ARL-TN-1137 ● Oct 2022

DEVCOM ARMY RESEARCH LABORATORY

Chemical Kinetics Database Translation for Machine-Learning-based Algorithm Development

by Lydia G Yeh, Jeffrey D Veals, and Christopher P Stone

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TN-1137 ● OCT 2022

Chemical Kinetics Database Translation for Machine-Learning-based Algorithm Development

Lydia G Yeh
Richard Montgomery High School, Rockville, MD

Jeffrey D Veals and Christopher P Stone
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
October 2022	Technical Note	27 June–16 September 2022

4. TITLE AND SUBTITLE
Chemical Kinetics Database Translation for Machine-Learning-based Algorithm Development

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Lydia G Yeh, Jeffrey D Veals, and Christopher P Stone

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DEVCOM Army Research Laboratory
ATTN: FCDD-RLW-WC
Aberdeen Proving Ground, MD 21005

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-1137

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES
ORCID ID: Jeffrey D Veals, 0000-0002-8238-345X

14. ABSTRACT

The efficacy of various chemical descriptor languages has been considered as part of a larger effort to develop a protocol to modernize and standardize entries in a large “in-house” database of chemical species information. This capability will help to improve the turnaround time of the data flow for US Army Combat Capabilities Development Command Army Research Laboratory chemical kinetics mechanism development efforts and make it possible to merge the in-house databases with additional open-source computational models. Several approaches have been considered to convert key information contained within Gaussian quantum chemistry simulation output files into several standardized formats including Simplified Molecular-Input Line-Entry System (SMILES), Canonical SMILES, InChlKeys, and Sybyl MOL2. Additional consideration has been made of the optimal contents and formatting of the full molecular species database moving forward. A brief overview of the initial protocol and progress to date is reported.

15. SUBJECT TERMS
Simplified Molecular-Input Line-Entry System, SMILES, Canonical SMILES, International Chemical Identifier, InChI, InChIKey, gaussian 16, gaussian09, machine learning, chemical descriptors, Weapons Sciences

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Jeffrey D Veals
Unclassified	Unclassified	Unclassified	UU	26	19b. TELEPHONE NUMBER (Include area code) (410) 278-2916

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

Acknowledgments

The authors thank Drs Michael J McQuaid, Chiung-Chu Chen, and Rose Pesce-Rodriguez for their thoughtful feedback and support. Additionally, we thank Weapon Propulsion Science Branch and Weapons Sciences Division management, the ARL summer student program management, and the ARL High School Apprenticeship Program for apprenticeship approval and financial support.

1. Introduction

While new technologies continue to emerge, making US Army research more effective, nonstandard practices could hinder technological advances due to their abstractness, inefficiency, and/or the possibility for human error. One potential inconsistency arises from the naming approach that computational research teams may use to label specific chemical compounds in their modeling studies (Ye et al. 2015; Martin et al. 2019; Long et al. 2019). Solving this problem by creating a universal database to act as a translator and consistent formatter for the team would allow team members to standardize the molecular descriptors used in their current database and potentially leverage the power of machine learning in future research, leading to faster turnaround times for combustion modeling studies (Chen et al. 2019; McQuaid et al. 2019; McQuaid 2020). This will impact the Army by allowing the predictive capabilities of this combustion modeling approach to be fully realized within the life cycle of typical mission programs (i.e., 1–3 years).

As shown in Fig. 1, we have implemented a Python program that takes a set of Gaussian (Frisch et al. 2016) quantum chemistry simulation result files and processes the information into a list of specified output descriptor formats using the open-source software Open Babel (O'Boyle 2011 et al.; Hutchinson et al. 2021; es 2022). The generated molecular descriptor information can be entered into a database or added to an existing database to standardize naming schemes and provide multiple chemical descriptor languages for use in an accessible manner.

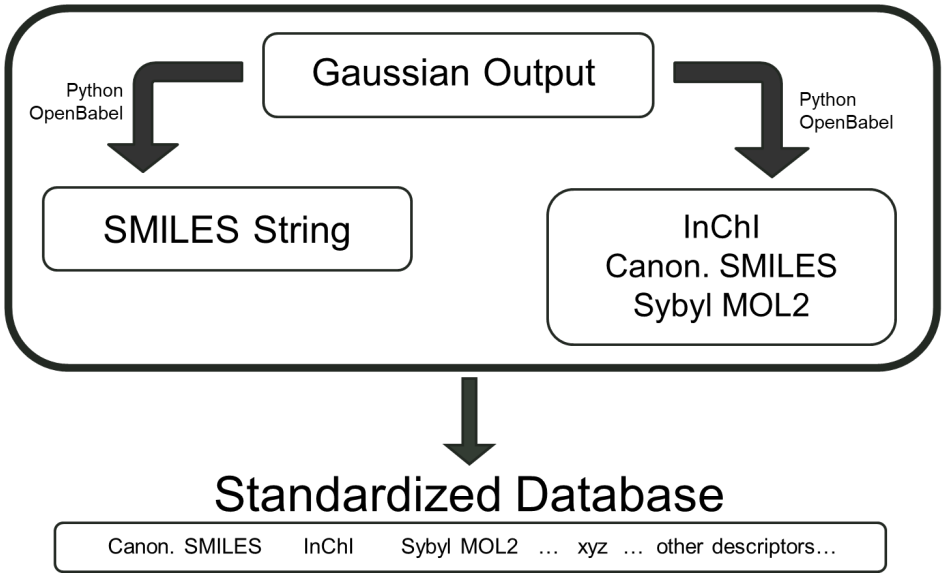

Fig. 1 Flow model of the conversion process from Gaussian output files into a standardized database of important molecular descriptor formats

Current obstacles are outlined in this report, and discussion of the Python code's limitations and ideas for future functionality extensions are presented.

2. Computational Background and Methodology

2.1 Historical Program Limitations and Chemical Language Inconsistencies

Within our research team, a nonstandard naming convention is used to label molecules when creating chemical reaction networks (i.e., mechanisms) used in combustion models. The existing convention is constrained by legacy software tools that did not anticipate or design support for large molecules. The resulting terse labels, commonly constrained to 16 characters, are largely untranslatable or unintelligible by anyone other than their creators. This makes collaboration difficult, with time spent asking the originators for clarification on the connection between the species labels and their chemical name, respective structure, and bonding patterns, instead of directly working with the project at hand

Not only are collaborators unable to readily understand the nonstandard chemical naming scheme, but the computer programs they use may not be able to interpret them either. Many programs only accept and process certain string formats, so these incompatibilities lead to a difficulty to collaborate, and/or inability to leverage potentially useful software.

One could use specific chemical names following standardized rules (e.g., the International Union of Pure and Applied Chemistry [IUPAC]) conventions (Leigh et al. 1998; IUPAC 2022). However, the names may still be too long for compatibility with software and necessary specificity is not guaranteed. For example, the general chemical formula for nitroglycerin, $C_3H_5N_3O_9$, is short enough to fit within common character limits, but hundreds of relevant chemical species could have the same general formula, making the name non-unique.

2.2 Input File Format

For this project, output files previously generated by the popular computational chemistry code Gaussian were used as the inputs for our conversion program. Gaussian output files hold very detailed information about a given molecular species. In addition to the electronic structure of each species, the files contain information on the coordinates, z-matrix, and bond matrices that describe the spatial location of atomic nuclei and the molecular bonding patterns connecting them. This structural information allows the information in the Gaussian output to be converted

into most common molecular descriptor formats. Note that both versions of Gaussian outputs (i.e., Gaussian09 and Gaussian16) we use were successfully translated using the given code.

2.3 Output Format Types

Throughout this investigation, several chemical descriptor languages were considered for inclusion to the database. Primary among these was the Simplified Molecular-Input Line-Entry System (SMILES), a chemically descriptive language that uses letters and symbols to express chemical structures.

SMILES, which was originally developed by Weininger (1988), has many advantages, a primary one being its universality. Many software systems can generate or import SMILES strings. There is also a degree of human readability; a chemist can read or create a SMILES string using their knowledge of the rules and the molecular structure. This can be very beneficial when adding naming labels for chemical species. Additionally, because they are single strings, not a collection of information like atomic coordinates, it is compact and easily stored (Belford 2022).

However, there are drawbacks and shortcomings with SMILES strings. Many applications can generate and use SMILES strings, but they may use different algorithms due to the proprietary nature of SMILES. This leads to different, non-unique SMILES strings that correlate to the same chemical species across different software applications. This means that SMILES strings are not always interchangeable across software applications and can make communication between teams difficult if these teams are not using the same software to generate SMILES strings. Another disadvantage of SMILES is its inability to retain molecular coordinate information

These drawbacks are what led us to consider other languages such as Canonical SMILES (O'Boyle 2012), IUPAC International Chemical Identifier (InChI) (Heller et al. 2015), and the Sybyl MOL2 format (Ash et al. 1997; Homer et al. 2008). Many other descriptor languages exist, but these meet the needs of our team currently. Each one has unique advantages and disadvantages outlined in the following.

Canonical SMILES is of particular interest due to its similarities to SMILES, and it resolves several disadvantages of SMILES. Like SMILES, Canonical SMILES is "universal," human readable, compact, and easily stored. However, being open-source, conversion software applications use the same standardized string-generation process. Every Canonical SMILES string is generated using a specific method, which greatly reduces the chances of the non-unique strings being

generated for the same molecular compound (i.e., uniqueness). For example, O'Boyle (2012) tested two databases, the chEMBL database and a subset of the PubChem Substance Database, with over one million compounds each. Unique identifiers were generated 99.79% and 99.77% of the time, respectively. Although it greatly reduces the uniqueness disadvantage, this language still cannot retain molecular coordinate information. Canonical SMILES can also take longer to generate than regular SMILES; however, a significant difference between the two has not been noticed during our testing of Open Babel and Python.

Differences between Canonical SMILES and SMILES are given in the following figures. While there are some simple molecules that have the same string in both languages, the differences become apparent as the structures get more complicated (e.g., forming rings or branches). Figure 2 also demonstrates the human readability of both descriptors (e.g., highlighted parts in the 2-D structure correspond directly to a few characters in the strings).

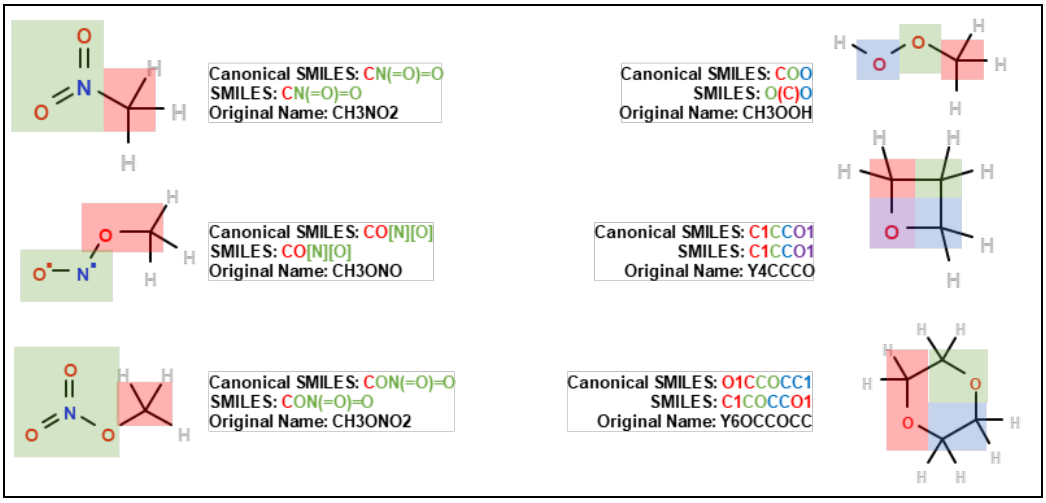

Fig. 2 Original database names, color-coded SMILES and 2-D structures for select molecular species, indicating what string sequence corresponds to specific substructure of each molecular species. The images and SMILES strings were generated with Open Babel version 3.1.0.

Another string-based language that will be employed within our database is InChI. Like Canonical SMILES, InChI is open-source, allowing software applications to leverage the same generation process and improving the uniqueness and universality Canonical of the strings. It provides a deeper level of detail than either SMILES or Canonical SMILES by encoding up to six layers of molecular structure information. The encoded strings are information-dense and fully machine-comprehendible but are more difficult for chemists to decipher, especially when considering more-complex compounds (Belford 2022). Therefore, it may not be a good basis for naming schemes, as it can be important to recognize a chemical

compound based on its name. The lack of human readability can also make it difficult for researchers to understand or manually recreate an InChI string. InChI strings can also become very long, depending on the complexity of the chemical structure, as there is no character limit on each of the six levels of detail.

To ameliorate this, InChI Keys were developed (Southan 2013). InChI Keys encode the long, variable-length InChI strings into fixed-length 27-character strings. This eases use of search engines with character limits such as Google or other software applications, but this compactness removes any human comprehensibility.

Figure 3 demonstrates the differences among InChI, SMILES, and Canonical SMILES. InChI strings are longer than the other formats. With CH_2OJ, for example, both SMILES require four characters, but the InChI uses 17 characters. Although meant to be machine-readable-only, we show how each InChI string corresponds to the simple molecular compounds. Within the InChI string there are bounds of the second and third slash (see Fig. 3)—aspects of the original chemical name can be seen—but as the string is read from left to right, the meaning of the characters become more indecipherable.

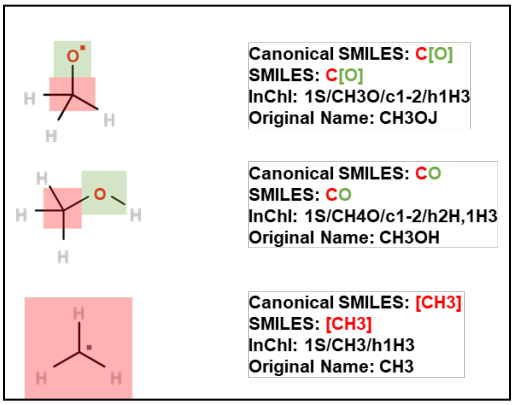

Fig. 3 Original database names, InChI, and color-coded SMILES and 2-D structures for select molecular species indicating what string sequence corresponds to the specific substructure of each molecular species. The images, SMILES and InChI strings, were generated with Open Babel version 3.1.0.

Sybyl MOL2 is another format that is within our database and is the only non-string-based format that has been considered. An example is shown in Fig. 4. One of the major shortcomings of languages like SMILES and Canonical SMILES is their inability to store molecular coordinate and information. The Sybyl Mol2 encoding retains connectivity information as well as additional information about atomic features and position. One of its disadvantages is that it is formatted as a collection of fixed-width tables, meaning it requires more formatting and space than SMILES and InChI strings.

```
@<TRIPOS>MOLECULE
CH3OJ_M06GTBas3_CCSDT_B4.out
 5 4 0 0 0
SMALL
GASTEIGER
****
COMMENT LINE
@<TRIPOS>ATOM
      1 H           0.3184   -0.5176   -1.3163 H       1  UNL1        0.0555
      2 C           0.3167   -0.5149   -0.2046 C.3     1  UNL1        0.0718
      3 H           1.3802   -0.5074    0.0940 H       1  UNL1        0.0555
      4 H          -0.1696   -1.4608    0.0938 H       1  UNL1        0.0555
      5 O          -0.3646    0.5925    0.1544 O.3     1  UNL1       -0.2383
@<TRIPOS>BOND
     1     1     2    1
     2     2     4    1
     3     2     3    1
     4     2     5    1
```

Fig. 4 Example of Sybyl Mol 2 format for the CH_3OJ radical

As shown in Fig. 5, the complexity of Canonical SMILES and standard SMILES increases for larger molecules. However, if a chemist has a sufficient understanding of SMILES syntax and rules, interpretation of molecular conformation and connectivity described in the SMILES format is fairly straightforward.

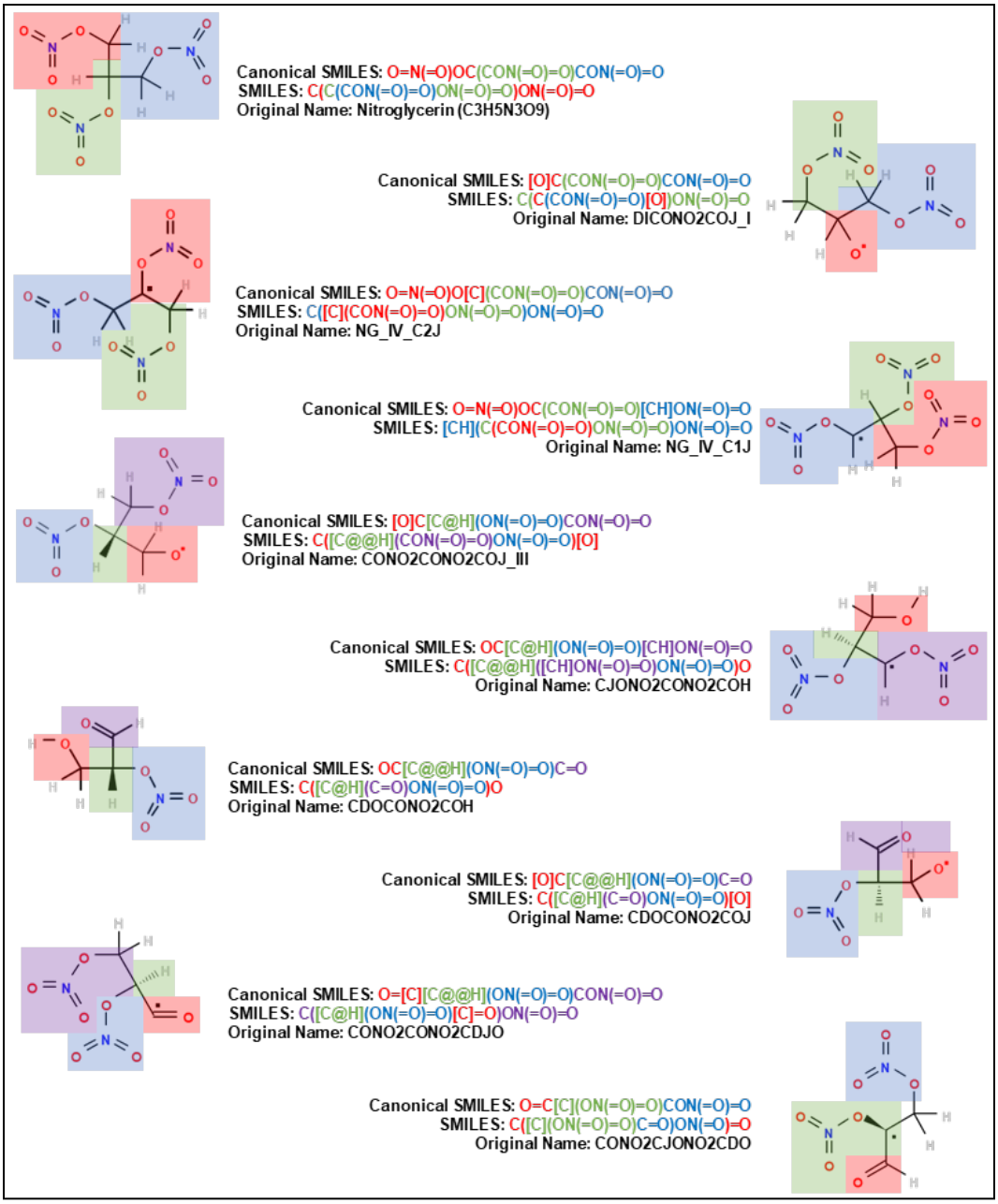

Fig. 5 Original database names, color-coded SMILES and 2-D molecular structures for select molecular species from our nitroglycerin pyrolysis mechanism database. Color-coding indicates what string sequence corresponds to specific substructure of each molecular species. The images and SMILES strings were generated with Open Babel version 3.1.0.

2.4 Code and Current Capabilities

In the current investigation, Python was used to implement two different code segments.

The first code segment (see Appendix A, Fig. A-1) searches for all Gaussian output files located in a specified folder (directory) as input to process and translate into the given output descriptor formats.

- For each Gaussian file, the code
    - Prints the title of the original Gaussian output file to distinguish each chemical species (this will help researchers identify what species it is and helps separate each of the species);
    - Sequentially generates and prints the molecular descriptors using the Canonical SMILES, SMILES, InChI, and Sybyl MOL2 formats; and
    - Creates the 2-D molecular representation of each molecule and saves it into an SVG image file.

The list and number of molecular descriptor formats can be modified so that certain other output types can be added or subtracted based upon need and versatility. The input format for this conversion code can also be changed, but only to a limited extent. For example, simulation results other than that of Gaussian outputs could be used theoretically (Molpro, GAMESS, simple xyz, etc.), but the user must ensure that the format has sufficient detail to generate the given molecular descriptor output formats of interest. It is easy to simplify and make it less dense, but extracting detailed information from a more simplistic structure is extremely difficult if not impossible. For example, we cannot generate a complete Sybyl MOL2 file from an xyz input because the xyz format lacks explicit information about the bonding and connectivity of a given molecular species.

The second segment of code is used to identify duplicates from both the Canonical SMILES and SMILES. This code compares all the generated Canonical SMILES and standard SMILES strings searching for duplicate output strings from different molecular information (i.e., non-unique strings) and then reports the strings and the respective input Gaussian files in question for further analysis.

3. Summary

This progress report highlights our recent efforts to consider tools and formats that could provide utility in future programs leveraging machine learning for our chemical kinetics mechanism development and modeling efforts. We evaluated several molecular descriptor formats that have potential utility: SMILES, Canonical SMILES, InChI, InChI Keys, and Sybyl Mol2. SMILES and Canonical SMILES have similar formats, but SMILES (due to the proprietary nature of the algorithm)

may be problematic because of possible molecular duplicate issues. Canonical SMILES strings, which are completely unique, are the recommended of the two (its only disadvantage relative to SMILES being the additional time needed to generate a string). Neither of these two formats have the complete molecular information needed to describe all the desired features of a molecule. InChI can describe more-complex molecular structure, but its strings can become quite long for larger molecules. InChI keys are unique 27-Character Keys generated from InChI, but they are machine-readable-only. The Sybyl Mol2 format is quite extensive, providing the molecular coordinates and bonding patterns of a given molecular structure, but they cannot be easily interpreted into a 3-D structure from visual inspection.

We find that ultimately a combination of these formats will be needed to provide the flexibility required for future machine learning research.

We have provided the code and concise instructions to set up a conda environment and begun testing the functionality of the Python Open Babel modules (see Appendix B for installation instructions). The Open Babel code has features/functionality beyond the scope of what was considered here. However, for the uses noted, the code works well and could be readily modified/extended as needs arise.

4. Future Directions

After this segment of the project is done, more complexity will be added to the code and database. One goal will be converting entire elementary chemical reactions with multiple molecules, not just the conversion of individual chemical species. An expanded list of output file formats will also be considered, with each format providing a different purpose within the database—acting as a translator for and within the team.

The code will also be developed to check for the possibility of duplicates or other mistakes within the outputs. For this to be done we must better understand the behavior of SMILES and Canonical SMILES string generation. To avoid the possibility of duplicates, we will code to determine the number of possible SMILES string generations and the likelihood of duplicates.

Additionally, the code may need be extended to refining the optimal molecular descriptor types needed within a machine-learning framework. Other considerations will be using (if possible) a connectivity representation to describe molecular connectivity changes within the bonding pattern of reactant molecules as they break apart/rearrange in going from reactants to products.

5. References

Ash S, Cline M, Homer RW, Hurst T, Smith G. SYBYL line notation (SLN): a versatile language for chemical structure representation. J Chem Inf Comput Sci. 1997;37(71).

Belford R. Chemical representations on computer. Part III. Chemistry LibreTexts; 2022 May 7 [accessed 2022 Aug 15]. https://chem.libretexts.org /Courses/University_of_Arkansas_Little_Rock/ChemInformatics_(2017)%3 A_Chem_4399_5399/2.3%3A_Chemical_Representations_on_Computer%3 A_Part_III.

Chen CC, Veals JD, McQuaid MJ. Modeling ammonia borane-red fuming nitric acid combustion. Part 1: gas-phase finite-rate chemical kinetics mechanism development. Proceedings of the 49th JANNAF Combustion Subcommittee Meeting; 2019.

Frisch MJ, Trucks GW, Schlegel HB, Scuseria GE, Robb MA, Cheeseman JR, Scalmani G, Barone V, Petersson GA, Nakatsuji H, et al. Gaussian 16, rev. C.01. Gaussian, Inc.; 2016.

Heller SR, McNaught A, Pletnev I, Stein S, Tchekhovskoi D. InChI, the IUPAC International Chemical Identifier. J Chemi Inf Model. 2015;7(23).

Homer RW, Swanson J, Jilek RJ, Hurst T, Clark RD. SYBYL line notation (SLN): a single notation to represent chemical structures, queries, reactions, and virtual libraries. J Chem Inf Model. 2008;48(2294).

Hutchison, GR, Morley C, James C, Swain C, De Winter H, Vandermeersch T, O'Boyle NM. Open Babel documentation. 2021 Mar 26. https://buildmedia.readthedocs.org/media/pdf/open-babel/latest/open-babel.pdf.

[IUPAC] International Union of Pure and Applied Chemistry. Color books [accessed 2022 Aug 15]. https://iupac.org/what-we-do/books /color-books/.

Leigh GJ, Favre HA, Metanomski WV. Leigh GJ, editor. Principles of chemical nomenclature: a guide to IUPAC recommendations. Blackwell Sciences LTD; 1998.

Long B, Bao JL, Truhlar DG. Rapid unimolecular reaction of stabilized Criegee intermediates and implications for atmospheric chemistry. Nat Commun. 2019;10:2003.

Martin D, Amado, AM, Gonzálvez AG, Marques MPM, Batista de Carvalho LAE, Ureña ÁG. FTIR spectroscopy and DFT calculations to probe the kinetics of β-carotene thermal degradation. J Phys Chem A. 2019;123:5266.

McQuaid MJ. An opposed-flow diffusion flame simulation-based implementation of the trial mechanism method for chemical kinetics mechanism reduction; application to the San Diego mechanism for HTPB-air combustion modeling. Army Research Laboratory (US); 2020 Aug. Report No.: ARL-TR-9032.

McQuaid MJ, Chen CC, Veals JD. Modeling ammonia borane-red fuming nitric acid combustion. Part 2: detailed finite-rate chemical kinetics mechanism reduction and validation. Proceedings of the 49th JANNAF Combustion Subcommittee Meeting; 2019.

O'Boyle NM. Towards a universal SMILES representation - a standard method to generate canonical SMILES based on the InChI. J Cheminform. 2012;4(22). https://jcheminf.biomedcentral.com/articles/10.1186/1758-2946-4-22#citeas.

O'Boyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR. Open Babel: an open chemical toolbox. J Cheminform. 2011;3:33. https://doi.org/10.1186/1758-2946-3-33.

Pybel. Open Babel: the open source chemistry toolbox [accessed 2022 Aug 15]. https://openbabel.org/docs/current/UseTheLibrary/Python_Pybel.html.

Southan C. InChI in the wild: an assessment of InChIKey searching in Google. J Cheminform. 2013;5(10).

Weininger D. SMILES, a chemical language and information system: 1: introduction to methodology and encoding rules. J Chem Inf Comput Sci. 1988;28(1):31–36.

Ye C-C, An Q, Cheng T, Zybin S, Naserifar S, Ju X-H, and Goddard WA. Reaction mechanism from quantum molecular dynamics for the initial thermal decomposition of 2,4,6-triamino-1,3,5-triazine-1,3,5-trioxide (MTO) and 2,4,6-trinitro-1,3,5-triazine-1,3,5-trioxide (MTO3N), promising green energetic materials. J Mater Chem A. 2015;3:12044–12050.

Appendix A. Code Details

The main code* is found in Fig. A-1.

```
import glob
from openbabel import pybel
filelist = glob.glob("**.out")
#filelist = glob.glob("**.out.txt")

csList = []
smiList = []
for file in filelist:
    print("Info for: ",file)
    mol = next(pybel.readfile("g16",file))
    outtypes = ['can','smi','InChI','InChIKey','sy2','svg']
    for types in outtypes:

        if (types == 'can'):
            canSMI = str(mol.write(format=types)).strip().split()
            print("Canonical SMILES: ",canSMI[0])
            csList.append(canSMI[0])
        elif (types == 'smi'):
            SMILES = str(mol.write(format=types)).strip().split()
            print("SMILES: ",SMILES[0])
            smiList.append(SMILES[0])
        elif (types == 'InChI'):
            ChI = str(mol.write(format=types)).strip().split()
            #print(mol.write(format=types))
            print("InChI: ",ChI[0])
        elif (types == 'InChIKey'):
            Key = str(mol.write(format=types)).strip().split()
            print("InChIKey: ",Key[0])
        elif (types == 'sy2'):
            Sybyl = str(mol.write(format=types))
            print("Sybyl MOL2: ",Sybyl)
        elif (types == 'svg'):
            mol.write('svg', file+ '.svg','overwrite=True')
        else:
            print(mol.write(format=types))
    print("#########################################################")
    print("    ")
```

Fig. A-1 Screenshot of the main code taken from a Jupyter notebook

* For this code segment to work, the program should be executed from a directory containing the Gaussian files, as the code must know where to look to find the files. To change directories, use the -cd command by writing cd and then the specific folder address (if using a bash emulator). This should also eliminate the extra address information that can be in each of the title printouts of each file.

The code for identifying duplicates from solely Canonical Simplified Molecular-Input Line-Entry System (SMILES), and not standard SMILES is shown in Fig. A-2.

```
#the following two lines will find and remove duplicates within Canonical SMILES
csNoDupList = list(set(csList))
csNoDupList

print("Canonical SMILES Duplicates:  ")

##   The block below first attempts to identify duplicate indices in the list
## of canonical smiles.
#csList

# set() sorts list and remove duplicate canonical SMILES strings,
# will only print out 1 instance to indicate the duplicate below
csNoDupList = list(set(csList))
for items in csNoDupList:

    python_indices = []
    if (csList.count(items)>1):
        print(items)
#        print(csList.index(items,3,32))

# below for loop construct to loop over all list items and indicate when the item in the
# there are multiple items with the same string value. By comparing each list element to a known
# value
        for cs in range(len(csList)):
            if csList[cs] == items:
                python_indices.append(cs)
#        print(python_indices)                  #can turn on to do a sanity check
        for idx in range(len(python_indices)):
#            print(idx)

            print(str("file Number_").strip()+str(idx).lstrip(" ")," :  ", filelist[python_indices[idx]])
#       print("second file name: ",filelist[python_indices[1]])
```

Fig. A-2 Screenshot of the code that identifies duplicates, taken from a Jupyter notebook

Appendix B. Gaining Access to Required Codes

This brief tutorial is for Windows users; modifications may be needed if working on a different operating system. Anaconda 4.13.0, Python 3.9.12, and Open Babel 3.1.0 versions were used to create and test the code.

Tool chain and workflow setup:

1. Download and install Anaconda from their website (anaconda.com/downloads), ensuring the right version of Python is being installed. A standard installer is available for this purpose.
    a. Open and run the downloaded .exe installer
    b. Click the "Next" and "Agree" options when prompted
    c. Open the Anaconda Prompt
2. Create an "environment" within Anaconda using the anaconda command prompt.
    a. This can be done by typing the following prompt into Anaconda
        i. conda create --name *newenvname*
    b. Indicate yes whenever the [y/n?] option appears.
    c. Next, type the following commands within the environment.
        i. conda config --set channel_priority false
        ii. conda update --all --yes
    d. To activate this environment for the future, which will need to be done every time the command prompt is closed or a different environment is activated, type:
        i. conda activate *newenvname*
3. Ensuring the environment is activated, install Open Babel within Anaconda. Within the command prompt, type
        i. conda install -c conda-forge Openbabel (*approach currently used*)

        or

        ii. conda install -c conda-forge/label/cf202003 openbabel
4. Although two other Python modules are used within the code (i.e., glob and pybel), glob should be installed already and pybel is automatically installed along with Open Babel.

5. Throughout the process, the Jupyter Notebook application was used to test iterations of the code in small chunks. Jupyter Notebook is typically already installed within Anaconda, and to open a notebook, type at the anaconda command prompt:

    i. jupyter notebook

    b. This should take you to a web browser tab (Microsoft Edge, Firefox, Google Chrome, etc.), where you can test small sections of code.

6. Two supplemental files containing the code have been provided as .txt files. To use them, either directly copy paste the code into Jupyter Notebook or change the ".txt" extension to ".py" to directly use Python from the command line.

    a. These files were provided separately because they need to be ran separately, with the main code given having to be run first, and the code finding duplicates running after.

List of Symbols, Abbreviations, and Acronyms

2-D	two-dimensional
3-D	three-dimensional
InChI	International Chemical Identifier
IUPAC	International Union of Pure and Applied Chemistry
SMILES	Simplified Molecular-Input Line-Entry System
SVG	Scalable Vector Graphics

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLD DCI TECH LIB
2 (PDF)	DEVCOM ARL FCDD RLW WC J VEALS C STONE