A Appendix

A.1 Example

As an example, we will go through encoding and decoding the molecule celecoxib.

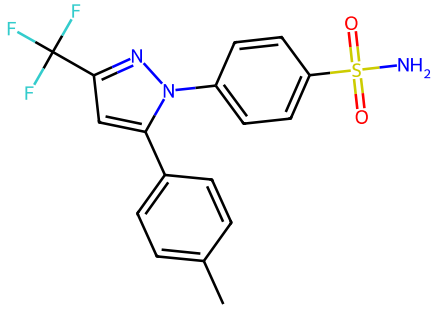


Figure 8: Celecoxib

We will define our group set as consisting of 4 main groups: trifluoromethane, toluene, benzene, and sulfonamide.

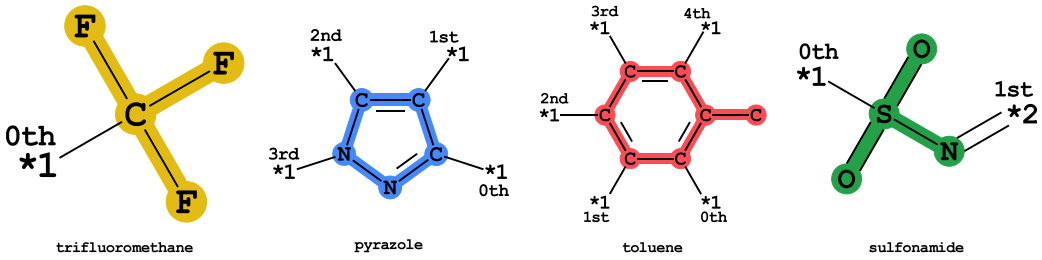
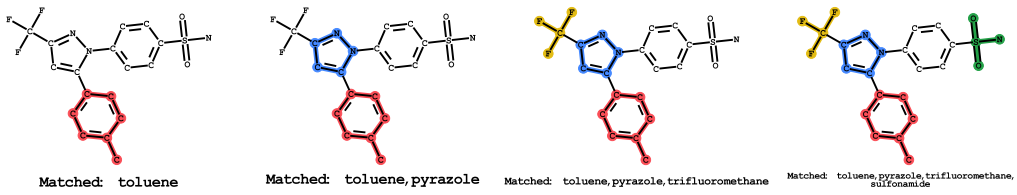
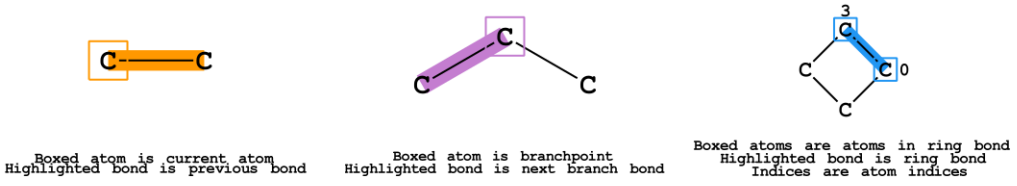


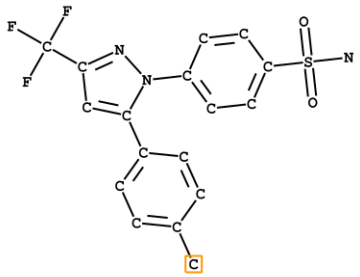
Figure 9: Group set used in this example.

We will now extract occurrences of the groups in the molecule. Since priorities were not specified, we match groups by size in descending order.

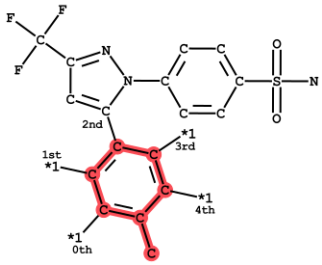


We will now traverse the graph. The following diagrams demonstrate this process.



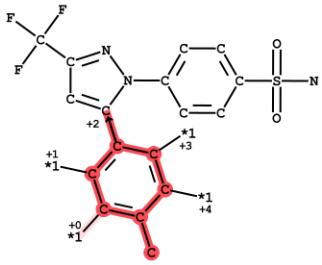


We then start traversing at an arbitrary atom, in this case the methyl carbon of the toluene.



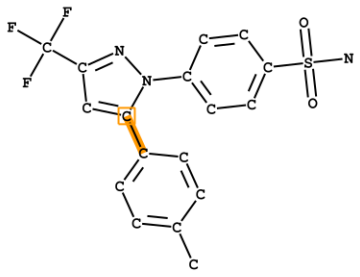
`[:0toluene]`

We find that this atom belongs to a group, and so place the appropriate group token. We arbitrarily select 0 to be the starting attachment point, since there is no previous connection into this group.

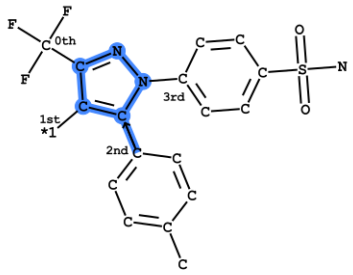


`[:0toluene] + [+2]`

We then place an index token of `[+2]` to go from the `0th` attachment point to the `2nd` attachment point. For readability, index tokens are shown as numbers, but in the actual encoder output a token with the appropriate overload value would be used (e.g. in this case `[Ring2]` is overloaded to `[+2]`).

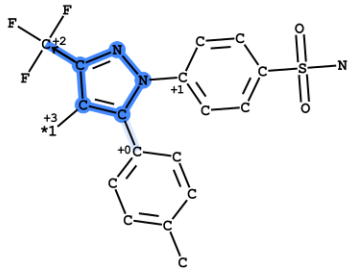


`[:0toluene][+2]`

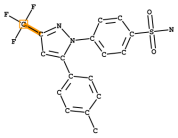


[:0toluene][+2] + [:2pyrazole]

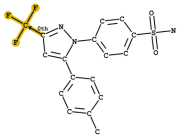
We again find that our current atom is in a group. This time, we place the group token with the starting attachment point at the 2nd attachment index, since that is where the last bond entered from.



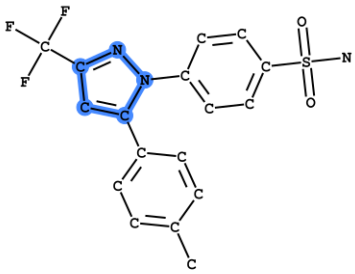
[:0toluene][+2][:2pyrazole] + [+2]



[:0toluene][+2][:2pyrazole][+2]

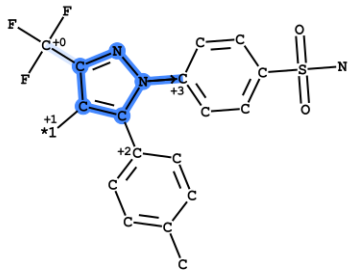


[:0toluene][+2][:2pyrazole][+2] + [:0trifluoromethane]



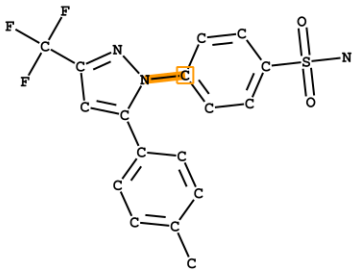
[:0toluene][+2][:2pyrazole][+2][:0trifluoromethane] + [pop]

We then use the [pop] token to exit the current group. In this case, we go from the trifluoromethane group back to the pyrazole group.

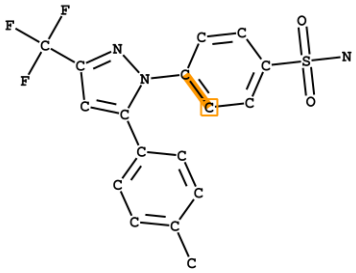


`[:0toluene][+2][:2pyrazole][+2][:0trifluoromethane][pop] + [+3]`

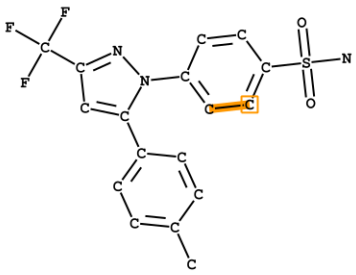
The next few steps build out the carbon chain of the benzene using atomic tokens, including placing a branch and a ring bond.



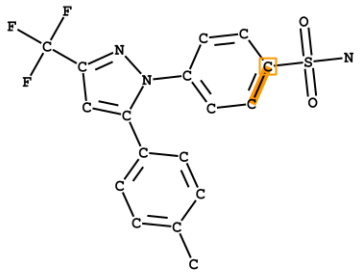
`[:0toluene][+2][:2pyrazole][+2][:0trifluoromethane][pop][+3] + [C]`



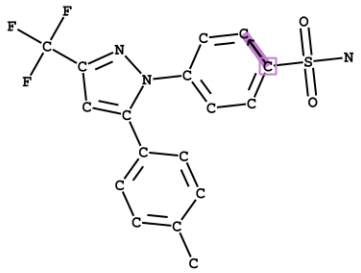
`[:0toluene][+2][:2pyrazole][+2][:0trifluoromethane][pop][+3][C] + [=C]`



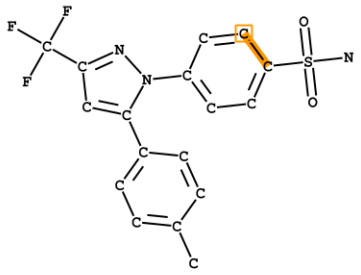
`[:0toluene][+2][:2pyrazole][+2][:0trifluoromethane][pop][+3][C][=C] + [C]`



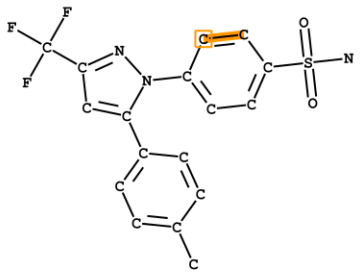
[:0toluene][+2][:2pyrazole][+2][:0trifluoromethane][pop][+3][C][=C][C] + [=C]



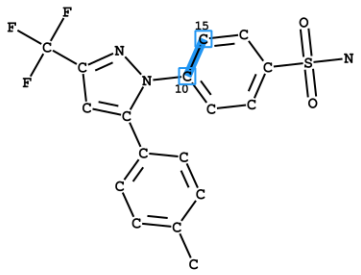
[:0toluene][+2][:2pyrazole][+2][:0trifluoromethane][pop][+3][C][=C][C][=C] + [Branch]



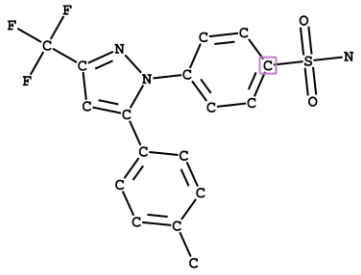
[:0toluene][+2][:2pyrazole][+2][:0trifluoromethane][pop][+3][C][=C][C][=C][Branch] + [C]



[:0toluene][+2][:2pyrazole][+2][:0trifluoromethane][pop][+3][C][=C][C][=C][Branch][C] + [=C]

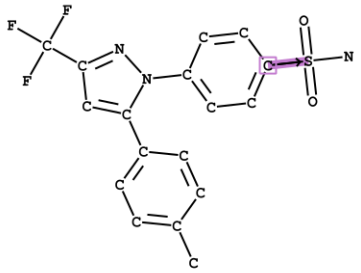


[:0toluene][+2][:2pyrazole][+2][:0trifluoromethane][pop][+3][C][=C][C][=C][Branch][C][=C] + [Ring1][+4]

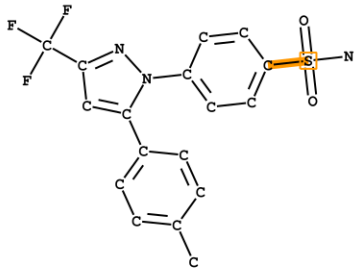


[:0toluene][+2][:2pyrazole][+2][:0trifluoromethane][pop][+3][C][=C][C][=C][Branch][C][=C][Ring1][+4] + [pop]

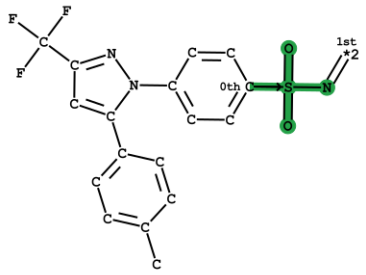
After the benzene is completed, the molecule is finished by placing a sulfonamide group.



[:0toluene][+2][:2pyrazole][+2][:0trifluoromethane][pop][+3][C][=C][C][=C][Branch][C][=C][Ring1][+4][pop]



[:0toluene][+2][:2pyrazole][+2][:0trifluoromethane][pop][+3][C][=C][C][=C][Branch][C][=C][Ring1][+4][pop]



The decoding process for this Group SELFIES string will read in token-by-token, building the molecular graph in the same order as shown in this encoding example.

A.2 Complexity and compression

To take into account the alphabet size of SMILES, SELFIES, and Group SELFIES, we compress each representation using `zlib` to estimate the complexity of each representation. We first do index encoding to convert each string into a list of ints by replacing each unique token with a unique int. We then compress this representation using `zlib`.

Representation	# unique tokens	strings	+index encoding	+index encoding +`zlib` compression
SMILES	34	**11.80 MB**	23.11 MB	3.70 MB
SELFIES	107	46.44 MB	19.68 MB	3.97 MB
Group SELFIES	247	42.05 MB	**15.90 MB**	**3.61 MB**

Table 3: Filesize of ZINC-250k when represented in SMILES, SELFIES, and Group SELFIES. When using strings, SMILES takes up the least space. When using index encoding and then compressing using `zlib`, Group SELFIES takes up the least space.

A.3 NFA experiments

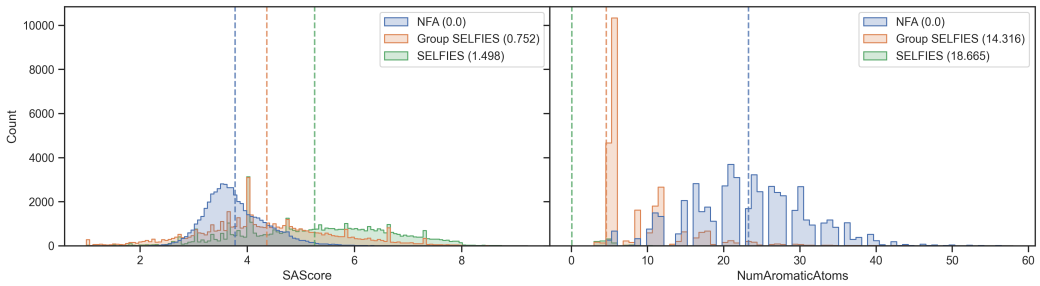


Figure 10: Generated Group SELFIES and regular SELFIES binned by SAScore and number of aromatic atoms. Molecules were generated from the NFA dataset.

We repeat Experiment 4.2 but for a dataset of nonfullerene acceptors (NFA) [50]. This dataset contains many conjugated aromatic systems. In this case, we histogram by SAScore and the number of aromatic atoms, as shown in Figure 10. Generated SELFIES are rarely able to preserve the aromatic systems found in NFA, but generated Group SELFIES preserve much more of the aromatic systems, and can produce molecules with lower and higher SAScore. When binning by number of aromatic atoms, molecules with 0 aromatic atoms are omitted for clarity. About 49% / 98% of molecules generated via Group SELFIES / SELFIES have 0 aromatic atoms. Group SELFIES uses a group set

containing 74 groups obtained via naïve fragmentation. $N = 51281$ molecules are generated via Group SELFIES / SELFIES, equal to the size of the NFA dataset.

A.4 Substring SELFIES

Substring SELFIES is generated by taking a Group SELFIES string and replacing every group token with a SELFIES substring corresponding to it. We compare Substring SELFIES and Group SELFIES in the context of Experiment 4.2 in Figure 11 and find that generated Substring SELFIES are on par with generated Group SELFIES.

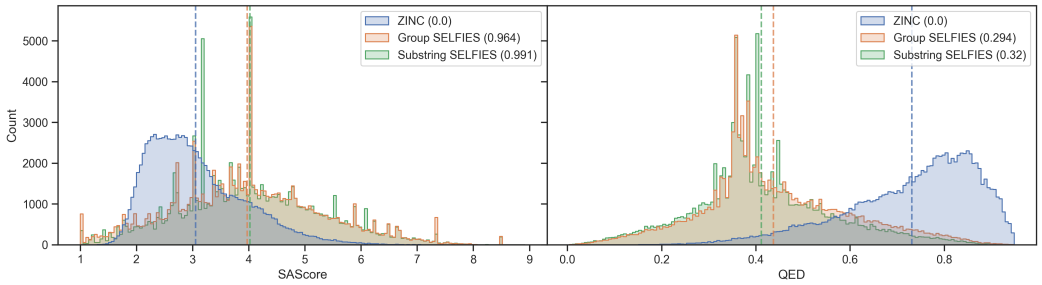


Figure 11: Generated Group SELFIES and Substring SELFIES binned by SAScore and QED. Molecules were generated from ZINC-250k.

A.5 No-Group SELFIES

No-Group SELFIES is generated by using the Group SELFIES encoder with an empty group set. We compare regular SELFIES to No-Group SELFIES to determine the effect of small changes in the base encoding process, such as branch tokens. In SELFIES, all `[BranchX]` expect a number token after to determine the length of the branch, whereas in Group SELFIES, all `[Branch]` tokens do not need a number token, instead using a `[pop]` token to exit branches. In Figure 12, generated No-Group SELFIES are on par with or worse than generated SELFIES, indicating that the modified encoding process does not contribute much to the overall performance of Group SELFIES.

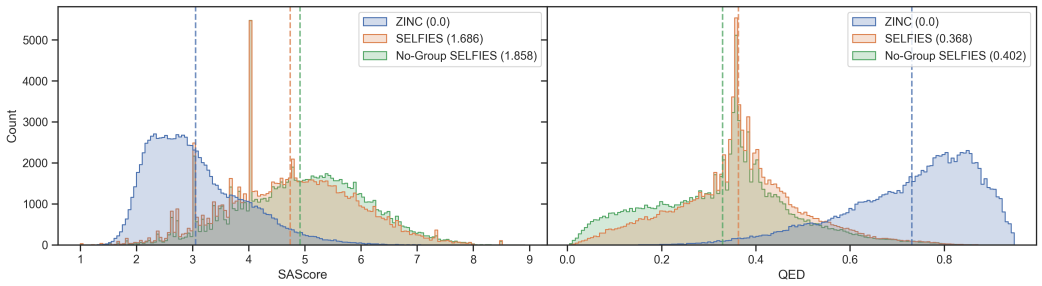


Figure 12: Generated regular SELFIES and No-Group SELFIES binned by SAScore and QED. Molecules were generated from ZINC-250k.

A.6 Distribution learning on other metrics

Distribution plots of molecular weight, QED, SAScore, and logP are shown to compare the distributions learned by the SELFIES and Group SELFIES VAE.

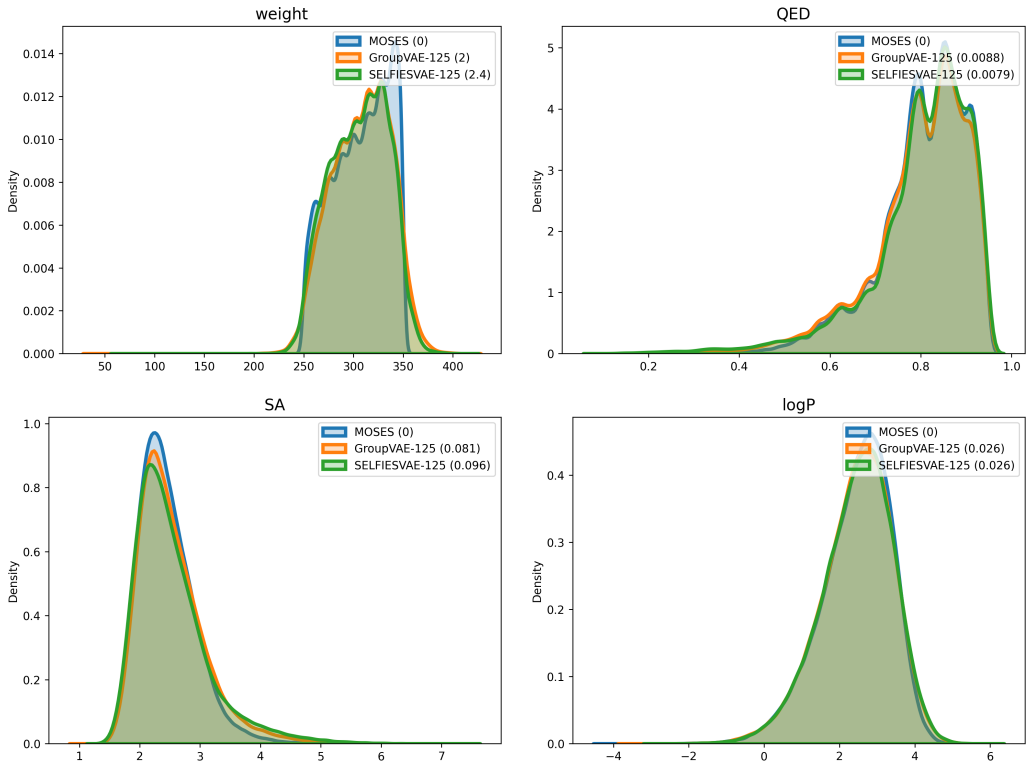


Figure 13: Distributions of molecular weight, QED, SAScore, and logP. Bracketed values in the legend represent the Wasserstein distance to the original MOSES distribution.

A.7 Timing

Representation	Encode	Decode
SELFIES	0.199 ms	0.133 ms
Group SELFIES	12.860 ms	2.494 ms

Table 4: Running time of encoder/decoder for SELFIES and Group SELFIES. The SELFIES encoder is 65x faster than the Group SELFIES encoder, and the SELFIES decoder is 19x faster than the Group SELFIES decoder. Timing is averaged per molecule over the ZINC-250k dataset, using the same group set of 53 groups as the length histogram experiments. Computations were done on a 2021 Apple MacBook Pro with a M1 Pro chip and 32 GB RAM.