A benchmark study of junction tree-based variational autoencoders for molecular optimization

Team: Drugs"R"Us

Brian Yoo
netid: briany6
briany6@illinois.edu

Ping Wang
netid: pingjw2
pingjw2@illinois.edu

Aijun (Peter) Zhang
netid: aijun2
aijun2@illinois.edu

ABSTRACT

We benchmark three state-of-the-art variational autoencoders (VAE) designed for molecular optimization. Specifically, we reproduce the results of Jin *et al.* by retraining their VAEs – the junction tree variational autoencoder (JT-VAE), the junction tree encoder-decoder with attention mechanism and graph-to-graph translation (JTNN), and the hierarchical junction tree encoder-decoder (HierVAE). The goal of these autoencoders is to efficiently generate new molecules with improved performance properties suitable for drug discovery applications. We report training times, percent chemical validity of SMILES, average improvement in property and diversity of the generated set, and finally the success rate of the models trained on multiple molecular optimization tasks.

1. INTRODUCTION

Discovering and developing new, successful drugs have become an increasingly difficult task for pharmaceutical companies in the past decade. Today, pharmaceutical companies are spending significantly more to come up with fewer successful drugs, with an estimated cost of bringing one to market nearly doubling between 2003 to 2013 to $2.6 billion [1]. This is in part because most drugs that are safe and effective for common diseases have been discovered, and what is left are the lesser common diseases that are more complex and substantially more difficult to treat. Hence, AI and deep learning methods for drug discovery have garnered much attention from both the pharmaceutical industry as well as investment firms, given its vast potential to improve the "hit to lead rate", i.e., the percentage of drug candidates that first make it through screening and then to clinical trials, which traditionally has taken over 10 years to go from lab to market with the majority of them failing human trials.

Drug discovery is essentially a combinatorial, multi-objective optimization problem. It is estimated that there are 10^{33} to 10^{60} drug-like molecules [2] and anywhere from tens to hundreds of target proteins for a single complex disease for which a (small molecule) drug needs to bind onto to effectively treat the disease. Moreover, these drugs must satisfy several additional criteria such as being nontoxic and soluble (e.g., in blood and/or within the lymphatic system) in human trials. Brute-force screening through such vast chemical space and optimizing drug design for multiple objectives is experimentally and even computationally intractable. At present, virtual screening of 10^6 to 10^9 molecules can be achieved within a reasonable time frame (i.e., less than a year) using state-of-the-art supercomputers [3; 4; 5]. Although this is a commendable achievement, the vast majority of the molecules that are screened are often unsuitable for the given application. Hence, there is a growing need to develop AI and deep learning methods that can more efficiently navigate and identify relevant regions of chemical search space that optimize for target molecular properties.

The current state-of-the-art approaches for de novo drug design include deep learning methods that can efficiently identify molecular structure optimized for structural features (e.g., molecular weight, number of aromatic rings, etc.), physicochemical properties (e.g., topological polar surface area (TPSA), log water-octanol partition coefficient (logP)), similarity or dissimilarity to other molecules or existing drugs (e.g., based on the quantitative estimate of drug-likeness (QED)), and ligand-docking (i.e., the ability for the drug to bind to a target protein, usually as an inhibitor) [6; 7]. Overall, predictions from deep learning approaches are meant to help more efficiently screen possible drug candidates that can identify hits and eventual leads to be used in clinical trials.

In the past several years, variational autoencoders (VAE) trained on large libraries of small molecule simplified molecular-input line-entry system (SMILES) strings have shown promise in generating new molecules that are optimized for molecular properties [8; 9; 10]. In general, these approaches are generative models that encode SMILES strings into latent representations, which are subsequently trained with a surrogate model to predict and optimize the molecular property of interest. By tuning the noise parameter in the VAE in latent space, the VAE can be decoded into the SMILES representation with variation from the original molecule SMILES string. This, in turn, yields new molecules that may have not yet been discovered but those that are still optimized for molecular property. However, many of these approaches suffer from the decoding step where the decoded SMILES strings end up being chemically invalid (i.e., they syntactically do not represent a real molecular structure), or from generating molecules that are too similar in chemical structure with the original molecule. Consequently, there have been several recent methods (e.g., junction tree VAE, self-referencing embedded strings, or SELFIES VAE) that can improve upon a VAE's ability to generate valid SMILES [11] while ensuring the chemical diversity of the translated molecular structures.

Recently, Jin *et al.* [9] have proposed using what they call junction tree variational autoencoder (JT-VAE) to generate SMILES with optimized properties. The JT-VAE incorporates a vocabulary of common molecular subunits or moieties (e.g., functional groups) that often comprise the molecular structures found in the vast majority of these chemical databases [13; 14; 15]. By representing each of these subunits as nodes on a graph connected by constrained edges (i.e., junctions), the JT-VAE can efficiently encode (with a message-passing network [16]) embeddings of molecules based on a subunits to subunits (or on a node-to-node) basis rather than by atom to atom, and thereby more effectively learn molecular representation.

In a subsequent study [10], Jin *et al.* studied how the JT-VAE could be adapted to translate low performing molecules X to high performing molecules Y (i.e., graph-to-graph translation) by training on datasets containing (X, Y) pairs of SMILES, then by benchmarking the performance of the model on a smaller test set of new X molecules. The authors called this approach the junction tree encoder-decoder (JTNN), which also included an attention mechanism and an adversarial scaffold regularizer (JTNN+GAN) to improve upon the validity and diversity, respectively, of the generated SMILES. Fu *et al.* [17] further enhanced the JTNN, by implementing a "copy and refine" (CORE) strategy, which allows the decoder to "copy" an original substructure from the input of the JTNN or "refine" by sampling during the generation step, a new substructure from the entire vocabulary. This improved the performance of the autoencoder, in particular, for generated molecules containing infrequent substructures.

Finally, in a most recent study, Jin *et al.* [12] developed a new variational autoencoder architecture called the hierarchical VAE (HierVAE), which introduces a hierarchical structure in the autoencoder and decoder. The added layer in the autoencoder architecture corresponds to a coarser representation of the molecule (i.e., motifs) with message passing connections between each layer. Here, motifs are defined as subgraphs or clusters that contain bridge bonds (e.g., aromatics that share two carbon ring atoms) as opposed to the original junction tree, which defines nodes as clusters of single rings or bonds. This was done to address the limitation of generating larger molecules (e.g., polymers) for which the previous autoencoders had lesser accuracy (i.e., the chance of generating a valid molecule was lessened as the generated molecule became larger). By introducing such hierarchical layers into the encoder-decoder, the authors were able to more efficiently generate molecules with improved validity and target performance. This model also improved the algorithmic performance in the decoder reportedly six-fold, since each motif generation step required fewer enumerations given the nature of its coarser representation.

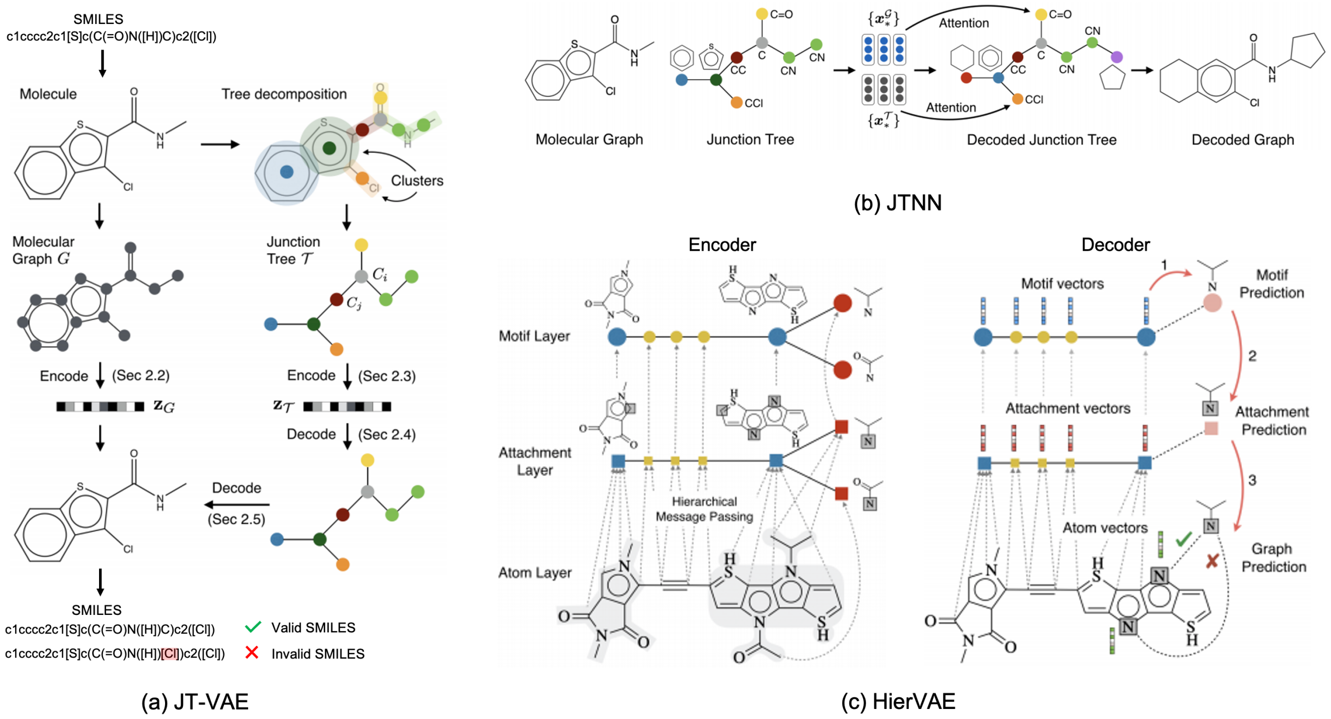
In this work, we present a benchmark study of the results obtained by Jin *et al.* with the junction tree VAE (JT-VAE), variational junction tree encoder-decoder with attention and graph-to-graph translation (JTNN), and the hierarchical junction tree encoder-decoder (HierVAE).

2. METHOD

We benchmark the junction tree variation autoencoder (JT-VAE), variational junction tree encoder-decoder (JTNN/graph-to-graph), and the hierarchical junction tree encoder-decoder (HierVAE) model based on improvements made to the penalized logP [18] (hereafter referred to as logP for simplicity), drug-likeness (QED) [19], and dopamine receptor score (DRD2) [20] and the diversity of the molecules generated provided a set of input molecules. In all three benchmarks, we also report the model training times, the percent chemical validity, and scores of the generated SMILES obtained from the validation set. All logP and QED scores are determined using the open-source library, RDKit [21] while the DRD2 score is determined using a pre-trained model from ref. [22].

Since the original junction tree VAE architecture is a core component in all subsequent models, it is instructive to outline the original modeling approach in more detail. Fig. 1a provides an overview of the JT-VAE method. A SMILES string is first encoded both as a molecular graph and as a junction tree, with graph encoder $q(\boldsymbol{z}^T|T)$ and tree encoder $q(\boldsymbol{z}^G|G)$, respectively. Each atom represents a node on the graph and molecular subunit represents a node on the junction tree. All subunits are based on a pre-defined vocabulary $\mathscr{X}$ containing 780 unique molecular substructures comprised of molecular rings and bonds. Here, a molecular graph is defined as $G = (V, E)$, which is comprised of the set of atoms in the molecule (i.e., vertices) V and bond connectivity (i.e., edges) E, while the corresponding junction tree $T = (\mathscr{V}, \mathscr{E}, \mathscr{X})$ is comprised of tree nodes $\mathscr{V}$ and edges $\mathscr{E}$ with label vocabulary $\mathscr{X}$. The junction tree is created based on vertex contractions algorithm performed on the molecular graph and then by subsequently obtaining a spanning tree of the resulting vertex contracted graph. Within the encoder, message passing is subsequently performed on both G and T by aggregating node and edge attributes containing information on the atom type, valence, and other molecular properties, or by one-hot encoding vectors of the label types and updating the vectors with gated recurrent unit, respectively, to obtain the resulting latent representations $\boldsymbol{z} = [\boldsymbol{z}^T, \boldsymbol{z}^G]$. The latent representation from the junction tree is decoded with a tree decoder $p(T|\boldsymbol{z}^T)$ and used to reconstruct the original junction tree (one node at a time). Secondly, the atom-level connectivity within nodes of the junction tree is then predicted based on a graph decoder $p(G|T, \boldsymbol{z}^G)$ to reconstruct the original molecular graph and corresponding SMILES. For more specific algorithmic detail, we refer the reader to ref. [9].

For JT-VAE, 220K SMILES strings (out of 250K total) in the ZINC database were used for training with the target property optimized in latent space using gradient ascent. For the JTNN and HierVAE, 34K to 99K "translation pairs" of SMILES were also obtained from the ZINC database and subsequently used for training. Translation pairs refer to the previously described (X, Y) SMILES pairs that have a source SMILES X with poor target performance, and a paired outcome SMILES Y with significant improvements



(a) JT-VAE

(b) JTNN

(c) HierVAE

Figure 1: Overview of the methods benchmarked in this study: (a) the junction tree variational autoencoder (JT-VAE), (b) the junction tree encoder-decoder (JTNN) with attention, and (c) the hierarchical junction tree encoder-decoder (HierVAE). Images have been adapted from ref. [9; 10; 12]. For all model architectures, the SMILES string is initially encoded as a graph (G) and tree (T) followed by an embedding step. The embedding vectors can then be decoded to obtain the original junction tree and subsequently a molecular graph structure. The validity of the decoded SMILES in the decoder is determined based on chemical validity (e.g., allowable valency of atoms, allowable bond connections, etc.) as shown in panel (a). In the JTNN method, an attention mechanism is added to the encoder. In the HierVAE approach, a coarser, motif-based junction tree encoder is created in conjunction with a fine-grained attachment layer (similar to the original junction tree). In all three approaches, new molecules can be generated by perturbing (i.e., adding noise to) the latent vectors, as commonly performed in variational autoencoders.

.

to the properties of interest and with a set structural similarity.

Each neural network model was independently trained for each target property: penalized logP (with 40% or 60% molecular structure similarity constraints as denoted by $sim \geq 0.4$ and $sim \geq 0.6$, respectively), DRD2, and QED. In total, we trained 11 models (3 models for the JT-VAE method over 2 epochs, and 4 models for JTNN + 4 models for HierVAE over 20 epochs). Note the difference between training input between the JTNN and HierVAE approach with that of the JT-VAE approach. For the former two methods, pairs of SMILES are used during model training, with all pairs being at least 40% similar in structure (hence a smaller subset of SMILES pairs obtained from the ZINC database), whereas in the original JT-VAE method, the entire ZINC database is used in training with a separate Gaussian process model trained on latent space vectors to predict and optimize the target property of interest. For logP in the JTNN and HierVAE studies, both $sim \geq 0.4$ and $sim \geq 0.6$ pairs were used in training to ensure that the autoencoders do not translate molecules into arbitrary chemistries. The JT-VAE and JTNN methods are trained based on minimizing the reconstruction loss with Kullback Leibler (KL) regularization, whereas the HierVAE minimizes the negative evidence lower bound with KL regularization.

Both the JTNN and HierVAE methods train SMILES pairs based on the differences in graph and tree embeddings between those obtained for Y and X. An overview of this pairwise training approach is shown in Fig. 2. For each X and Y SMILES, a graph ($\boldsymbol{z}^G$) and tree ($\boldsymbol{z}^T$) embedding are first generated. The differences in embeddings for X and Y (δ_Y^T=$\boldsymbol{z}_Y^T$-$\boldsymbol{z}_X^T$ and δ_Y^G=$\boldsymbol{z}_Y^G$-$\boldsymbol{z}_X^G$, respectively) are used during model training. By training on these "difference embeddings" and on the embeddings of the original source molecule X, the model learns to improve a molecular property by translating substructure within a given source molecule. To obtain new molecules, the difference embeddings are combined with the original source molecule embeddings and subsequently "perturbed" to obtain the embeddings $\widetilde{\boldsymbol{x}}_*^T$ and $\widetilde{\boldsymbol{x}}_*^G$. These embeddings can be decoded similarly to how decoding is performed in the original JT-VAE approach. Note that after training, the model only needs to be provided a

Table 1: Model training times in hours, number of parameters, and size of training sets in number of SMILES strings or SMILES translation pairs. Note that JT-VAE train sizes are based on individual SMILES whereas JTNN and HierVAE training sizes are based on the number of SMILES translation pairs. The JT-VAE based models were trained over 2 epochs while the JTNN and HierVAE based models were trained over 20 epochs, all using NVIDIA Tesla T4 GPUs on AWS.

Method	$\log P_{sim \geq 0.4}$			$\log P_{sim \geq 0.6}$			QED			DRD2		
	train time	# params	train size	train time	# params	train size	train time	# params	train size	train time	# params	train size
JT-VAE	34.4h	2479K	220K	34.4h	2479K	220K	34.5h	2479K	220K	34.5h	2479K	220K
JTNN	13.2h	3139K	99.9K pairs	16.4h	3142K	75.3K pairs	10.3h	3733K	88.3K pairs	7.5h	3902K	34.4K pairs
HierVAE	19.7h	8245K	99.9K pairs	15.4h	8302K	75.3K pairs	14.8h	7813K	88.3K pairs	6.6h	7751K	34.4K pairs

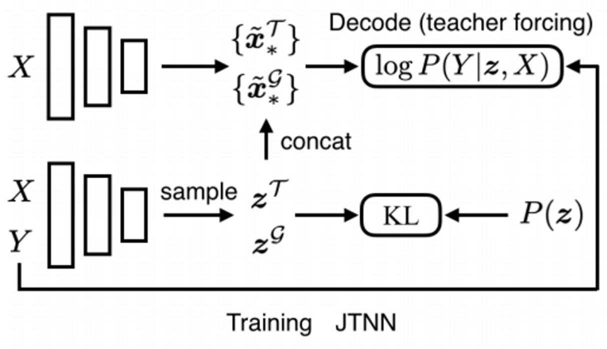


Figure 2: Method for training on translational (X, Y) pairs as performed in JTNN and HierVAE. Figure has been obtained from ref.[10].

source SMILES string to output a new molecule with improved property as opposed to pairs of SMILES, since the autoencoder is trained on the difference embedding vectors.

3. RESULTS

All models were trained with Pytorch [23] using NVIDIA Tesla T4 GPUs (g4dn.2xlarge or g4dn.xlarge) on Amazon Web Services (AWS) EC2 instances. Input data, trained models, and logs were stored in AWS S3, along with transient training and hardware performance detail (e.g., learning curves) for select models logged in the Weights and Biases platform [24]. All models contain dimension sizes of 300 for the hidden and embedding layers. Latent representation dimension was set to 56 for JT-VAE and 8 for JTNN and HierVAE. In the JT-VAE method, the regularization weight was set to $\lambda_{KL} = 0.005$ whereas $\lambda_{KL} = 0.3$ for the JTNN and HierVAE methods. For all methods, optimization was performed using the Adam optimizer with default hyperparameters. Graph and tree message passing iterations are set to 20 iterations per layer. To accelerate training for the JTNN and HierVAE methods, models were additionally preprocessed as done by Jin *et al.* to obtain the graph embeddings for each SMILES in the training set, before performing the actual training. The JT-VAE models were trained over 2 epochs, while the JTNN and HierVAE based models were trained over 20 epochs for the reasons noted previously. All models were trained on AWS in separate EC2 instances with GPU support. A subsequent validation step was performed after each epoch to determine the best model as means to prevent overfitting. We report the results obtained from the best-performing validation set based on its averaged property score. All log files can be found in our public GitHub repository[1].

Table 2: Percent validity of SMILES generated using the best model on the validation set. For each input SMILES, $K = 20$ new molecules are generated. For each property, we provide the average number of heavy atoms $\bar{n}_{HA}$ of SMILES within the validation set. Jin *et al.* report 100.0% validity for all SMILES decoded on their benchmark set. Note that validity performance, for JT-VAE and JTNN, however, is known to suffer significantly as the number of heavy atoms increase as noted in ref. [12].

Method	$\log P_{sim \geq 0.4}$	$\log P_{sim \geq 0.6}$	QED	DRD2
$\bar{n}_{HA}$	20.60	20.60	22.54	24.7
JT-VAE	100%	100%	100%	100%
JTNN	99.8%	99.6%	99.8%	100%
HierVAE	99.6%	91.1%	100%	100%

Table 1 presents the total training times in hours, the number of model parameters, and the size of training sets for each of the methods. Total training time may vary due to the differences in the number of parameters in the models, as well as the differences in the train sizes (i.e., the number of SMILES strings or SMILES string pairs). Interestingly, we find that HierVAE training times are marginally increased or similar to those for JTNN even though the models contain more than double the number of parameters. Additional learning curve analysis, gradient, and hardware utility information for select models with HierVAE is provided in our Weights and Biases dashboard[2]. (Note that we report a select number of models rather than all models to avoid retraining the same models twice and incurring additional costs from AWS.)

Table 2 shows the percent validity of SMILES for the molecules generated using the best performing models obtained based on the validation set. We also provide the average number of heavy atoms contained in each of the validation sets. For each of the four properties ($\log P_{sim \geq 0.4}$, $\log P_{sim \geq 0.6}$, QED, and DRD2), 2 models for JT-VAE and 20 models (1 for each epoch) for JTNN and HierVAE were benchmarked during validation. The model that performed the best in improving the target property on the validation set was used to obtain our benchmark results shown in the table. The validation set contains 500, 360, 200, and 200 source SMILES

[1] https://github.com/pingjiewang/sagemaker-dlh

[2] https://wandb.ai/peterzhang/drugsrus/reports/Wandb-Reports-A-benchmark-study-for-molecular-optimization--Vmlldzo2Njk1MzU

Table 3: Average improvement and diversity of the generated set for logP, and success rate and diversity of the generated set for QED and DRD2. Success rate for QED is defined as QED $\in$ [0.9, 1.0], whereas that for DRD2 is defined as values greater than 0.5. Diversity is defined as 1-$sim(X, Y)$.

Method	logP($sim \geq 0.4$)		logP($sim \geq 0.6$)		QED		DRD2	
	improvement	diversity	improvement	diversity	success rate	diversity	success rate	diversity
JT-VAE (this work)	0.48 ± 1.31	-	0.15 ± 0.56	-	10.5%	-	1.0%	-
JT-VAE (orig.)	1.03 ± 1.39	-	0.28 ± 0.79	-	8.8%	-	3.4%	-
JTNN (this work)	3.66 ± 1.64	0.500	2.00 ± 1.27	0.280	60.8%	0.369	74.6%	0.159
JTNN (orig.)	3.55 ± 1.67	0.480	2.33 ± 1.24	0.333	59.9%	0.373	77.8%	0.156
HierVAE (this work)	3.32 ± 1.63	0.479	2.22 ± 0.91	0.345	74.1%	0.444	74.4%	0.204
HierVAE (orig.)	3.98 ± 1.46	0.564	2.49 ± 1.09	0.381	76.9%	0.477	85.9%	0.192

for the DRD2, QED, logP$_{sim \geq 0.4}$, and logP$_{sim \geq 0.6}$ trained models, respectively. For each SMILES within a validation set, $K = 20$ new improved SMILES are generated for the JTNN and HierVAE methods using the trained models. For the JT-VAE method, $K = 1$, and thus diversity scores are not reported. Note that all validation runs are performed on CPU. Interestingly, we find that the chemical validity obtained for logP$_{sim \geq 0.6}$ with HierVAE has a slight decrease in performance. We suspect that this may be caused due to relatively small molecule sizes (i.e., number of heavy atoms) used in the benchmark set, as the HierVAE was primarily intended to address issues when predicting performance on large molecules.

Table 3 shows the average improvement in logP (with similarities ≥ 0.4 and ≥ 0.6), and the success rate of QED and DRD2. Success rate for QED is defined as QED $\in$ [0.9, 1.0], whereas that for DRD2 is defined as values greater than 0.5. The majority of our results are in agreement with those of Jin *et al.*, with some discrepancy between the results obtained using the JT-VAE and HierVAE method. Our JT-VAE models tend to underestimate the mean improvement and success rate for logP and DRD2, respectively. Similarly, our HierVAE models tend to underestimate mean improvement and diversity for logP($sim \geq 0.4$) as well as the success rate for DRD2. However, we note that all of our estimated means are within the estimated uncertainties of the results obtained by Jin *et al.*. Deviation from their results may be caused by the hyperparameters being slightly different in our work, as we chose to use the same hidden and embedding layer dimensions for all VAEs (300) whereas Jin *et al.* chose to size the dimensions based on matching the total number of model parameters (approx. 4M). For the JT-VAE method, we note that the number of epochs was 1 less than that performed by Jin *et al.* due to the time constraint of our project timeline.

Figure 3 presents the average decoding times for each of the VAE approaches. We find that on average for a given source SMILES, it takes approximately 2.98s to generate a single new SMILES using the JT-VAE method, 0.22s (or 4.4s for $K = 20$ molecules) using the JTNN method, and 0.063s (or 1.26s for $K = 20$ molecules) using the HierVAE method. Surprisingly, we find that the speedup is roughly 4 to 15 times faster for each architecture iteration (based on chronological order). This is also consistent with the reported speedup in decode time for the HierVAE relative to that of the JTNN reported by the original authors.

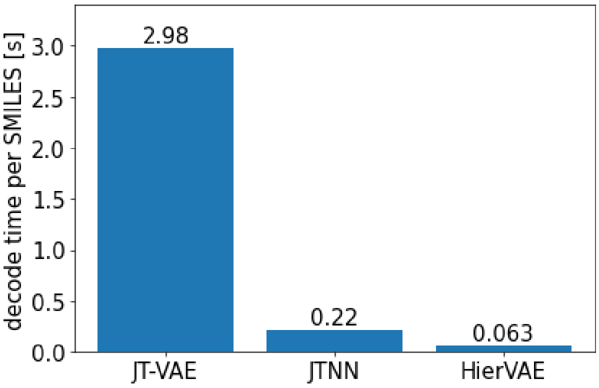


Figure 3: Average decode times per SMILES using the JT-VAE, JTNN, and HierVAE methods. Decode times are averaged over SMILES decoded within the validation sets.

.

4. CONCLUSION

Overall, our study reproduces results in excellent agreement with those that have been reported by Jin *et al.* for the JTNN and HierVAE. We were successfully able to retrain the models presented in the authors' work, albeit with some initial difficulty.

While the authors provided all of the code in the associated GitHub repositories, we faced several unexpected issues during this benchmark study. For instance, we found that parts of the code were particularly difficult to reproduce, as we learned later on that the code was quite sensitive to the version of RDKit that was installed (e.g., for HierVAE RDKit version 2019.03 must be used whereas for JTNN, older versions of RDKit can be used; likewise, for JT-VAE CUDA 8.0 version of Pytorch must be installed). This led to us encountering several issues with the code related to the vocabulary fragment generation and "preprocessing" steps implemented for HierVAE, which also then led to us contacting the lead author directly via GitHub issues as well as email to identify which specific version of RDKit was used in his study. Upon our inquiry, the author updated his README accordingly, providing updates to version specifications.

Moreover, during this study, we found that not only the training but also the validation step took quite some time, in particular for the JTNN method. This was because the validation step (i.e., identifying which model out of the 20 epochs for a target property performed best) utilizes the decoder, which does not leverage any GPU (whereas the training step did). To speed up this process, we modified the validation scripts so that they could be run in parallel on multiple CPU cores. On average, it took approximately 2.98s to generate a single new SMILES for a given source SMILES using the JT-VAE method, 0.22s (or 4.4s for $K = 20$ molecules) using the JTNN method, and 0.063s (or 1.26s for $K = 20$ molecules) using the HierVAE method. While these speeds may seem fast at first glance, they result in long validation times (on the order of hours) provided the large number of molecules that need to be decoded over many epochs. These relatively slow translation times are caused by the decoder having to enumerate all possible junctions (node by node) and subsequently bonds (atom by atom) that can be made for a given SMILES. Furthermore, for the JTNN and HierVAE methods, the decoding step repeats this task $K = 20$ times for each SMILES to obtain a diverse set of proposed molecules. The long decoding times can be further exacerbated especially when the resulting molecule is large. Hence, depending on the application, the performance of these decoding times may be prohibitive compared to traditional virtual screening approaches.

5. AUTHOR CONTRIBUTION

B.Y. prototyped the VAE methods. P.W. set up the AWS infrastructure to perform model training and storing in parallel, and performed the necessary modifications to the code to automate the entire process on AWS. P.W. and A.Z. performed all model training, validation, and logging. A.Z. modified the code to support logging onto the Weights and Biases platform. B.Y. wrote the manuscript with significant input from P.W. and A.Z.

6. REFERENCES

[1] David H Freedman. Hunting for new drugs with ai. *Nature*, 576(7787):S49–S53, 2019.

[2] Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of Cheminformatics*, 1(1), June 2009.

[3] Kristy A Carpenter, David S Cohen, Juliet T Jarrell, and Xudong Huang. Deep learning and virtual drug screening. *Future medicinal chemistry*, 10(21):2557–2567, 2018.

[4] David E Gloriam. Bigger is better in virtual drug screens, 2019.

[5] Anh-Tien Ton, Francesco Gentile, Michael Hsing, Fuqiang Ban, and Artem Cherkasov. Rapid identification of potential inhibitors of sars-cov-2 main protease by deep docking of 1.3 billion compounds. *Molecular informatics*, 39(8):2000028, 2020.

[6] Nathan Brown, Marco Fiscato, Marwin HS Segler, and Alain C Vaucher. Guacamol: benchmarking models for de novo molecular design. *Journal of chemical information and modeling*, 59(3):1096–1108, 2019.

[7] Stefano Forli, Ruth Huey, Michael E Pique, Michel F Sanner, David S Goodsell, and Arthur J Olson. Computational protein–ligand docking and virtual drug screening with the autodock suite. *Nature protocols*, 11(5):905–919, 2016.

[8] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[9] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning*, pages 2323–2332. PMLR, 2018.

[10] Wengong Jin, Kevin Yang, Regina Barzilay, and Tommi Jaakkola. Learning multimodal graph-to-graph translation for molecular optimization. *arXiv preprint arXiv:1812.01070*, 2018.

[11] Mario Krenn, Florian Häse, AkshatKumar Nigam, Pascal Friederich, and Alan Aspuru-Guzik. Self-referencing embedded strings (selfies): A 100% robust molecular string representation. *Machine Learning: Science and Technology*, 1(4):045024, 2020.

[12] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical generation of molecular graphs using structural motifs. In *International Conference on Machine Learning*, pages 4839–4848. PMLR, 2020.

[13] Teague Sterling and John J Irwin. Zinc 15–ligand discovery for everyone. *Journal of chemical information and modeling*, 55(11):2324–2337, 2015.

[14] Anna Gaulton, Anne Hersey, Michał Nowotka, A Patricia Bento, Jon Chambers, David Mendez, Prudence Mutowo, Francis Atkinson, Louisa J Bellis, Elena Cibrián-Uhalte, et al. The chembl database in 2017. *Nucleic acids research*, 45(D1):D945–D954, 2017.

[15] Daniil Polykovskiy, Alexander Zhebrak, Benjamin Sanchez-Lengeling, Sergey Golovanov, Oktai Tatanov, Stanislav Belyaev, Rauf Kurbanov, Aleksey Artamonov, Vladimir Aladinskiy, Mark Veselov, et al. Molecular sets (moses): a benchmarking platform for molecular generation models. *Frontiers in pharmacology*, 11, 2020.

[16] Kevin Yang, Kyle Swanson, Wengong Jin, Connor Coley, Philipp Eiden, Hua Gao, Angel Guzman-Perez, Timothy Hopper, Brian Kelley, Miriam Mathea, et al. Analyzing learned molecular representations for property prediction. *Journal of chemical information and modeling*, 59(8):3370–3388, 2019.

[17] Tianfan Fu, Cao Xiao, and Jimeng Sun. Core: Automatic molecule optimization using copy & refine strategy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 638–645, 2020.

[18] Scott A Wildman and Gordon M Crippen. Prediction of physicochemical parameters by atomic contributions. *Journal of chemical information and computer sciences*, 39(5):868–873, 1999.

[19] G Richard Bickerton, Gaia V Paolini, Jérémy Besnard, Sorel Muresan, and Andrew L Hopkins. Quantifying the chemical beauty of drugs. *Nature chemistry*, 4(2):90–98, 2012.

[20] Bruce R Lawford, Ross Young, Ernest P Noble, Burnett Kann, and Terry Ritchie. The d 2 dopamine receptor (drd2) gene is associated with co-morbid depression, anxiety and social dysfunction in untreated veterans with post-traumatic stress disorder: Supported by the greenslopes private hospital research foundation, australia, queensland university of technology, australia and the adele c. smithers-fornaci and christopher d. smithers foundation, united states. *European psychiatry*, 21(3):180–185, 2006.

[21] Rdkit: Open-source cheminformatics, 2021.

[22] Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics*, 9(1):1–14, 2017.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[24] Lukas Biewald. Experiment tracking with weights and biases, 2021. Software available from wandb.com.