**National Cancer Institute (NCI)-US Department of Energy (DOE) Collaboration Use Case for JTVAE on Azure Machine Learning**

**Introduction**

This Use Case Template is intended to describe and interpret real-world examples of computational resources developed by the NCI-DOE Collaboration—and others—that can be used to explore solutions to complex cancer-risk factors, cancer diagnosis and treatments, as well as related biomedical research challenges. This template was developed with input from external subject matter experts. We ask you to share your experience in using these computational resources by completing this template in accordance with the instructions below. To provide feedback please contact *computational-cancer@nih.gov* Thank you!

**Instructions**

Please fill in the information requested. Each section includes the level of expected length or level of detail: **brief** (one-two sentences), **informative** (two-four sentences), and **detailed** (as much detail as required to properly inform the user). As each use case is different, please provide additional information as necessary.

**Use Case Overview**

Provide a brief overview of: *(1) use case goals and research focus; (2) methods, data type, and computational resource used; and (3) results and expected outputs. What is the expected time to reproduce this use case?*

This use case aims to provide users of the Junction Tree Variational AutoEncoder (JTVAE) pipeline with a template for application on Azure Machine Learning Studio (AML). For this example, we have trained an autoencoder on a subset of the zinc dataset composed of 220,000 drug like compounds, using a vocabulary of molecular components. This use case can be used to train a new autoencoder that can be used in the Generalized Generative Molecular Design (GGMD) pipeline described in a separate use case. In this example, a pipeline is created to complete all four steps, vocabulary generation, preprocessing the dataset, training the model, and reconstruction accuracy. A cpu cluster (Standard\_E4ds\_v4 (4 cores, 32 GB RAM, 150 GB disk)) is used for every step besides the training, which uses a gpu setup (Standard\_NC24s\_v3 (24 cores, 448 GB RAM, 2948 GB disk)). After completion of this use case, a user can expect 10 epochs of a training session, and a final reconstruction accuracy calculation. This use case on the zinc dataset will take approximately 8 hours from start to completion. You can final all the necessary files in the GitHub repo at <https://github.com/joverhul/JTVAE_Azure>. For instructions to run JTVAE not on AML please go to https://github.com/CBIIT/JTVAE.

**Use Case Details**

Scientific Background and Goals

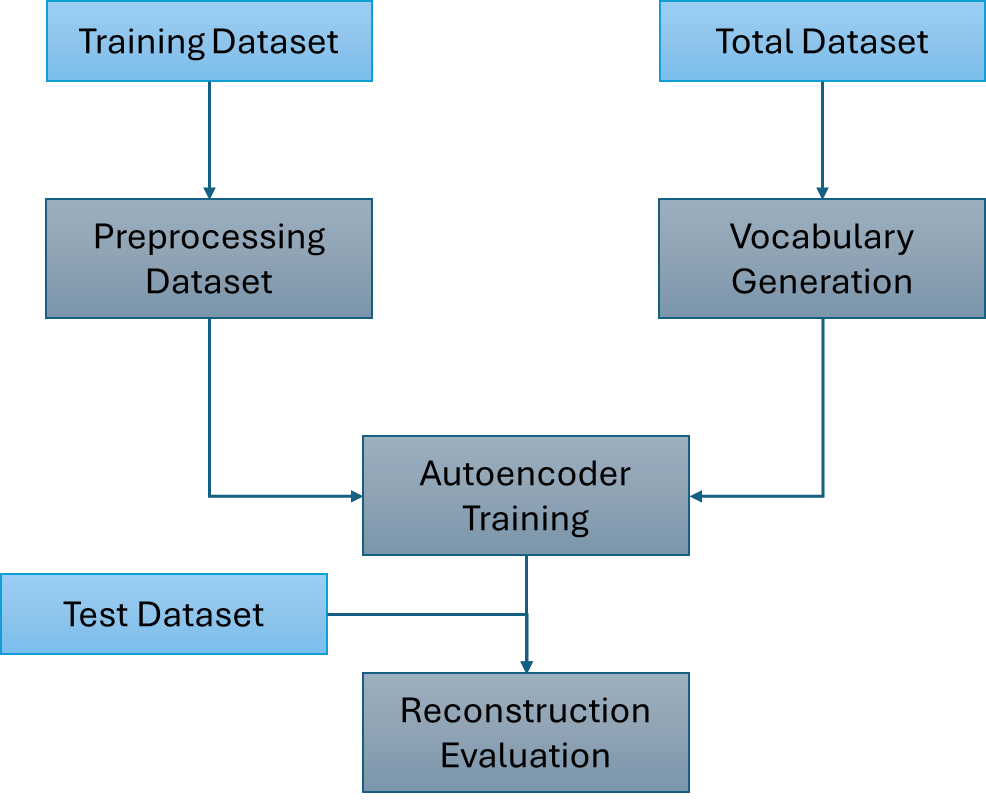
* **Problem**: This use case uses a previously used data from a Zinc dataset to train an autoencoder which can be used in other tools such as the GGMD pipeline. Users will use Azure Machine Learning (AML) resources to perform JTVAE pipeline.
* **Background**:JTVAE is a useful tool that drug discovery groups have been using to generate novel compounds, exploring the chemical space. JTVAE is an autoencoder that rebuilds moleculars from subgraphs built from a vocabulary of molecular components. This process works by encoding a smiles string into latent vectors, manipulate these vectors to produce optimized features, and then decoding the vectors back into the smiles string. Users can use Azure Machine Learning (AML) resources to perform the generalized generative molecular design pipeline. Previous documentation shows setup on an HPC or local computer, but this implementation is specific to AML.
* **Scientific purpose**: This use case shows the implementation of the JTVAE pipeline on a cloud computing platform, specifically AML. A user can produce their own autoencoder to be used in the GGMD pipelines or other pipelines if they do not have access to their own HPC systems. This use is best positioned for individuals who do not have access to their own HPC resources but are able to use cloud resources. Use of cloud resources also gives access to using large datasets with computational times being reduced.
* **Goal**: The goal of this use case is to provide an example for future users to use on Microsoft Azure, or other cloud computing services. The long-term goal is to allow users to utilize the cloud services to run generative design on much larger datasets than what was provided. This use case also highlights the ability that users who do not have their own HPC resources are able to use the provided resources from cloud services.
* **Objective(s)**: This use case provides an example of training a dataset (Zinc dataset) to produce an autoencoder which can be used to optimize compounds. This use case shows how to use the JTVAE method using AML, which can be used to speed up computational time, as well as gives access to users without direct access to a HPC.
* **Constraints**:Users who have access to their own HPC may not benefit from this specific use case on Azure. Another constraint to this use case is if the users have a complex dataset, there could be a reduction in reconstruction accuracy. This use case is developed specifically for Microsoft Azure.

User Background and Skills

* **Education**: This use case is specifically developed for Microsoft Azure, so having a working knowledge of how to run .yaml files and submit jobs on Microsoft Azure Machine Learning is recommended. Also, having a general knowledge of generative models and chemistry is recommended.
* **Skills required**: This use case requires a basic understanding of python coding, and running jobs with .yaml files, specifically the format needed for Azure Machine Learning. Additionally, having a basic understanding of the JTVAE pipeline from the original paper is recommended. Some experience with Microsoft Azure Machine Learning is recommended.

Methods

* **Computational resource selection**: Microsoft AML was selected due to its cloud use capabilities. Users can upload data assets to their storage bin, and then access these assets to run jobs. These resources give users the ability to use cloud services if they do not have access to their own HPC which could also be beneficial to run larger datasets.
* **Inputs**: Using the github, the inputs needed are clearly described and an example if provided. The inputs from the example include the training dataset (train.txt), the total dataset (total.txt), and the test dataset (test.txt). These inputs should be stored as assets on AML. The inputs necessary for each step of the process should be properly inputted into the respective yaml job submission files. These inputs are customizable, and users should provide their own total, train, and test sets to explore the chemical space they are interested in.
* **Methods**:This use case provides the documentation to train an autoencoder specific to the chemical space the user is interested in. In this specific example, the chemical space is the Zinc dataset. Before use this use case, users must have their own smiles lists of total, train and test sets as described above. This use case requires both a CPU cluster (Standard\_E4ds\_v4 (4 cores, 32 GB RAM, 150 GB disk)) to be used for every step besides the training, and GPUs for training (Standard\_NC24s\_v3 (24 cores, 448 GB RAM, 2948 GB disk)). Instructions for using this tool on AML can be found at https://github.com/joverhul/JTVAE\_Azure/blob/main/AML\_instructions.md.
* **Workflow figure**:

**

*Diagram

AI-generated content may be incorrect.*

This figure is an example of what to expect from the output if the pipeline is properly set up.

* **Methods customizability**: This use case is highly customizable. The provided example datasets is the Zinc dataset, but users can use any starting point smiles strings to train their autoencoder on their specific chemical space they are interested in.
* **Common pitfalls**:Some common issues with this computational resource is not correctly adding the assets to the yaml file and within the job submission. Other pitfalls could include issues with the computational resources on Azure, which may crash occasionally. Other pitfalls may include incorrectly setting up the pipeline to kick jobs off efficiently.
* **Requirements**:This use case requires dedicated cpu resources. The jobs can run on non-dedicated resources but could result in jobs failing due to time depending on parameters set. This use case requires both CPUs and GPUs.

Results

* **Outputs**: The final output will result in a folder of pkl files from the preprocessed datasets, a vocabulary text file, a folder of autoencoders trained to the number of epochs interested in, and the reconstruction evaluation output. Users area able to run each individual job not as a pipeline if that is preferred to monitor more diligently the outputs after each step.
* **Results**: The output of this specific use case will produce 10 epochs of autoencoders. After the completion of the 10th epoch, this autoencoder is used to test the reconstruction accuracy using a test set not a part of the training set. Below is an example output of this calculation.

Number of SMILES = 5000

Batch size in joblib = 20

Verbose in joblib.Parallel = False

Print failed reconstructs = False

Starting latent\_vectors: Total time (to load data/model) = 2 seconds

Encoding computation time, Total time = 94, 96 seconds

Decoding computation time, Total time = 605, 701 seconds

Number, percent of successfully reconstructed SMILES = 3996, 79.92 %

Discussion

* **Results interpretation:** Other The results for this example produces the generated vocabulary list, preprocessed pkl files, an autoencoder trained to the inputted epoch number, and the reconstruction evaluation. Users should be aware that if you need to restart the model training due to lower reconstruction accuracy, you need to add the correct model and optimizer file as a data asset in AML and properly call that folder when restarting. The user should also be wary that the final epoch training may be over trained, and using a previously saved epoch may produce better results.
* **Objective(s) met**:The original objective was met, and users are able to train their own autoencoder using the provided GitHub link. This use case is an example. In practice I would recommend following similar to the original paper training up to 40 epochs. This process will take ~72 hours but will increase the reconstruction accuracy to >80%. Additional examples have been trained to produce autoencoders trained on the coconut natural product dataset.

Conclusion

* **Limitations:** The limitations of this use case are related to the complexity of the input molecules. With too complex of molecules occasionally the pipeline will break. Users can look to the code to print out the specific smiles strings that are causing the problems and manually remove them to continue the training. Other limitations include use of non-dedicated computing resources, which may result in failed jobs on AML. This use case could also be used as a template with other cloud services if they have similar infrastructure as AML.
* **Goal**: *This work highlights an example use of the JTVAE pipeline on cloud services. This will give researchers the ability to use this pipeline on services if they do not have access to their own HPC as well as if they are using a larger dataset.*

**Key Links to Related Portal Pages:**

*Related resources or activities relevant to this use case may be found at the links below.*

*NOTE: these are intentionally not hyperlinked because the site is under development.*

* Publications

Wengong Jin, R. B., Tommi Jaakkola (2019). "Junction Tree Variational Autoencoder for Molecular Graph Generation." arxiv.

* Computational Resources: Models, Software, Datasets

https://github.com/CBIIT/JTVAE

https://github.com/joverhul/JTVAE\_Azure

* Educational Materials

<https://computational.cancer.gov/model/junction-tree-variational-autoencoder-zinc>