AI in Drug Discovery and Development

Prof. Rajnish Kumar
Dept. of Pharmaceutical Engineering and Technology
IIT-(BHU), Varanasi

Week-09
Lecture-45

Welcome to the course "AI in Drug Discovery and Development." In today's session, we will have a hands-on session on de novo generative design using the graph invent tool. So, as we discussed earlier, the generation of new molecules using generative AI is one of the emerging areas. So, today we will have a hands-on session on how we can use Graph Invent, which is a de novo generative AI tool developed by AstraZeneca's molecular AI team, to generate a novel molecule. So, for that, we have to, you know, follow and open this link in a browser, typically Chrome, if we have it. So, you just copy this link and then open it in a browser; once you copy and open this link in the browser, what you will find is that it will open the Colab notebook.

Which is de novo drug design using graph invent dot ipynb, and as we discussed earlier, Google Colab is a very good platform. Where we can write code, execute code, install tools, and then plot. So it's a kind of all-in-one solution where you can do almost every type of coding, and then you can run all kinds of programs as well. And the beauty is that you are not using your local computational power; instead, you are using the computational power of the servers hosted by Google, and that is, you know, kind of cloud computing, actually.

So, in today's session, we will be using the graph invent. So, as we discussed during the lectures on generative AI and generative modeling for drug design. So, if you remember, we talked about the advantages of generative AI and those tools that are being used. and one of the biggest advantage was the coverage actually, because if we compare the virtual screening methods with the generative AI methods or de novo methods. So, the biggest advantage of de novo methods is the coverage because in virtual screening we can only explore a small amount of chemical space, and then we are only exploring what is actually known.

But in the case of generative AI or de novo drug design, we are exploring the unknown, and we are, you know, Trying to identify novel scaffolds that are very important in case we are pursuing intellectual property rights. So, it will be very easy for us to obtain intellectual property rights on those molecules if we generate a completely novel structure that is not known yet. And that is the beauty of, you know, the de novo generative tools. And another thing is that AstraZeneca is doing excellent work in this area, especially their

molecular AI team. So, what they have done is develop several tools, you know, like they have developed Re-invent, which is a chemical language model tool.

Where you are generating the smiles by learning the language of the smiles, how smiles are made, and then they have developed this graph invention. So, graph invent is a graph neural network-based method and graph-based technique. We are generating the molecules based on the graphs, and then they have also developed a tool called Dockstream, which is where you use the Reinvent platform in the backend. And you are using reinforcement learning, and you are using the docking score to feed that docking score into the re-invent to generate molecules that have a higher docking score for a specific target. That is also another beautiful tool that you can explore.

So, let us talk about this graph invention. So what you have to do first, as always, is make a copy of it into your local drive. so that you can import your files into Colab and then save the data to your drive as well. So, for that, you just click on File and then save a copy in Drive, and once the copy is done, you can open it in a new tab, okay? So, once you have made a copy of it, what you can do is, since this is kind of a graph neural net, it is computationally intensive. So, that was, if you remember, one of the disadvantages of it or the limitations, because if we are converting each molecule into a graph.

So, that takes a lot of memory as well as a load of computational power. So, these tools are a little bit slower compared to the chemical language model tools like the re-invent. And for that, we need to have a GPU, actually. So, you can just check whether we have got a GPU. So, you click on the runtime.

And then I can click on the change runtime type, and here you can see that we have CPU, so you can select the T4 GPU as well. Of course, you can run it on the CPU as well, but it will be a little bit slower, actually. And then you can just do that since I have run it already, so I will not change the runtime type because this takes a little bit of time to run. In this short session, it will not be possible, so I've already run it, and then I will show how it actually works. So, talking about the graph invents.

So, deep learning applied to chemistry can be used to accelerate the discovery of new molecules, and Graph Invent is a platform developed by graph-based molecular design using graph neural nets. Graph invent uses a deep neural net architecture to probabilistically generate new molecules one bond at a time. So, the models implemented in Graph Invent can quickly learn to build molecules resembling training set molecules without any explicit programming of the chemical rules. And the framework is implemented in PyTorch and provides high modularity, making it easy to experiment with different graph neural architectures. If you are interested, you can go to the full paper on

this by clicking on the ChemArchive link, which will take you to the full paper.

And then you can also refer to their GitHub repo, where you will get all the details. So, how this graph invention works is that the first thing is the pre-processing, which means graph deconstruction. So, as always, like those models, we need the training data set, right? So, in this case, for example, if we wanted to identify some molecules, first the model shall learn what a molecule looks like, right? So, for example, we can take 1000 molecules, and then the model will get to know the structures of those molecules. So, that is what we have to convert since this is a graph neural network-based method. So, we actually have to convert those molecular structures into graphs.

So, the molecules from the data sets are deconstructed step by step into smaller subgraphs using a modified breadth-first search, which is known as BFS. So, this process defines a decoding route that instructs the model how to reconstruct the molecule from an empty graph ok. And then each subgraph step is associated with an action probability distribution (APD) over possible actions, either adding an atom, forming a bond, or terminating. And then it has, if you look at the model architecture. So, it has two core blocks: one is the GNN block, which has six types: MNN, GGNN, S2V, Attention GGNN, Attention S2V, and EMN, which learn node and edge level features.

And then there is another block, which is the global readout block, that computes the APD for the entire graph using learned node features and embeddings. So, the APD contains three vectors: FADD, which adds a new node with the bond; Fconn, which connects the last node to another; and Fterm, which terminates the graph generation. And then for training, the APDs are the model's output, and the training goal is to minimize the KL divergence between the predicted and the target APDs. It uses mini-batches and the Adam optimizer trained using datasets like GDB13 and MOSES. So, for generation, it starts with an empty graph and, in each step, samples an action from the APD either to add the atoms, connect the atoms, or terminate.

It continues until a termination signal is generated or an invalid action occurs, such as exceeding the maximum atoms. So, that was just kind of a quick intro to what graph invent is and how it works. So, to run it, what we have to do is first prepare the environment we are running it on in Colab, actually. So, this is not an ideal way to run graph invent because, as I said, it is computationally intensive and resource intensive. So, it is better if you run it on your local workstation, but running it in Colab is the easiest way for me to demonstrate it.

So, that is why I thought we should do it on Colab. So that everyone can at least run and get a flavor of how the graph neural network de novo drug design using graph neural

networks or using graph invent works actually. So, first things first, we need to install the required dependencies and create an environment. So, we will need the Miniconda installation with Python 3.8, which is suitable for compatibility with Graph Invent's dependencies.

And then we will do the conda initialization and updates, which initialize the conda shell and update all conda packages for a stable environment. And then we do the system path update, where we add a newly installed Python package to the system path to ensure modules are discoverable by Colab. And then we do the graph invent cloning; you know we copy the repository from GitHub. And then we do the environment activation, where we activate the conda environment in the graph inventory yml, once it is created. So, for example, this code block will take 9 minutes to run; therefore, I will not run it because I have already run it.

So, you can run it and then you can see how it actually works. So, once we run it and the running is just clicking on this, you know, play button, it will create the environment and then it will install all the dependencies needed for running the graph, okay. So, if we run the next cell, what it will do is activate the graph environment that we have created. So, it will also install the ipykernel. Now you can see after running this.

So, it has installed, you know, activated the graph invent environment which we created, and then it has installed the IPY kernel as well. So, that we have done now. The next thing is once we have installed all the dependencies. Now, our model is able to run. So, we have downloaded the graph inventory from the GitHub repo, installed it, and now it is running.

So, the next thing we have to do is train a model, okay? But before training... So, we need to get the data, actually the training set, on which we will train our model. If you look at the paper, what they have done is train the model on different data sets, and some of those data sets are like the GDB 13 and 1K.

1K means it contains 1,000 molecules from the GDB data set, okay, and then so. During their model development, they used a training size, a test size, and a validation size of 1k, 1k, and 1k, respectively, and the maximum nodes were about 12k. And then, the atom types in the GDB 13.1 k were carbon, nitrogen, oxygen, sulfur, and chlorine, and the formal charges were plus 1, minus 1, and 0, okay, and likewise. So, this and then this they called is like RAND, ok, and then RAND is representing random, ok.

And then Canon is representing the canonical. So, and then GDB 13.1k canonical. So, again it was 1k in test, training, and validation size. The atom types were carbon, nitrogen, oxygen, sulfur, and chlorine, and the formal charges were again minus 1, 0, and plus 1.

And then they used a Moses as well. MOSES is like a larger database where they used, you know, 1.5 million in the training, 176k in the test, and 10k in the validation, and the maximum nodes were like 33 million. They have increased their scope by including fluorine and bromine as well. So, you can see the atom types in MOSES were carbon, nitrogen, oxygen, fluorine, sulfur, chlorine, and bromine, and likewise, they have MOSES cannon and MOSES aromatic. where the aromatic bond type is included explicitly.

So, the GDB13-1K is a small subset of the GDB13 dataset used for fast testing and hyperparameter tuning. And then RAND and Canon, as we talked, refer to random and canonical node ordering during pre-processing, and MOSES is a curated subset of the Zinc database focused on drug-like molecules. And there are three versions: random RAND, random deconstruction, canonical deconstruction, and arom, which includes the aromatic bond type explicitly. So, in this case as well, since you know how to say, it's a small exercise where I just wanted to show you and give you a flavor of how de novo drug design works. So, in this case, what we will do is take this small data set, the GDB 13 1k only.

So, for that, once we have downloaded the dataset, we need to, you know, prepare the dataset. So, once you have selected a dataset to study, for example, we have now selected the GDB 13 1K. So, you must prepare it so that it agrees with the formal format expected by the program, and graph invent expects three splits in this file's format for each dataset; each split should be named as train.

smi, test.smi, and valid.smi. So, this should contain the training set, the test set, and the validation set, respectively. It is not important for the files to be canonical, and it also does not matter if the file has a header or not; how many structures you put in each split is also up to us, actually. Generally, the training set is larger than the testing and validation sets. So, as a rule of thumb, we use 60 percent in the training set, 20 percent in the test set, and 20 percent in the validation set.

Okay, so then we have to run this cell. Here, you can see that it takes about eight minutes, so I'll just skip this because I have already run it. So when you run this cell, what it will do is activate the graph event, right? This is a comment, actually. So this is an example of a job submission step; if you wanted to submit it, we have to do it like this. If you go down a little bit, this is what we are going to do, actually. So, we import all these important modules that are needed, since it is using PyTorch as well.

So, we import the torch you can see here, and then we need to define what we want to do with this job, actually. So, the data set we are using again is GDB 13 1k debug, which is already given in the data/pre-training folder in this GitHub repo. which we have

downloaded or, you know, cloned from the GitHub repo of the Graph Invent And then in the next thing, the job type we are picking up training here, there are multiple options you can pick, like pre-process aware; for example, you have a data set. So, we wanted to pre-process it because you are starting with new data, as this is an already prepared data set, GDB 13 1 K, by the molecular AI team and the graph invent team. But if we wanted to prepare our own data set, we have to make sure that we do the preprocessing.

In preprocessing, what it will do is filter out those molecules based on our requirements, and then it will convert them into graphs and use them. So, we can use it as training. So, now, we are training a model, which means this model will learn how the graph and the molecules are composed of by using the graph neural net. We can also generate it if we put it, so it will generate new molecules that we will do in the next step once we train a model. So, once we have a model, in the next step we will generate molecules using that model.

And then you can put a test as well, where you can just evaluate your model, actually. So, the job directory start index indicates where to start the indexing job directory, and the number of jobs to run per model is given as one. And then whether or not this is a restart job is false because we are just, you know, starting it for the first time. You can keep it true if you are continuing with the earlier run job, actually. and then false override override job directories which already exist give it true And then you know the job name is "example job name," used to create a subdirectory.

We can give it some name as well, like "gdb 13 1k training" or something like that. Since we are running on, you know, the Google call apps, we are using the cloud computing from Google. But what we can do is, if you are running it on a cluster, actually, since, as I said, it takes a lot of computational power. So, it is always wise to run it on clusters, whether it is a GPU cluster or a CPU cluster. So, in clusters, we usually have this job manager, and that is one of the popular ones, Slurm.

So, Slurm is a job manager that is used for managing jobs on a cluster, so if you are running it using Slurm, then you need to specify these details. As well as how much the required RAM will be in GB, how much the runtime will be, and whether it is using Slurm or Node. Okay, and for a Slurm job, you need to set the partitions, etc.

You have to set the paths here. OK. So, all these details you actually need to set. The next code block is showing the defined data set specific parameters. So, now we are talking about the parameters of those molecules that are in the data set, actually.

OK. So, the atom types. So, as we saw in the table, we have the GDB 13 1 K. Data contains, you know, carbon, nitrogen, oxygen, sulfur, and chlorine, okay. So, this is what we have

to give: formal charge minus 1, 0, plus 1; maximum n nodes 13 means it can only contain molecules having less than or equal to 13 atoms. And then, job type, job underscore type, which you have already given, the data set directory, and then restart or not if it is broken somewhere. And then model GGN, and so you can see here now we are using GGN, and there are other models.

As we talked about earlier, there are six different models, so we can choose among those models as well. Okay, and then some more details. Okay, so once we have n samples, which is one hundred, these additional parameters can be defined if different from the default. For instance, for generating jobs, don't forget to specify generation epochs and the n samples.

Ok, and then we create an output directory. So here we have created an output directory, and after that, we submit the jobs. So this is especially for the Slurm, actually running it on the server or the cluster, and then finally, we run these cells. Once you run this cell, you will see something like this. So, it creates a dataset directory, blah blah blah, and then you know generating 90, generating batch 1 of 2, generating 90 molecules in this many seconds, and then you know it has several epochs; several times it will train the model. Right, and then finally you will get something like, "Yeah, so you will say I get something like this float.

" So once the training is done, once you see a green tick on the cell, you will see that the cell has run successfully and has actually completed. So now our model has actually been trained.

Right. The next step is to generate the molecules. Right. The next step is to generate the molecules. OK. In this cell, what we are going to do is generate new molecules from the trained model. And it is exactly the same as the training cell as the earlier cell where we trained the model, so the only difference is—let me show you—the only difference is here instead of using the train. So what we are doing is we are just asking it to generate, okay? So here we are taking this generate command, putting the generate command here, which will generate the molecules by using the model that we have trained in the earlier cell, okay.

So, again it will take about 8 minutes. Therefore, I will not run. So, just click on this cell; it will run it, and then finally, what you will see here is in the end, and one more thing I just wanted to highlight. So, one more thing: when it is generating the molecules, these will actually be the limits. For example, it can only generate molecules containing carbon, nitrogen, oxygen, sulfur, and chlorine. Having charges of minus 1, 0, and plus 1, and a maximum load of 13, these limitations were put in place while training the data as well. It

is something like this: if you are training a child in the English language and you have just given him 100 words, make him learn those 100 words.

So, now you ask him or her to generate sentences from those words that he or she knows. So, he or she is only allowed to use those 100 words. He or she cannot, you know, go beyond those 100 words, actually. Something like that is here. We have given those, you know, those atoms that these are the only atoms which you can use.

So, these can be the only charges: -1, 0, and +1, which you can use, and the maximum number of atoms one molecule can have is 13. So, then it will generate only within these limits actually. And you cannot change this limit as well because you have trained the model like that. So, if the model is trained on carbon, nitrogen, oxygen, sulfur, and chlorine only, it cannot generate a molecule having a bromine atom, nor can it generate a molecule having a formal charge of plus 2 or minus 2. So, once you run this, what you will see here is that it will generate the molecules, right? So, you can see here that 50 molecules were generated in 0 seconds, then 53, then 52; likewise, it will show you how this molecule was generated.

Okay, so if you remember, one of the challenges with generative modeling was that the generated molecules are usually not, you know, valid. And that is why we talked about a lot of benchmarking of those tools as well, like how to benchmark and what those methods are to benchmark those tools. So, once we have done it, we now have to see whether those molecules are, you know, valid or not. So, we will filter them out.

We will just pre-process here in this cell; we can just click run this cell. I cannot run it because I have not run the previous cells, so it will show some errors here. So, you can run this cell and then it will run. So, what you are doing here is just concatenating all those epochs in which molecules are generated. So, for example, if there are 10 epochs. Club all those outputs from those 10 epochs, actually, and then we will filter them out.

So, we will filter out the invalid entries. So, by default, Graph Invent writes an XC placeholder when an invalid molecule graph is generated, as an invalid molecular graph cannot be converted to a SMILES string; for saving, the placeholder is used because the NLL is written. For all generated graphs in a separate file, the same line number in the dot NLL file corresponds to the same line number in the dot SMI file. So, similarly, if an empty graph samples an invalid action as the first action, then no smiles can be generated for an empty graph. So, the corresponding line for an empty graph in a SMILES file cannot contain only the IDs of the molecules. So, for visualization, we need to see only the valid molecular graphs that we are interested in.

So, the smiles for the generated molecules can thus be post-processed as follows to remove the empty and invalid entries from a file before visualization. So, then we will run this cell, so what it will do is here is it will you know remove the xe placeholder from the file okay because whenever the graph invent was you know generating an invalid graph so it put it it places an xe in the front of that actually so we'll just remove that that xe from the smiles and then we will also remove the empty so the second command here which we are using here in second command so what it is doing is it is removing the empty empty graphs from the files as well So, once we have done that, so then what we can do is we can download by running this cell and you can download the generated file which is the epoch2.smi file. So, you can download the epoch2.smi file by clicking on this cell and then we can also have a quick look at the you know the generated molecules.

So, if you if you just issue this more command so more and then epoch 2 dot smi. So, what you can see smile and then name. So, you can see this is a smile and then name 0, 2, 3, 5, 6, 7, 8, 11, 12, 15, 16, 18, 19, 20, 23, 25. So, you can see some of them are missing some of them are missing in between like 4 is missing because either 4 was an invalid smiles generated using the graph invent or it was you know it was an empty line actually. So, that has got removed and then you can see these are the only valid smiles which we have got from this generation actually.

So, now, we can see that these all are you know valid smiles. However, you will see for example, CCOC, CCOC 5 and 6 are similar right. So, we did not remove the duplicates. The next step is you can remove the duplicates as well and then you can see how many of them are you know valid and known duplicate smiles or the molecules which are generated in the you know by using this graph invent tool. And then further you can always you know use the RD kit for visualizing these molecules, these generated smiles into either 2D or even you can use like open bsael or you know other tools to convert them into 3D and then you can visualize their structures as well. And some of you might be wondering that why it has only created like generated a molecule with just one carbon atom or just NO or CO.

So, the you know we have given it very small data set. So, we just trained our model on you know 1K molecule, 1000 molecules. So, it is not able to learn all these you know nitty-gritty details of those molecules you know trained it on large data set. So, we can generate even complex molecules with this tool and that is the beauty of this tool you know ok. Another important thing is that whenever we are running it, whenever we are doing it. The generation of molecules always depends on the kind of a type of training data set we have used.

And we can also use active learning in it as well like where we have for example, some known molecules, known active molecules against some target. So, we can use those active

molecules and we can supply them and then we can ask the model to generate molecules like that. We can just train it to a large large number of molecules to learn how molecules are made and then we can ask it to generate molecules similar to our active data sets. So, that that is also a possibility yeah.

So, that was you know the that is the end of this tutorial in this hands on session. In this session we have seen how we can use the graph invent to you know to generate to train a model and then to use that model to generate kind of a small molecules based on our requirements. and then you can always go to this github repo actually so you just click on this github repo click on the github repo so it will app open up here the molecular ai graph invent and then you can always go through this and then you can because since they are keeping it updated as well so you can always go to this and then you can get all these details and then they have you know the sample the examples as well um so which is already pre processed and then also you can always you know you can you can click on the tutorials and then you can get all these you know how to do the transfer learning or you do the benchmarking with MOSES as well.

So, that I forgot to tell that the models, the model when they have developed and when they have validated it. So, they use the benchmarking tool MOSES which we discussed in our earlier lecture as well. So, you can also benchmark your model generation using the MOSES and then you can if you want to pre process large data sets like millions of molecules, then you can use use this as well this details and then you can do the reinforcement learning or you can do the transfer learning by by using this graph invent tool.

Okay, so that was all. So, with that, I hope you enjoyed this hands-on tutorial. So, you just open that link and that whole app notebook is always accessible to you. So, you can anytime you can open that, you can run that and you can, you know, play with it and learn how to use the graph invent or the de novo generative drug design using graph invent tool. With that, thank you.