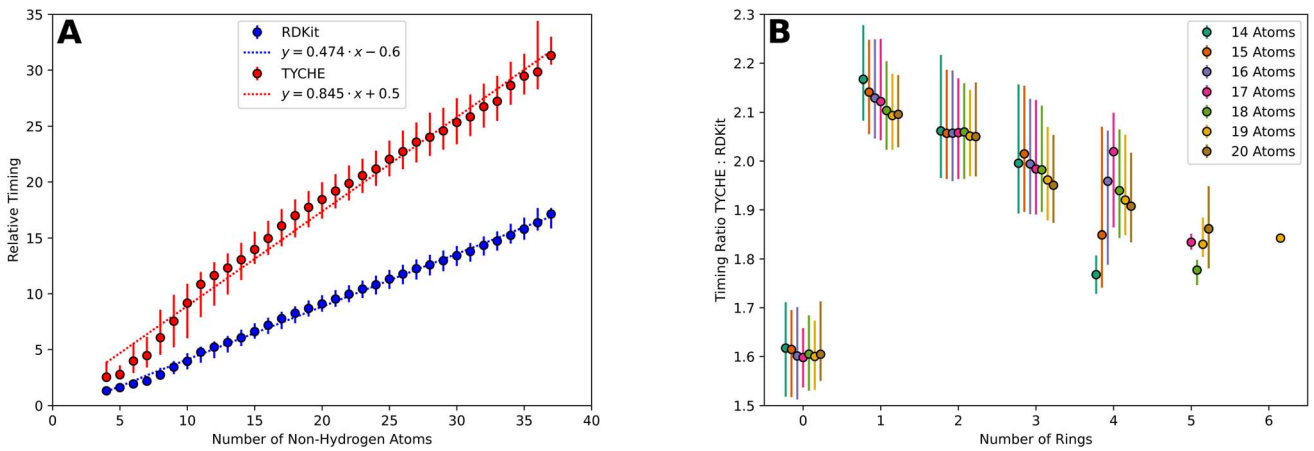

Fig. 5 Timing comparison of randomization with either TYCHE or RDKit on the DTP_{SUB} dataset. (A) Relative timing depending on the number of non-hydrogen atoms with linear regressions. (B) Timing ratio depending on the number of rings for subsets with a fixed number of non-hydrogen atoms. Values are medians over 25 repetitions and error bars are between first and third quartiles.

Table 2 Timing comparison of randomization with either TYCHE or RDKit on extreme molecules. Relative timing is the ratio of median timings from TYCHE and RDKit, respectively over 100 repetitions

Molecule	Distinctive feature	Relative timing
Polyaromatic hydrocarbon	43 annulated rings	0.95
Macrocycle 1	52-membered ring	1.92
Macrocycle 2	46-membered ring	1.05
Maitotoxin	98 stereocenters	3.00
Palytoxin	64 stereocenters	3.65

rings for subsets with a fixed number of non-hydrogen atoms. It reveals a significant increase in the timing ratio when introducing the first ring. Subsequent introduction of additional rings results in decreasing timing ratios. Additional results for the GDB-13_{SUB} dataset show analogous trends and are provided in the SI Material.

Finally, we performed a stress test of TYCHE by comparing it to RDKit on a few extreme molecules. First, we picked a very large polycyclic aromatic hydrocarbon with 114 carbons and 43 annulated rings that has previously been synthesized.[41] Second, we looked into two macrocyclic compounds that have previously been prepared, one with a 52-membered ring,[42] the other with a 46-membered ring[43] in the corresponding SMILES string. Third, we were interested in molecules with high stereochemical complexity and took two of the most complex known natural products, maitotoxin,[44,45] which consists of 234 non-hydrogen atoms and contains 98 stereocenters and 2 double bonds with stereochemistry, and palytoxin,[46] which consists of 186 non-hydrogen atoms and contains 64 stereocenters and 7 double bonds with stereochemistry. Our timing results are summarized in Table 2. For molecules with high stereochemical complexity, TYCHE can show a 3 to 4-fold increase in timing. However, for the polyaromatic hydrocarbon, TYCHE even outperforms RDKit, suggesting that the trend observed in Fig. 5B is systematic and TYCHE handles additional rings faster. The SMILES strings of these structures are provided in the SI Material.

3.2 Training data augmentation

The effect of randomization on supervised learning performance is shown in Fig. 6–8. We consider three datasets: ESOL, FreeSolv and LIPO. For each, we aim to decouple the effects of randomization applied separately to the training and validation sets, using either RDKit or TYCHE. Each panel in Fig. 6–8 keeps the validation set fixed, either unaugmented (canonical) or augmented 256-fold with RDKit or TYCHE, while the trend lines show the validation loss (MAE) as a function of the training set augmentation factor, again using either RDKit or TYCHE.

While training on data augmented with random SMILES either does not significantly improve performance on unaugmented validation data or even degrades it (*cf.* Fig. 6A–8A), benefits are seen for augmented validation data. When the validation data is augmented with RDKit, both TYCHE and RDKit randomizations improve performance consistently with increasing augmentation of the training set (*cf.* Fig. 6B–8B). For ESOL and FreeSolv, both randomization methods provide similar performance gains. Only for LIPO, augmentation of the training set with RDKit results in significantly better validation performance when the validation data is augmented in the same way. The results for LIPO also show that moderate augmentation during training can result in significant performance drops on the unaugmented validation set compared to when no augmentation is used. When training data augmentation is increased, performance on the unaugmented validation set is recovered.

In contrast, for validation data augmented with TYCHE, only TYCHE randomization consistently improves performance (*cf.* Fig. 6C–8C). In that case, TYCHE outperforms RDKit significantly for all three datasets when data augmentation is highest and RDKit augmentation even tends to degrade performance at the highest augmentation factor. Results using the coefficient of determination (R^2) and the root mean squared error (RMSE) as alternative metrics are in the SI Material. Overall, these results suggest that TYCHE augmentation is important for prediction performance on randomized SMILES strings but does not show improved chemical generalization on unaugmented validation sets.

3.3 SELFIES mutation

Following previous work,[18] we explored the structural neighborhood of the known drugs aripiprazole, albuterol, and mestranol[36] with a modified version of STONED[18] that supports TYCHE randomization. The corresponding results are summarized in Table 3. As all these drugs contain rings, the number of

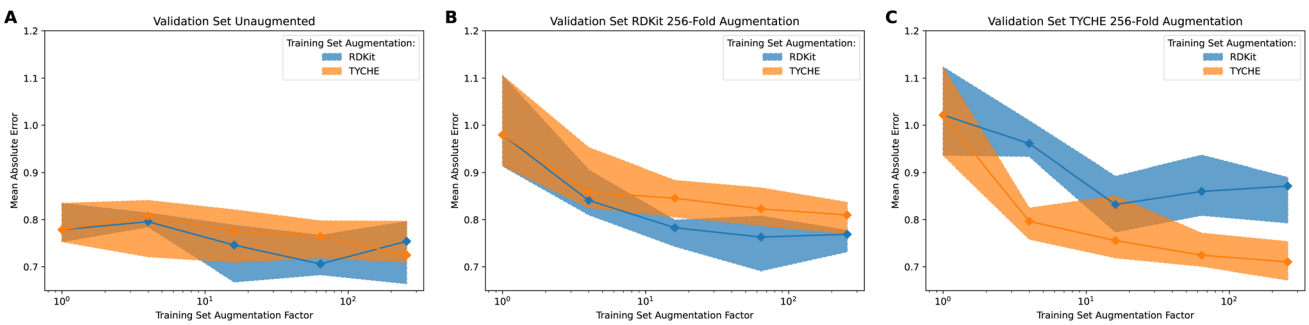

Fig. 6 Validation performance on the ESOL dataset[32] with increasing randomized SMILES augmentation *via* either RDKit or TYCHE. Mean absolute errors (MAEs) were evaluated on the (A) unaugmented, (B) RDKit-augmented, and (C) TYCHE-augmented validation set. Trend lines are medians, errors are between first and third quartile across 10 distinct splits.

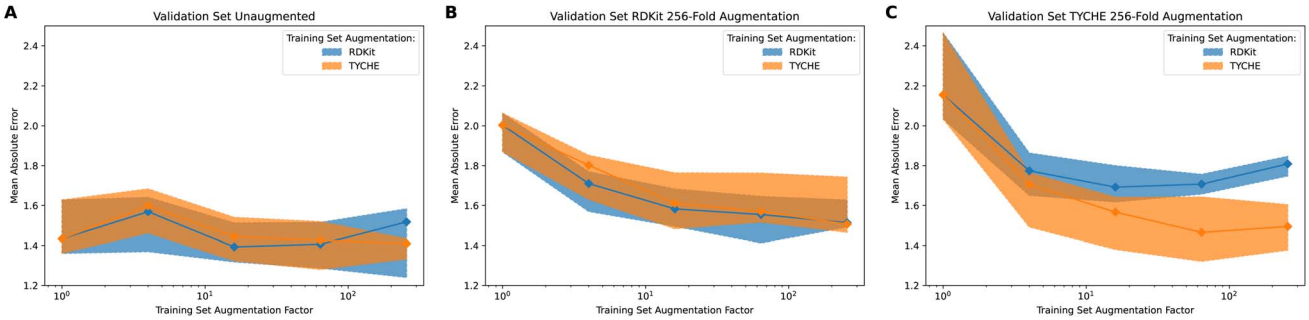

Fig. 7 Validation performance on the FreeSolv dataset[31] with increasing randomized SMILES augmentation *via* either RDKit or TYCHE. Mean absolute errors (MAEs) were evaluated on the (A) unaugmented, (B) RDKit-augmented, and (C) TYCHE-augmented validation set. Trend lines are medians, errors are between first and third quartile across 10 distinct splits.

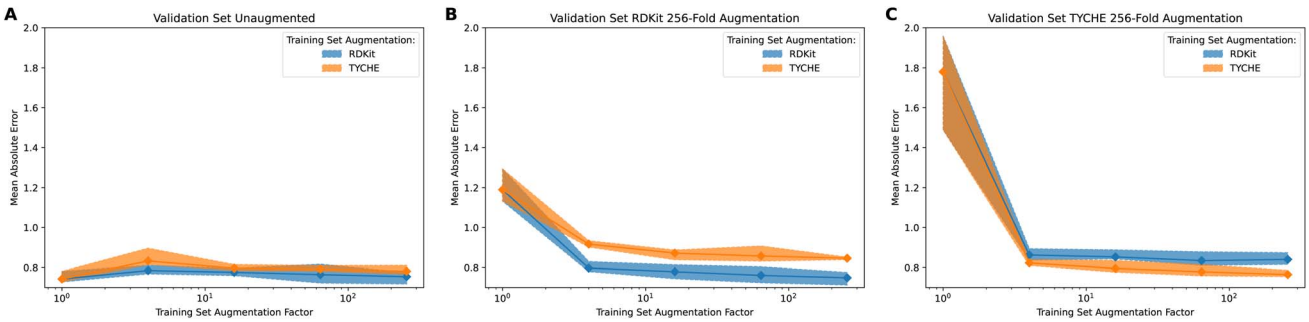

Fig. 8 Validation performance on the LIPO dataset[30] with increasing randomized SMILES augmentation *via* either RDKit or TYCHE. Mean absolute errors (MAEs) were evaluated on the (A) unaugmented, (B) RDKit-augmented, and (C) TYCHE-augmented validation set. Trend lines are medians, errors are between first and third quartile across 10 distinct splits.

unique SELFIES strings obtained *via* TYCHE is 1 order of magnitude higher for albuterol and several orders of magnitude higher for both aripiprazole and mestranol. Consequently, TYCHE allows generating a substantially larger number of similar structures from the original molecules. Notably, for both aripiprazole and mestranol, we do not even reach saturation of unique SMILES after 100 million randomizations with TYCHE, whereas RDKit randomization saturates within 1 million cycles for all three molecules.

3.4 SELFIES chemical paths

The impact of the improved randomization provided by TYCHE is also apparent for the generation of chemical paths with SELFIES. When assessing the joint similarities of the molecules generated through chemical paths[18] between either tadalafil[47] and sildenafil[48] or between dihydroergotamine[49] and prinomastat,[50] 4 well-known drugs, we found that the use of TYCHE results in structures with consistently larger joint similarities in the 100 top candidates as summarized in Table 4. Notably, all four reference structures contain at least 3 cycles and are quite large. Hence, the 25 randomization cycles performed are very far from being exhaustive for either TYCHE or RDKit. Selected median molecules and their joint similarities are depicted in Fig. 9.

3.5 GuacaMol benchmarks

To assess the impact of TYCHE on structure exploration through SELFIES modification, we performed the GuacaMol benchmarks[36] with our modified version of STONED.[18] The

Table 3 Comparison of the generation of local chemical subspaces *via* SELFIES mutation with either TYCHE or RDKit string randomization. Molecules are grouped by fingerprint-based similarity thresholds (δ) relative to their input structures

		Unique mutated structures (fraction of total)			Unique random
Starting structure	Fingerprint	$\delta > 0.75$	$\delta > 0.60$	$\delta > 0.40$	SELFIES
Aripiprazole (RDKit)[18]	ECFP4	511 (1.50%)	1934 (5.67%)	8457 (24.80%)	7776
Aripiprazole (TYCHE)	ECFP4	**40 380 (0.01%)**	**1 528 645 (0.50%)**	**31,881 841 (10.51%)**	**57,376 874**
Albuterol (RDKit)[18]	FCFP4	76 (3.68%)	209 (10.11%)	384 (18.59%)	432
Albuterol (TYCHE)	FCFP4	**480 (1.67%)**	**1541 (5.36%)**	**3368 (11.71%)**	**6528**
Mestranol (RDKit)[18]	AP	852 (0.77%)	4075 (3.68%)	26 989 (24.36%)	25 688
Mestranol (TYCHE)	AP	**83 809 (0.02%)**	**1 833 634 (0.54%)**	**47,854 668 (14.05%)**	**79,254 498**

Table 4 Joint similarities of top 100 median molecules generated by TYCHE and RDKit along chemical paths between two reference molecules

			Joint similarity		
Reference 1	Reference 2	Algorithm	1st quartile	Median	3rd quartile
Tadalafil	Sildenafil	RDKit	0.260	0.265	0.275
Tadalafil	Sildenafil	TYCHE	**0.296**	**0.301**	**0.312**
Dihydroergotamine	Prinomastat	RDKit	0.229	0.234	0.241
Dihydroergotamine	Prinomastat	TYCHE	**0.285**	**0.290**	**0.298**

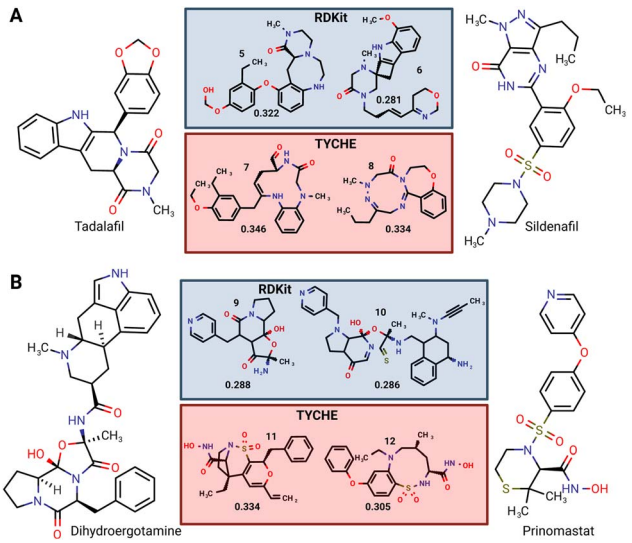

Fig. 9 Selection of median molecules obtained *via* RDKit or TYCHE randomization from chemical paths between (A) tadalafil and sildenafil, (B) dihydroergotamine and prinomastat. Values are joint similarities.

corresponding results, alongside representative inverse design algorithms as literature baselines, are summarized in Table 5. While these baselines all rely on feedback for molecule generation *via* the respective performance scores, STONED modifies strings randomly without any feedback.[18] Our results show that STONED with TYCHE randomization outperforms RDKit randomization significantly. Performance improves for all but 1 task, resulting in a substantially improved total score. While STONED with RDKit randomization cannot compete with any of the inverse design algorithms, TYCHE brings it into the same performance range.

3.6 Tartarus benchmarks

Finally, we wanted to evaluate the impact of TYCHE on the performance of the inverse molecular design algorithm JANUS, a genetic algorithm that relies on STONED[18] for candidate generation and artificial neural networks for candidate selection.[39] Accordingly, we implemented a modified version of JANUS[39] that supports TYCHE randomization and used it to perform both the organic photovoltaics and the organic emitters design benchmarks of TARTARUS.[23] The corresponding

Table 5 Comparison of literature baselines for the GuacaMol benchmark suite[36] to modified versions of STONED that sample 10 million (10M) synonymous random strings making use of either TYCHE or RDKit randomization. MPO = multiproperty objective

Benchmark	REINVENT[40,51]	SMILES LSTM[36]	GB-GA[36,52]	JANUS (original)[39]	EvoMol[37]	GEGL[38]	STONED (10M, RDKit)	STONED (10M, TYCHE)
Celecoxib rediscovery	**1.000**	**1.000**	**1.000**	**1.000**	1.000	**1.000**	0.612	0.763
Troglitazone rediscovery	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.594	0.724
Thiothixene rediscovery	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.681	0.703
Aripiprazole similarity	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.734	**1.000**
Albuterol similarity	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.961	**1.000**
Mestranol similarity	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.785	**1.000**
$C_{11}H_{24}$	0.999	0.993	0.971	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
$C_9H_{10}N_2O_2PF_2Cl$	0.877	0.879	0.982	**1.000**	**1.000**	**1.000**	0.931	**1.000**
Median molecules 1	0.434	0.438	0.406	0.434	**0.455**	**0.455**	0.397	0.426
Median molecules 2	0.395	0.422	0.432	0.416	0.417	**0.437**	0.410	0.423
Osimertinib MPO	0.889	0.907	0.953	0.967	0.969	**1.000**	0.891	0.889
Fexofenadine MPO	**1.000**	0.959	0.998	0.999	**1.000**	**1.000**	0.910	0.923
Ranolazine MPO	0.895	0.855	0.920	0.920	0.957	**0.958**	0.850	0.884
Perindopril MPO	0.764	0.808	0.792	0.817	0.827	**0.882**	0.659	0.682
Amlodipine MPO	0.888	0.894	0.894	0.905	0.869	**0.924**	0.758	0.804
Sitagliptin MPO	0.539	0.545	0.891	0.901	**0.926**	0.922	0.632	0.642
Zaleplon MPO	0.590	0.669	0.754	0.774	0.793	**0.834**	0.698	0.712
Valsartan SMARTS	0.095	0.978	0.990	0.993	0.998	**1.000**	0.884	0.893
Deco hop	0.994	0.996	**1.000**	**1.000**	**1.000**	**1.000**	0.975	**1.000**
Scafold hop	0.990	0.998	**1.000**	**1.000**	**1.000**	**1.000**	0.874	**1.000**
Total score	16.350	17.340	17.983	18.126	18.211	**18.412**	15.236	16.713

Table 6 Results for the organic photovoltaics benchmarks. All results except JANUS (TYCHE) are from the literature.[23] Results are provided as mean ± standard deviation of the best target objective values from five independent runs. Metrics: PCE_{PCBM}: PCBM power conversion efficiency; PCE_{PCDTBT}: PCDTBT power conversion efficiency; SAscore: synthetic accessibility score.[53] Arrows indicate optimum

	PCE_{PCBM} − SAscore (↑)	PCE_{PCDTBT} − SAscore (↑)
Dataset	7.57	31.71
SMILES-VAE	7.44 ± 0.28	10.23 ± 11.14
SELFIES-VAE	7.05 ± 0.66	29.24 ± 0.65
SMILES-LSTM-HC	6.69 ± 0.40	31.79 ± 0.15
SELFIES-LSTM-HC	7.40 ± 0.41	30.71 ± 1.20
REINVENT	7.48 ± 0.11	30.47 ± 0.44
GB-GA	7.78 ± 0.02	30.24 ± 0.80
JANUS (RDKit)	7.59 ± 0.14	31.34 ± 0.74
JANUS (TYCHE)	**7.81 ± 0.01**	**31.91 ± 0.12**

results are summarized in Tables 6 and 7. When leveraging TYCHE, JANUS not only outperforms its original implementation with RDKit randomization but also reaches state-of-the-art performance on all the individual benchmark tasks. This shows that the exploration capacity improvements of STONED introduced by TYCHE (*vide supra*) also impact downstream inverse design performance.

4 Discussion

We divide our discussion into two separate sections, one focusing on the algorithmic aspects of TYCHE and one focusing on the application examples presented above.

4.1 Algorithm

The randomization algorithm of TYCHE was implemented in a modular manner such that each type of randomization has its own function. This not only makes the code more user-friendly but it also allows for individual randomization mechanisms to be readily turned on or off. The associated Python package is lightweight, with the `selfies` package[20] as its only external dependency. Version 2 of `selfies` has no external dependencies, making TYCHE easy to maintain. All the randomization mechanisms of the `tyche` package are built on top of the capabilities of the `selfies` package to handle both SMILES strings and directed molecular graphs, and interconvert between these two objects.

The core feature of TYCHE is spanning tree randomization. This is implemented by a dedicated function that randomly shuffles the positions of all the ring bonds in each ring. While there are formally faster algorithms for generating all unique spanning trees of graphs systematically like Wilson's algorithm,[54] we opted for using an algorithm that guarantees for a new spanning tree to be selected randomly starting from an initial spanning tree instead. As long as we are not interested in obtaining all possible spanning trees in one randomization loop, which is typically not relevant for practical applications of TYCHE, a systematic algorithm like Wilson's algorithm is not required. While there is an implementation of Wilson's algorithm in the Boost Graph Library,[55] we wanted to avoid any additional dependencies of our package as much as possible. As demonstrated by our results for the number of unique SMILES strings as a function of randomization cycles (*cf.* Fig. 3), both the RDKit and the TYCHE randomization profiles follow approximately an additive-inverted exponential decay. Assuming there is a half-life in units of randomization cycles for the number of undiscovered unique strings, TYCHE should discover roughly 93.75% (*i.e.*, $\frac{15}{16}$) of all unique strings within only 4 half-lives. Hence, TYCHE can generate a large fraction of all possible unique strings efficiently, however, for discovering all of them, it is inefficient.

Apart from spanning tree randomization, TYCHE also implements randomization of Kekulé structures and double bond stereochemistry labels. Both these mechanisms are not available through RDKit. As SELFIES does not support aromatic (sub)graphs and only supports Kekulé structures as underlying molecular graphs,[6,7] randomizing the corresponding delocalized subgraphs directly impacts both data augmentation *via* string randomization and molecule generation *via* random string modification. Double bond geometry is not defined by absolute stereochemical labels but by the relative stereochemical labels of two bonds adjacent to a double bond. Inverting both these labels does not change the stereochemistry with respect to this double bond. The support of double bond

Table 7 Results for the organic emitters design benchmark objectives. All results except JANUS (TYCHE) are from the literature.[23] Results are provided as mean ± standard deviation of the best target objective values from five independent runs. Metrics: $\Delta E(S_1–T_1)$: singlet–triplet gap; f_{12}: S_1 and S_0 transition oscillator strength; $\Delta E(S_0–S_1)$: S_0 and S_1 vertical excitation energy. Arrows indicate optimum

Model	$\Delta E(S_1–T_1)$ (↓)	f_{12} (↑)	$+f_{12} - \Delta E(S_1–T_1) - \lvert\Delta E(S_0–S_1) - 3.2\text{eV}\rvert$ (↑)
Dataset	0.020	2.97	−0.04
SMILES-VAE	0.071 ± 0.003	0.50 ± 0.27	−0.57 ± 0.33
SELFIES-VAE	0.016 ± 0.001	0.36 ± 0.31	0.17 ± 0.10
SMILES-LSTM-HC	0.015 ± 0.002	1.00 ± 0.01	−0.24 ± 0.01
SELFIES-LSTM-HC	0.013 ± 0.003	1.00 ± 0.01	−0.24 ± 0.01
REINVENT	0.014 ± 0.003	1.16 ± 0.18	−0.15 ± 0.05
GB-GA	0.012 ± 0.002	2.14 ± 0.45	0.07 ± 0.03
JANUS (RDKit)	0.008 ± 0.001	2.07 ± 0.16	0.02 ± 0.05
JANUS (TYCHE)	**0.007 ± 0.001**	**2.21 ± 0.12**	**0.19 ± 0.03**

stereochemistry randomization makes TYCHE complete in terms of principal molecular string randomization mechanisms.

It is important to understand that the contributions of the individual randomization mechanisms are expected to combine approximately in a multiplicative manner. Each mechanism operates independently, and the number of alternative SMILES strings generated by a given mechanism is expected to be roughly the same, regardless of the underlying spanning tree. As a result, the total number of distinct SMILES representations grows approximately as the product of the contributions from each individual randomization mechanism.

Furthermore, we were pleased to see that TYCHE randomization is not dramatically slower compared to RDKit. Our benchmarks on the DTP_{SUB} dataset (*cf.* Fig. 5) suggest a relative timing of 1.78 compared to RDKit in the asymptotic limit, with a larger performance overhead of TYCHE observed for smaller molecules. Runtime follows roughly a linear scaling for both TYCHE and RDKit (*cf.* Fig. 5A), allowing for randomization to be readily applied to large structures. This systematic difference in timing is quite small when considering that TYCHE implements additional randomization mechanisms compared to RDKit, the most important being spanning tree randomization. There is a significant runtime increase of TYCHE relative to RDKit when going from acyclic molecules to molecules with 1 ring (*cf.* Fig. 5B) which is caused by spanning tree randomization. For any additional ring, TYCHE becomes increasingly efficient compared to RDKit, which suggests TYCHE to scale favorably with the number of rings. This is confirmed by TYCHE being actually faster than RDKit for the large polycyclic aromatic hydrocarbon with 43 annulated rings (*cf.* Table 2). Furthermore, as the core routines of RDKit are written in C++, one might expect a substantial performance advantage over TYCHE, which is written in pure Python.[56] Given these considerations, we believe that TYCHE is sufficiently competitive in terms of speed compared to RDKit for practical applications. While some code performance optimization was carried out for TYCHE, we believe that significant further improvements are feasible.

Finally, rather than covering the complete diversity of molecular string representations, recent efforts in cheminformatics have targeted the reduction of SMILES redundancy. TokenSMILES is a standardization framework that interprets SMILES strings as structured sentences and are composed of context-free words.[57] By introducing several syntactic constraints, TokenSMILES reduces redundant SMILES string representations substantially and was shown to be particularly useful for exhaustive chemical space enumeration.[57] Reducing the redundancy of molecular representations has the potential to accelerate the traversal of chemical space compared to current approaches based on SMILES strings. However, TokenSMILES does not address the spanning tree randomization, Kekulé structure randomization, and stereolabel randomization mechanisms leading to increased SMILES diversity that are explicitly accounted for with TYCHE. In other words, while TokenSMILES reduces "unstructured" redundancy that is not chemically informative and, thus, makes exhaustive traversals more tractable, TYCHE serves a complementary purpose of producing controlled diversity from well-understood mechanisms, which can then be used for principled augmentation. Hence, our work shows the inherent limitations of the current version of TokenSMILES and provides a roadmap for future research directions. Until a complete framework minimizing the redundancy of SMILES strings in all principal aspects exists, tools like TYCHE ensure that the full diversity of molecular string representations such as SMILES and SELFIES can be leveraged for molecular ML and artificial molecular design.

4.2 Applications

We have opted for a diverse set of case studies to demonstrate the utility of TYCHE as a complete algorithm for molecular string randomization. TYCHE can provide meaningful advantages in two overarching application domains, namely data augmentation for training ML models[17,58] and SELFIES-based molecular structure exploration.[18,39] The former is critical for models to learn comprehending molecular strings in their entirety. When string randomization is not used for model training, the resulting models will perceive synonymous molecular strings differently, which introduces systematic errors in the corresponding learned embedding space.[58] As demonstrated by our training data augmentation case study, even chemical foundation models trained on large datasets of SMILES strings such as ChemBERTa[34] do not recognize synonymous molecular strings correctly. Accordingly, we believe that TYCHE has significant potential for impacting the development of future molecular foundation models[59] by providing less biased augmented training datasets. This is because the number of molecules where TYCHE will provide an advantage over RDKit in common molecular ML datasets is substantial. Across the ZINC-250k, ESOL, FreeSolv, LIPO, CEP_{SUB}, DTP_{SUB}, $GDB\text{-}13_{SUB}$, and SNB-60K datasets used in this work, which cover the most important application domains of ML in chemistry, only 1% of all structures have no ring, 8% have 1 ring, and 91% have more than 1 ring (details in the SI Material). This suggests that TYCHE will provide more augmented structures than RDKit for the vast majority of molecules in commonly used datasets.

SELFIES-based molecular structure exploration shows substantial performance improvements in terms of chemical space exploration as assessed by the impact of TYCHE on the performance of the STONED algorithm (*cf.* Tables 3–5). We observe that both the exploration of local chemical structure neighborhoods through SELFIES mutation and the interpolation between multiple SELFIES uncovers structures with significantly improved similarities to the starting structures when TYCHE randomization is leveraged. This performance improvement also transfers to inverse molecular design algorithms that utilize SELFIES modification for candidate generation such as JANUS (*cf.* Tables 6 and 7).[39] We rationalize these performance gains by incomplete randomization effectively making a significant portion of the chemical space inaccessible through partial SELFIES modification. This is because two molecular graphs of high similarity might not necessarily have a high SELFIES string similarity as measured by the Levenshtein

distance or edit distance.[60] The edit distance between two SELFIES strings depends strongly on the specific choice of the underlying spanning trees. While the extent of string similarity is defined by the edit distance, the direction of similarity paths is determined by the spanning tree. Consequently, when the spanning tree of one structure is incompatible with small edit distances to the other structure, a limited number of point mutations cannot interconvert these two molecules. String randomization *via* TYCHE allows for alternative spanning trees to be explored, some of which allow for smaller edit distances between the corresponding SELFIES, making the corresponding interconversion feasible.

We note that the differences in some individual TARTARUS benchmark task scores are quite small but still statistically meaningful. This is a characteristic feature of the TARTARUS benchmark suite as the total number of molecule evaluations is strictly limited.[23] The ability of TYCHE to improve performance meaningfully, even under these strict evaluation constraints, demonstrates its potential impact in practical scenarios. Under such practical conditions, any meaningful performance gains that are essentially free compared to typically expensive property evaluations are critical to increase the success rate of inverse molecular design campaigns. Accordingly, we believe that these performance improvements of TYCHE are sufficient to justify replacing RDKit as the randomization algorithm of choice.

5 Conclusions

In this work, we have developed TYCHE, an algorithm for SMILES string randomization that accounts for all the corresponding principal randomization mechanisms. Unlike established algorithms for SMILES randomization such as the RDKit implementation, TYCHE is capable of randomizing spanning trees, Kekulé structures, and double bond stereochemistry labels. Consequently, the number of synonymous strings generated by TYCHE for molecules with cycles is at least 1 order of magnitude higher. This increased coverage of synonymous SMILES strings impacts both the performance of ML models trained with the help of data augmentation *via* string randomization and the capability of SELFIES-based molecule generation algorithms to explore the chemical space. We have implemented TYCHE as a lightweight and efficient Python library that operates on top of the `selfies` package. Accordingly, we believe that TYCHE will find widespread application in the cheminformatics and molecular ML communities.

Author contributions

AkshatKumar Nigam: conceptualization (supporting); data curation (equal); formal analysis (equal); funding acquisition (equal); investigation (equal); methodology (supporting); resources (supporting); software (supporting); validation (supporting); visualization (equal); writing – original draft (supporting); writing – review and editing (supporting). Serhii Tretiakov: conceptualization (supporting); data curation (supporting); formal analysis (supporting); methodology (supporting); software (supporting); validation (supporting); visualization (supporting); writing – review and editing (supporting). Robert Pollice: conceptualization (lead); data curation (equal); formal analysis (equal); funding acquisition (equal); investigation (equal); methodology (lead); project administration; resources (lead); software (lead); supervision; validation (lead); visualization (equal); writing – original draft (lead); writing – review and editing (lead).

Conflicts of interest

There are no conflicts to declare.

Data availability

The code used for performing exhaustive molecular string randomization and molecular structure exploration is available at our GitLab Repository. Alternatively, the code can also be downloaded at https://doi.org/10.5281/zenodo.21001466.

Supplementary information (SI): additional methodological details and complementary results and discussions. See DOI: https://doi.org/10.1039/d6dd00190d.

Acknowledgements

S.T. and R.P. acknowledge funding through the project NGF.1609.23.018 by the Dutch Research Council (NWO) and the AiNed Foundation. MarvinSketch (version 25.3.5), a product of Chemaxon (https://www.chemaxon.com), was used to prepare chemical structure schemes for this publication. We thank the Center for Information Technology of the University of Groningen for their support and for providing access to the Hábrók high-performance computing cluster.

Notes and references

1 D. Bonchev, *Chemical Graph Theory: Introduction and Fundamentals*, Taylor & Francis, London, 1st edn, 1991.
2 D. Weininger, *J. Chem. Inf. Comput. Sci.*, 1988, **28**, 31–36.
3 D. Weininger, A. Weininger and J. L. Weininger, *J. Chem. Inf. Comput. Sci.*, 1989, **29**, 97–101.
4 S. Heller, A. McNaught, S. Stein, D. Tchekhovskoi and I. Pletnev, *J. Cheminf.*, 2013, **5**, 7.
5 S. R. Heller, A. McNaught, I. Pletnev, S. Stein and D. Tchekhovskoi, *J. Cheminf.*, 2015, **7**, 23.
6 M. Krenn, F. Häse, A. Nigam, P. Friederich and A. Aspuru-Guzik, *Mach. Learn.: Sci. Technol.*, 2020, **1**, 045024.
7 A. Lo, R. Pollice, A. Nigam, A. D. White, M. Krenn and A. Aspuru-Guzik, *Digital Discovery*, 2023, **2**, 897–908.
8 K. T. Butler, D. W. Davies, H. Cartwright, O. Isayev and A. Walsh, *Nature*, 2018, **559**, 547–555.
9 B. Sanchez-Lengeling and A. Aspuru-Guzik, *Science*, 2018, **361**, 360–365.
10 D. S. Wigh, J. M. Goodman and A. A. Lapkin, *Wiley Interdiscip. Rev.:Comput. Mol. Sci.*, 2022, **12**, e1603.
11 M. H. S. Segler, T. Kogej, C. Tyrchan and M. P. Waller, *ACS Cent. Sci.*, 2018, **4**, 120–131.
12 F. Grisoni, M. Moret, R. Lingwood and G. Schneider, *J. Chem. Inf. Model.*, 2020, **60**, 1175–1183.

13 P. Schwaller, T. Laino, T. Gaudin, P. Bolgar, C. A. Hunter, C. Bekas and A. A. Lee, *ACS Cent. Sci.*, 2019, **5**, 1572–1583.
14 R. Irwin, S. Dimitriadis, J. He and E. J. Bjerrum, *Mach. Learn.: Sci. Technol.*, 2022, **3**, 015022.
15 N. M. O'Boyle, *J. Cheminf.*, 2012, **4**, 22.
16 M. Krenn, Q. Ai, S. Barthel, N. Carson, A. Frei, N. C. Frey, P. Friederich, T. Gaudin, A. A. Gayle, K. M. Jablonka, *et al.*, *Patterns*, 2022, **3**, 100588.
17 J. Arús-Pous, S. V. Johansson, O. Prykhodko, E. J. Bjerrum, C. Tyrchan, J.-L. Reymond, H. Chen and O. Engkvist, *J. Cheminf.*, 2019, **11**, 71.
18 A. Nigam, R. Pollice, M. Krenn, G. d. P. Gomes and A. Aspuru-Guzik, *Chem. Sci.*, 2021, **12**, 7079–7090.
19 RDKit, Open-source cheminformatics, https://www.rdkit.org/.
20 M. Krenn, F. Häse, A. Nigam, P. Friederich and A. Aspuru-Guzik, *SELFIES*, 2019, https://github.com/aspuru-guzik-group/selfies.
21 R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams and A. Aspuru-Guzik, *ACS Cent. Sci.*, 2018, **4**, 268–276.
22 J. J. Irwin, T. Sterling, M. M. Mysinger, E. S. Bolstad and R. G. Coleman, *J. Chem. Inf. Model.*, 2012, **52**, 1757–1768.
23 A. Nigam, R. Pollice, G. Tom, K. Jorner, J. Willes, L. Thiede, A. Kundaje and A. Aspuru-Guzik, *Adv. Neural Inf. Process. Syst.*, 2023, 3263–3306.
24 J. Hachmann, R. Olivares-Amaya, A. Jinich, A. L. Appleton, M. A. Blood-Forsythe, L. R. Seress, C. Román-Salgado, K. Trepte, S. Atahan-Evrenk, S. Er, S. Shrestha, R. Mondal, A. Sokolov, Z. Bao and A. Aspuru-Guzik, *Energy Environ. Sci.*, 2014, **7**, 698–704.
25 L. C. Blum and J.-L. Reymond, *J. Am. Chem. Soc.*, 2009, **131**, 8732–8733.
26 G. W. A. Milne, M. C. Nicklaus, J. S. Driscoll, S. Wang and D. Zaharevitz, *J. Chem. Inf. Comput. Sci.*, 1994, **34**, 1219–1224.
27 J. H. Voigt, B. Bienfait, S. Wang and M. C. Nicklaus, *J. Chem. Inf. Comput. Sci.*, 2001, **41**, 702–712.
28 W.-D. Ihlenfeldt, J. H. Voigt, B. Bienfait, F. Oellien and M. C. Nicklaus, *J. Chem. Inf. Comput. Sci.*, 2002, **42**, 46–57.
29 M. Monga and E. A. Sausville, *Leukemia*, 2002, **16**, 520–526.
30 Z. Wu, B. Ramsundar, E. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing and V. Pande, *Chem. Sci.*, 2018, **9**, 513–530.
31 D. L. Mobley and J. P. Guthrie, *J. Comput.-Aided Mol. Des.*, 2014, **28**, 711–720.
32 J. S. Delaney, *J. Chem. Inf. Comput. Sci.*, 2004, **44**, 1000–1005.
33 B. Ramsundar, P. Eastman, P. Walters, V. Pande, K. Leswing and Z. Wu, *Deep Learning for the Life Sciences*, O'Reilly Media, 2019.
34 W. Ahmad, E. Simon, S. Chithrananda, G. Grand and B. Ramsundar, *arXiv*, 2022, preprint, arxiv:2209.01712, DOI: 10.48550/arXiv.2209.01712.
35 D. Rogers and M. Hahn, *J. Chem. Inf. Model.*, 2010, **50**, 742–754.
36 N. Brown, M. Fiscato, M. H. Segler and A. C. Vaucher, *J. Chem. Inf. Model.*, 2019, **59**, 1096–1108.
37 J. Leguy, T. Cauchy, M. Glavatskikh, B. Duval and B. Da Mota, *J. Cheminf.*, 2020, **12**, 1–19.
38 S. Ahn, J. Kim, H. Lee and J. Shin, *Adv. Neural Inf. Process. Syst.*, 2020, 12008–12021.
39 A. Nigam, R. Pollice and A. Aspuru-Guzik, *Digital Discovery*, 2022, **1**, 390–404.
40 X. Hu, G. Liu, Y. Zhao and H. Zhang, *Adv. Neural Inf. Process. Syst.*, 2023, 7405–7418.
41 F. Dötz, J. D. Brand, S. Ito, L. Gherghel and K. Müllen, *J. Am. Chem. Soc.*, 2000, **122**, 7707–7717.
42 Y. Matsuo, A. Nakagawa, S. Seki and T. Tanaka, *Beilstein J. Org. Chem.*, 2025, **21**, 1561–1567.
43 J. E. Hertzog, V. J. Maddi, L. F. Hart, B. W. Rawe, P. M. Rauscher, K. M. Herbert, E. P. Bruckner, J. J. de Pablo and S. J. Rowan, *Chem. Sci.*, 2022, **13**, 5333–5344.
44 M. Sasaki, N. Matsumori, T. Maruyama, T. Nonomura, M. Murata, K. Tachibana and T. Yasumoto, *Angew. Chem., Int. Ed.*, 1996, **35**, 1672–1675.
45 T. Nonomura, M. Sasaki, N. Matsumori, M. Murata, K. Tachibana and T. Yasumoto, *Angew. Chem., Int. Ed.*, 1996, **35**, 1675–1678.
46 R. E. Moore, G. Bartolini, J. Barchi, A. A. Bothner-By, J. Dadok and J. Ford, *J. Am. Chem. Soc.*, 1982, **104**, 3776–3779.
47 A. Daugan, P. Grondin, C. Ruault, A.-C. Le Monnier de Gouville, H. Coste, J. Kirilovsky, F. Hyafil and R. Labaudinière, *J. Med. Chem.*, 2003, **46**, 4525–4532.
48 H. A. Ghofrani, I. H. Osterloh and F. Grimminger, *Nat. Rev. Drug Discovery*, 2006, **5**, 689–702.
49 C. Wang, Y. Jiang, J. Ma, H. Wu, D. Wacker, V. Katritch, G. W. Han, W. Liu, X.-P. Huang, E. Vardy, J. D. McCorvy, X. Gao, X. E. Zhou, K. Melcher, C. Zhang, F. Bai, H. Yang, L. Yang, H. Jiang, B. L. Roth, V. Cherezov, R. C. Stevens and H. E. Xu, *Science*, 2013, **340**, 610–614.
50 A. Wang, U. Savas, M.-H. Hsu, C. D. Stout and E. F. Johnson, *J. Biol. Chem.*, 2012, **287**, 10834–10843.
51 M. Olivecrona, T. Blaschke, O. Engkvist and H. Chen, *J. Cheminf.*, 2017, **9**, 48.
52 J. H. Jensen, *Chem. Sci.*, 2019, **10**, 3567–3572.
53 P. Ertl and A. Schuffenhauer, *J. Cheminf.*, 2009, **1**, 8.
54 D. B. Wilson, *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, New York, NY, USA, 1996, pp. 296–303.
55 J. G. Siek, L.-Q. Lee and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual, The*, Pearson Education, 2001.
56 R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes and J. Saraiva, *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2017*, Vancouver, BC, Canada, 2017, pp. 256–267.
57 L. A. Gonzalez-Ortiz, L. Noriega, F. Ortiz-Chi, G. Vidales-Ayala, E. Soberanis-Cáceres, A. Meneses-Viveros, A. Aspuru-Guzik and G. Merino, *Chem. Sci.*, 2026, **17**, 1666–1675.
58 T. B. Kimber, M. Gagnebin and A. Volkamer, *Artif. Intell. Life Sci.*, 2021, **1**, 100014.
59 J. Choi, G. Nam, J. Choi and Y. Jung, *JACS Au*, 2025, **5**, 1499–1518.
60 V. I. Levenshtein, *Sov. Phys. Dokl.*, 1966, 707–710.