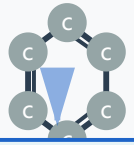
SMILES Notation

Ethanol

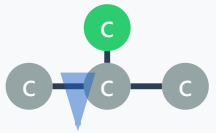
CCO
SMILES String

Benzene



c1ccccc1
Ring Closure

Branched



CC(C)C
Branching ()

Syntax Rules

String-based molecular encoding

Canonical SMILES

Unique molecular representation

SMARTS Patterns

Substructure search patterns

Tokenization

Breaking into meaningful units

Augmentation Strategies

Data augmentation techniques

Syntax Rules

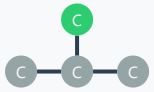
Atoms

C	N	O	[Cl]	[NH3+]
Carbon	Nitrogen	Oxygen	Chlorine	Ion

Bonds

CC	C=C	C#C	c1ccccc1
Single (-)	Double (=)	Triple (#)	Aromatic

Branches



`CC(C)C`

Ring Closures

`C1CC1`

`c1ccccc1`

Basic Rules

SMILES uses a linear text format to represent molecular structures. Atoms are represented by their element symbols, and bonds are either implicit (single bonds) or explicit using special characters.

Atom Representation

Organic atoms (C, N, O, S, P) can be written without brackets. Other atoms and charged species must be enclosed in square brackets.

```
CCO → Ethanol
[NH4+] → Ammonium ion
```

Bond Types

Single bonds are implicit. Double (=), triple (#), and aromatic bonds are explicitly denoted. Aromatic atoms use lowercase letters.

```
C=C → Ethene
C#N → Acetonitrile
```

Key Point: SMILES follows a depth-first tree traversal to encode molecular connectivity.

Canonical SMILES

Non-Canonical (Multiple Valid Forms)

CCCC
C(CC)C
C(C)CC

All represent the same molecule!

Canonical SMILES (Unique)

CCCC

Generated by standardized algorithm

Canonicalization Steps

1. Compute atom invariants
2. Rank atoms by symmetry
3. Generate canonical traversal
4. Output unique SMILES

What is Canonicalization?

A single molecule can be represented by many different valid SMILES strings. Canonical SMILES ensures that each unique molecule has exactly one standardized representation.

Why is it Important?

Canonical SMILES enables reliable molecule comparison, database searching, and duplicate detection. Without canonicalization, the same molecule might not be recognized as identical.

Generation Algorithm

The algorithm computes unique atom invariants based on connectivity, then systematically ranks and orders atoms to produce a consistent traversal path.

```
from rdkit import Chem mol =
Chem.MolFromSmiles('C(C)CC') canonical =
Chem.MolToSmiles(mol) # Output: 'CCCC'
```

Applications: Structure searching, molecular databases, machine learning datasets, quality control in cheminformatics

SMARTS Patterns

SMARTS vs SMILES

SMARTS (SMiles ARbitrary Target Specification) extends SMILES with pattern-matching capabilities. While SMILES describes specific molecules, SMARTS describes molecular patterns for substructure searching.

Substructure Matching Language

Alcohol Pattern

`[CX4][OX2H]`

Matches: sp^3 carbon bonded to OH group

Aromatic Ring

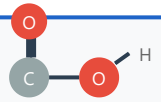
`a1aaaaa1`

Matches: any 6-membered aromatic ring

Carboxylic Acid

`C(=O)[OH]`

Matches: -COOH functional group



Common SMARTS Operators

- [#6] = Carbon atom
- [R] = In ring
- [D3] = 3 connections
- [a] = Aromatic
- [+] = Positive charge
- [!#6] = NOT carbon

Pattern Matching Features

SMARTS supports logical operators (AND, OR, NOT), atom properties (charge, connectivity), and wildcard matching, making it powerful for identifying functional groups and chemical motifs.

```
[CX4] → sp³ carbon (4 connections)
[OX2H] → OH group
[$([NX3])] → Nitrogen with 3 bonds
```

Applications

Drug discovery, toxicity prediction, reaction site identification, and automated molecular filtering.

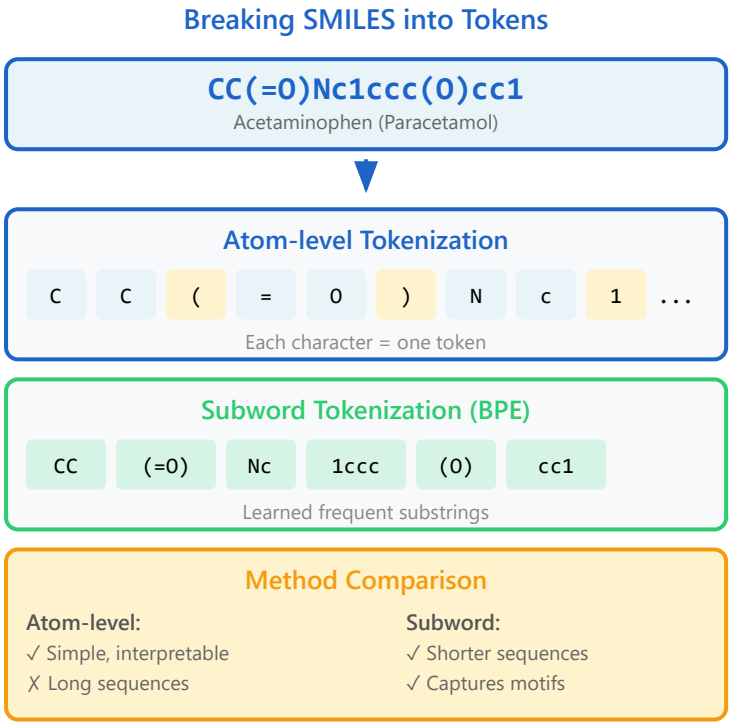
```
from rdkit import Chem pattern =
Chem.MolFromSmarts('[CX4][OX2H]') mol =
Chem.MolFromSmiles('CCO') matches =
mol.GetSubstructMatches(pattern) # Returns atom
indices of matches
```

Power Tool: SMARTS enables sophisticated molecular queries impossible with simple text search

Tokenization

Why Tokenization Matters

Machine learning models process sequences of discrete tokens, not raw strings. Tokenization determines how SMILES strings are split into meaningful units for neural networks and transformers.



Tokenization Strategies

Character-level: Each atom, bond, and bracket is a token. Simple but creates long sequences.

Subword (BPE/WordPiece): Learns frequent substrings from data. Captures chemical motifs and reduces sequence length.

```
Character: ['C', 'C', '(', '=', 'O', ')']
Subword: ['CC', '(=O)']
```

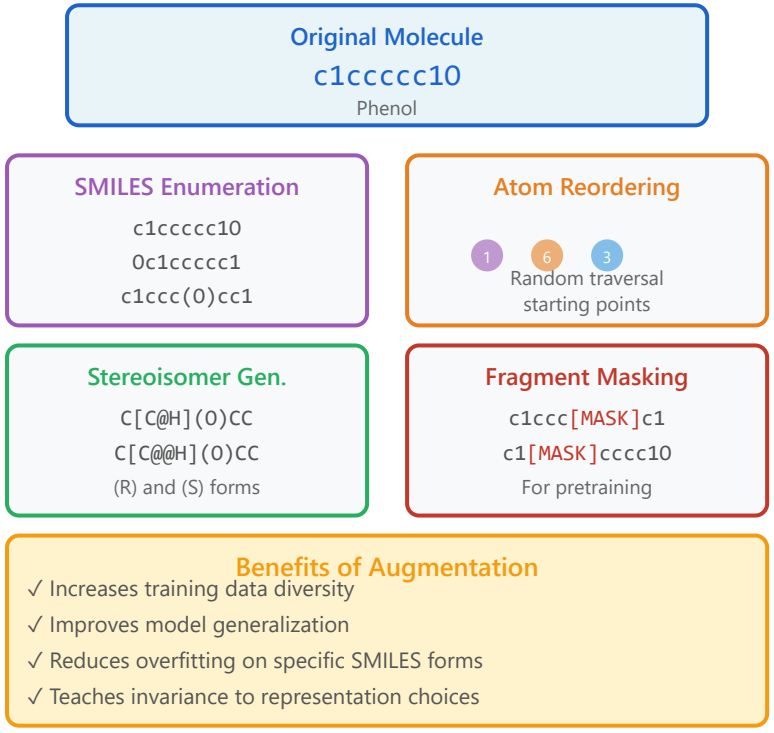
Impact on ML Models

Better tokenization improves model performance, training efficiency, and chemical understanding. Subword methods help models learn functional group patterns naturally.

```
from transformers import AutoTokenizer tokenizer
= AutoTokenizer.from_pretrained(
"seyonec/ChemBERTa-zinc-base-v1" ) tokens =
tokenizer.tokenize("CC(=O)O")
```

Trade-off: Character-level is interpretable; subword is efficient but requires learned vocabulary

Augmentation Strategies



Data Augmentation for SMILES

Since one molecule can have multiple valid SMILES representations, we can generate augmented training data by creating different valid SMILES strings for the same molecule.

Augmentation Techniques

SMILES Enumeration: Generate all valid SMILES by varying atom ordering and ring numbering.

Random Perturbations: Apply small structural or notation changes while preserving molecular identity.

Fragment Masking: Randomly mask portions for self-supervised learning tasks.

Implementation Example

```
from rdkit import Chem mol =
Chem.MolFromSmiles('c1ccccc1O') # Generate 5
random SMILES augmented = [] for i in range(5):
smi = Chem.MolToSmiles(mol, doRandom=True)
augmented.append(smi) # Result: ['Oc1ccccc1',
'c1ccc(O)cc1', ...]
```

Key Insight: Augmentation teaches models to focus on molecular structure rather than specific notation choices, improving robustness