Monday, April 27th 2020

Yash Gandhi
Lenny Fobe

Final Report: Boltzmann Generators

Final State of Project:

In the final code sprint for our implementation of Boltzmann generators we were able to complete a sizable part of the initial goals of the project. In our code's final state, it is able to simulate 3 toy molecular dynamics (MD) simulations and train Boltzmann generators using a variety of loss functions on short MD simulations. The Boltzmann generator training process can be monitored using TensorBoard and final trained models can be logged and hosted online using the Streamlit app builder. Informative output can be generated, such as loss graphs, generated sample distributions and free energy profiles as the Botlzman generator is trained. Most behaviors of this program are manifested as interactions between objects.

While getting the molecular simulation and machine learning parts of the program working together in this final code sprint, the construction of the various loss functions used in the paper required a closer relationship between the SimulationDate object and the Model objects. Due to the need to calculate energy of generated configurations on the fly while training, we were only able to train Boltzman generators for systems we simulated in our simulation_library. If we wanted to evaluate configurations from a real MD engine, like Gromacs, we would have to build an adapter which would wrap Gromacs and return values for the common interfaces needed to interact with our program. In the time left, we were not able to complete this task, and focused more energy on completing goals on the other toy systems. Also, because of time constraints, we were not able to fully train a RealNVP model on the dimer simulation. This simulation requires many more samples, has many higher dimensions (72 dimensions versus 2 dimensions for other simulations), and requires a significant search on the hyperparameter space, but the code does allow for this type of model to be trained.

Final Class Diagram:

Our final class diagram is attached below, along with previous iterations of our class diagram. We also added a class diagram for the facade pattern we used for the simulation objects used in this project.

We used a number of object oriented design principles and patterns in our project. Some of these include:

1.) We used inheritance in a number of ways. There were multiple levels of inheritance when building the neural network because the RealNVP network consists of multiple layers each with their own set of neural network blocks. Each section - the neural network block, the single RealNVP layer, and the RealNVP model - inherit from base tensorflow classes and then uses references to build a working network.
2.) When designing the components for tensorboard, it became clear that not all types of logging were necessary for all types of training. Therefore, we built the tensorboard logging as a decorator pattern. This way the user can decide the different types of logging they would like to use to keep track of training the model.
3.) As a complement to the logging, we implemented a subject/observer pattern to update the tensorboard at every training iteration. Once a single training iteration happened, the network

called update on the tensorboard and the tensorboard got a dictionary of values to plot. With an internal processing of each value, it appropriately plots the results from that iteration.

4.) For our model, we are often changing the loss and optimizer functions to test new methods for different data sets. Therefore, we implemented a strategy pattern for both the loss and optimizers. This allows us to easily add new loss functions and optimizers without worrying about the repercussions on the network because they are loosely coupled. We have 4 different loss functions, two of which inherit from one of the loss functions and 3 optimization functions (although we did find that RMSProp was often the only useful optimization function, we did not do a full search on the hyperparameters, so we left the other optimization functions).
5.) The Integrator, Thermostat, BoundaryCondition and Potential objects use the strategy pattern. Both have common interfaces which are used by the System and Integrator objects to easily swap between integrator and thermostat methods. In this project we have 4 integrator methods implemented and 1 thermostat implemented. With the use of the strategy pattern, more integrators and thermostats can easily be added to the program without having to rewrite other objects.
6.) The Thermostat, Integrator and System objects can all be built from simple factory patterns, where their construction is put into a dedicated class. This simplifies the instantiation of these classes to a simple passing of strings and keyword arguments.
7.) The Logger objects used for recording data from the System class were modeled after a Subject/Observer pattern. The concrete observers (CoordinateLogger, EnergyLogger, DistanceLogger, etc.) all observe the System and request pertinent data from the system after a certain number of simulation steps.

Class Diagram Comparison:

At the beginning of the project, we already had an idea about what types of classes and the overall structure we would need. Our early class diagram, while simplistic, did provide a good general overview of the project. The objects responsible for simulations, model parameters, and data output were all clearly segregated into various objects that interacted. As the development progressed, the intricacies of the project revealed themselves and we slowly incorporated more and more detailed class structure into our diagram. This included all of the sub-modules that the simulation objects used and the number of different loss functions and possible logging techniques we would use to track the model during training. Now, we have included a number of different, and more detailed, classes into our diagram and the connections between different classes and sections of the program have become more defined, as we implemented them.

Third-Party Code Statement:

Python has a host of libraries and API to extend different programs. Our project uses a number of these packages especially for scientific calculations and neural network training. We use the following pacakges throughout our code:

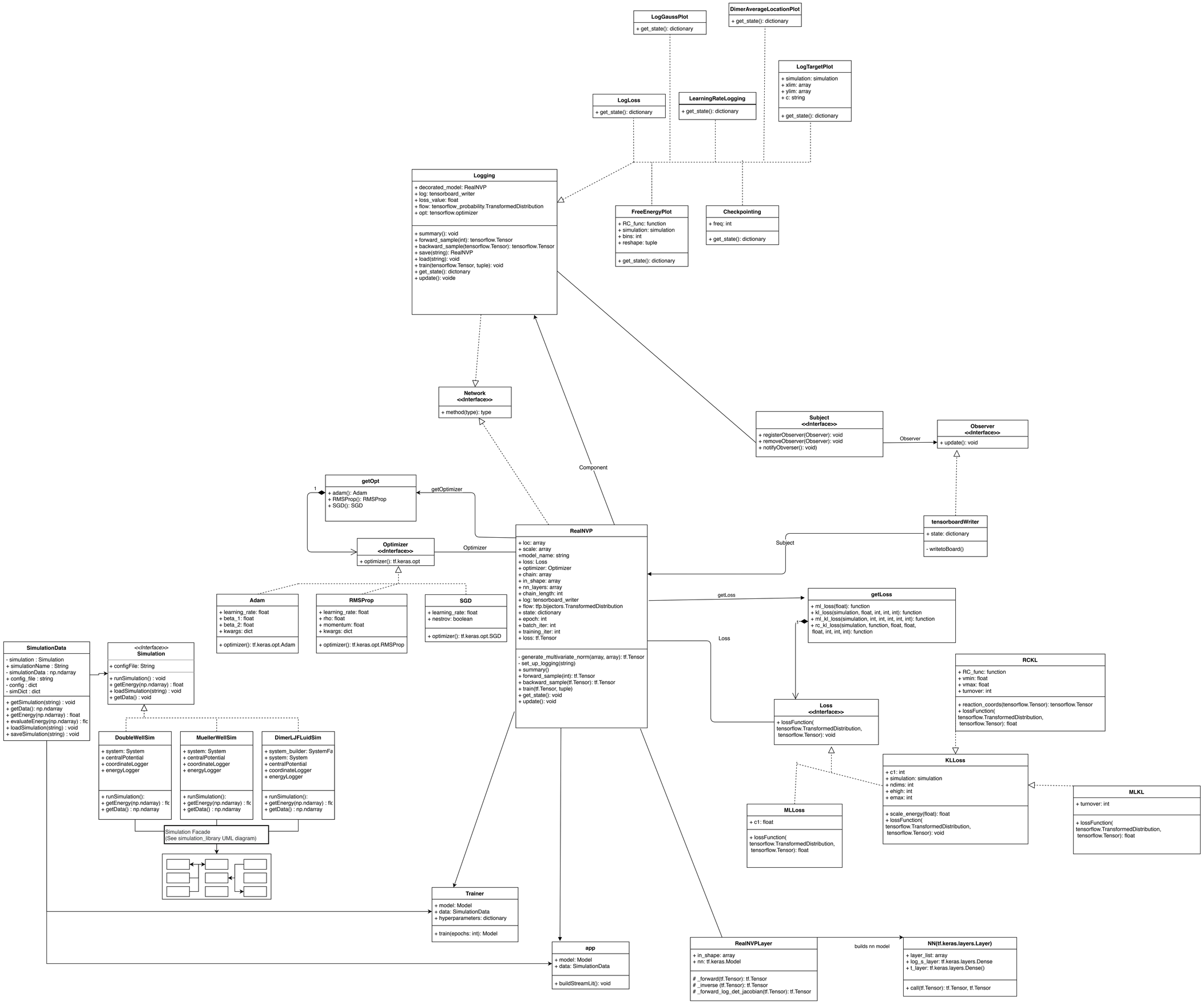
1.) tensorflow == 2.1.0
2.) tensorflow-probability == 0.8.0
3.) numpy == 1.17.0
4.) streamlit == 0.57.3

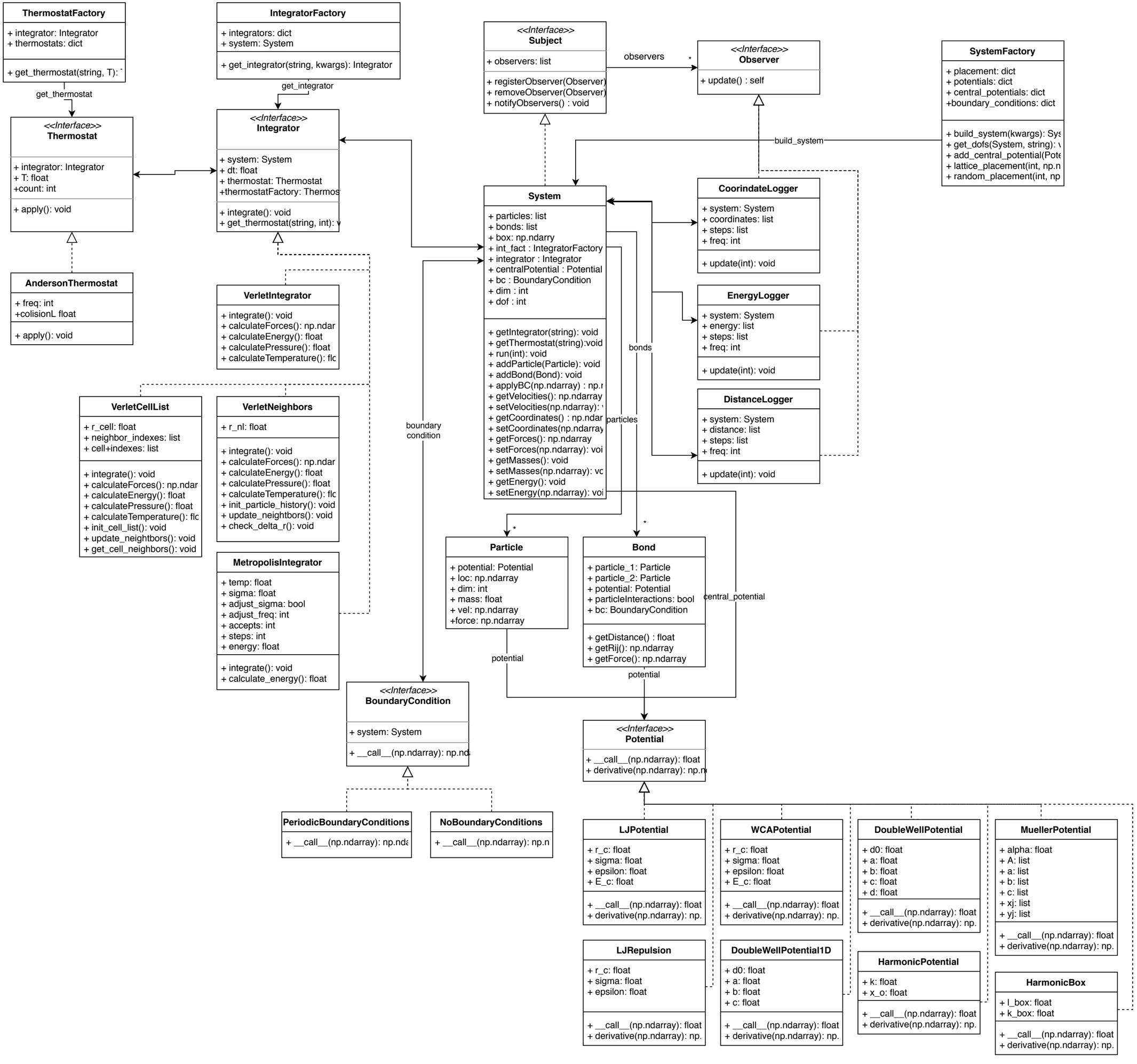
5.) pyyaml == 5.1.2
6.) scipy == 1.4.1
7.) matplotlib == 3.1.1
8.) Pillow == 6.2.0

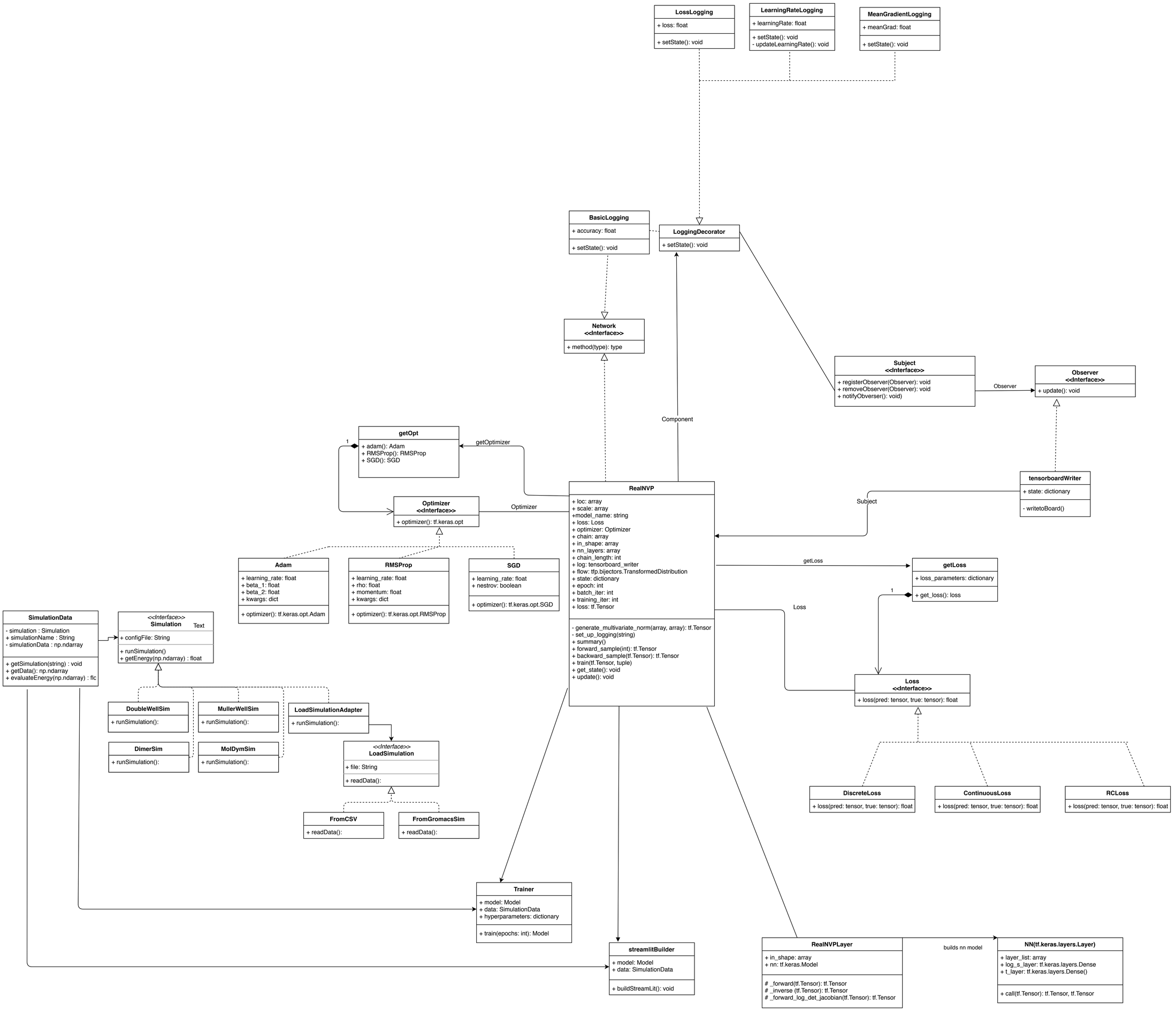
In addition to these packages, we use examples from RealNVP tutorials to build our architecture and get the functions for generating the double moon distribution. The link to the specific tutorial is within the network_base.py file. The simulation library we used in this program was in part developed for Computational Statistical Physics (PHYS7810) and modified and expanded for the purposes of this project. All other code is original code built for this project.

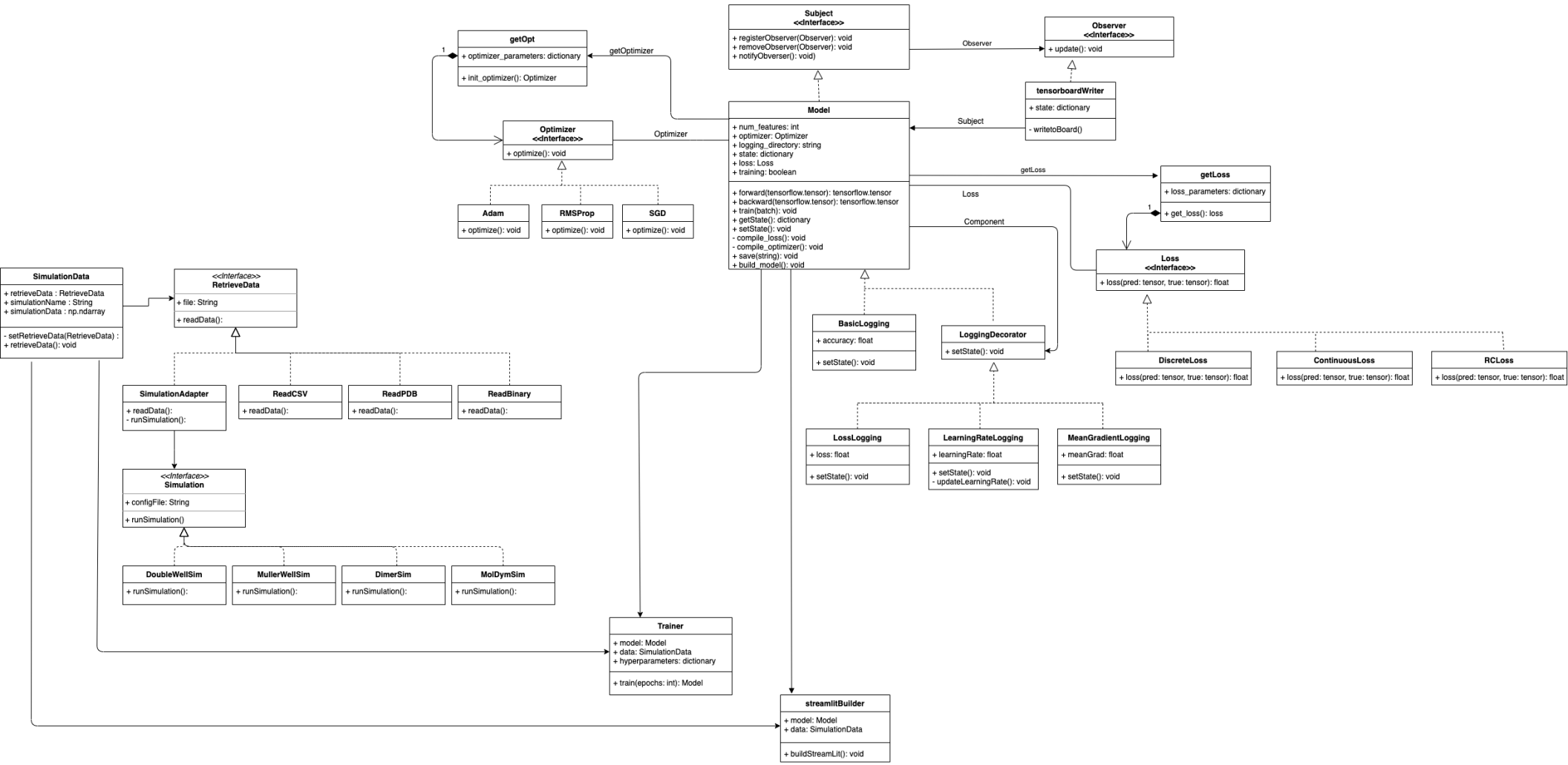
OOAD Process:

1. Before we became fully invested in this project, we spent time researching RealNVP networks, neural network normalizing flows, molecular dynamics, and other related topics to better understand the underlying equations and algorithms that we were going to be implementing. This analysis of the field gave us guidelines for how the code should behave if it had been designed correctly. By identifying the relevant components that drive the research, we were able to conceptualize the problems within the research and formalize the methods that we would need to develop to build a software that could address those problems.
2. We used Trello, a Kanban-like workflow manager, to delegate tasks and notify each other when coding tasks were completed. This development tool ensured we were both on the same page when meeting to discuss code.
3. The required UML diagrams in project 4 greatly helped us organize the class structure and data responsibility of our program early on in the development process. Building the sequence diagrams had us think about the order of interactions between high level objects. The activity and use case diagrams gave us general tasks we wanted our system to be able to accomplish and how users would interact with our code, well before any coding was done. The architecture diagram gave us an initial view as to how data would be segregated between various levels of abstraction.
4. The multiple planning and coding sprints we did throughout projects 4, 5 and 6, resulted in a project on which we had iterated on multiple ideas. This iterative planning resulted in some improvements from coding spring to coding sprint, as we encountered problems along the way. Notably, the initial independence of simulation logic and neural network logic was scrapped as we learned the loss functions in the paper required energy calculation on the fly of generated configurations.


LogGaussPlot
+ get_state(): dictionary
DimerAverageLocationPlot
+ get_state(): dictionary
LogTargetPlot
+ simulation: simulation
+ xlim: array
+ ylim: array
+ c: string
+ get_state(): dictionary
LogLoss
+ get_state(): dictionary
LearningRateLogging
+ get_state(): dictionary
Logging
+ decorated_model: RealNVP
+ log: tensorboard_writer
+ loss_value: float
+ flow: tensorflow._probability.TransformedDistribution
+ opt: tensorflow.optimizer
+ summary(): void
+ forward_sample(int): tensorflow.Tensor
+ backward_sample(tensorflow.Tensor): tensorflow.Tensor
+ save(string): RealNVP
+ load(string): void
+ train(tensorflow.Tensor, tuple): void
+ get_state(): dictonary
+ update(): voide
FreeEnergyPlot
+ RC_func: function
+ simulation: simulation
+ bins: int
+ reshape: tuple
+ get_state(): dictionary
Checkpointing
+ freq: int
+ get_state(): dictionary
Network
<<Interface>>
+ method(type): type
Component
Subject
<<Interface>>
+ registerObserver(Observer): void
+ removeObserver(Observer): void
+ notifyObverser(): void)
Observer
Observer
<<Interface>>
+ update(): void
getOpt
+ adam(): Adam
+ RMSProp(): RMSProp
+ SGD(): SGD
1
getOptimizer
Optimizer
<<Interface>>
+ optimizer(): tf.keras.opt
Optimizer
RealNVP
+ loc: array
+ scale: array
+model_name: string
+ loss: Loss
+ optimizer: Optimizer
+ chain: array
+ in_shape: array
+ nn_layers: array
+ chain_length: int
+ log: tensorboard_writer
+ flow: tfp.bijectors.TransformedDistribution
+ state: dictionary
+ epoch: int
+ batch_iter: int
+ training_iter: int
+ loss: tf.Tensor
- generate_multivariate_norm(array, array): tf.Tensor
- set_up_logging(string)
+ summary()
+ forward_sample(int): tf.Tensor
+ backward_sample(tf.Tensor): tf.Tensor
+ train(tf.Tensor, tuple)
+ get_state(): void
+ update(): void
tensorboardWriter
+ state: dictionary
- writetoBoard()
Subject
getLoss
getLoss
+ ml_loss(float): function
+ kl_loss(simulation, float, int, int, int): function
+ ml_kl_loss(simulation, int, int, int, int, int): function
+ rc_kl_loss(simulation, function, float, float,
float, int, int, int): function
Loss
Adam
+ learning_rate: float
+ beta_1: float
+ beta_2: float
+ kwargs: dict
+ optimizer(): tf.keras.opt.Adam
RMSProp
+ learning_rate: float
+ rho: float
+ momentum: float
+ kwargs: dict
+ optimizer(): tf.keras.opt.RMSProp
SGD
+ learning_rate: float
+ nestrov: boolean
+ optimizer(): tf.keras.opt.SGD
SimulationData
- simulation : Simulation
+ simulationName : String
- simulationData : np.ndarray
+ config_file : string
- config : dict
- simDict : dict
+ getSimulation(string) : void
+ getData(): np.ndarray
+ getEnergy(np.ndarray) : float
+ evaluateEnergy(np.ndarray) : flo
+ loadSimulation(string) : void
+ saveSimulation(string) : void
<<Interface>>
Simulation
+ configFile: String
+ runSimulation() : void
+ getEnergy(np.ndarray) : float
+ loadSimulation(string) : void
+ getData() : void
DoubleWellSim
+ system: System
+ centralPotential
+ coordinateLogger
+ energyLogger
+ runSimulation():
+ getEnergy(np.ndarray) : flo
+ getData() : np.ndarray
MuellerWellSim
+ system: System
+ centralPotential
+ coordinateLogger
+ energyLogger
+ runSimulation():
+ getEnergy(np.ndarray) : flo
+ getData() : np.ndarray
DimerLJFLuidSim
+ system_builder: SystemFa
+ system: System
+ centralPotential
+ coordinateLogger
+ energyLogger
+ runSimulation():
+ getEnergy(np.ndarray) : flo
+ getData() : np.ndarray
Simulation Facade
(See simulation_library UML diagram)
Loss
<<Interface>>
+ lossFunction(
tensorflow.TransformedDistribution,
tensorflow.Tensor): void
RCKL
+ RC_func: function
+ vmin: float
+ vmax: float
+ turnover: int
+ reaction_coords(tensorflow.Tensor): tensorflow.Tensor
+ lossFunction(
tensorflow.TransformedDistribution,
tensorflow.Tensor): float
KLLoss
+ c1: int
+ simulation: simulation
+ ndims: int
+ ehigh: int
+ emax: int
+ scale_energy(float): float
+ lossFunction(
tensorflow.TransformedDistribution,
tensorflow.Tensor): void
MLLoss
+ c1: float
+ lossFunction(
tensorflow.TransformedDistribution,
tensorflow.Tensor): float
MLKL
+ turnover: int
+ lossFunction(
tensorflow.TransformedDistribution,
tensorflow.Tensor): float
Trainer
+ model: Model
+ data: SimulationData
+ hyperparameters: dictionary
+ train(epochs: int): Model
app
+ model: Model
+ data: SimulationData
+ buildStreamLit(): void
RealNVPLayer
+ in_shape: array
+ nn: tf.keras.Model
# _forward(tf.Tensor): tf.Tensor
# _inverse (tf.Tensor): tf.Tensor
# _forward_log_det_jacobian(tf.Tensor): tf.Tensor
builds nn model
NN(tf.keras.layers.Layer)
+ layer_list: array
+ log_s_layer: tf.keras.layers.Dense
+ t_layer: tf.keras.layers.Dense()
+ call(tf.Tensor): tf.Tensor, tf.Tensor


ThermostatFactory
+ integrator: Integrator
+ thermostats: dict
+ get_thermostat(string, T):
get_thermostat
IntegratorFactory
+ integrators: dict
+ system: System
+ get_integrator(string, kwargs): Integrator
get_integrator
<<Interface>>
Subject
+ observers: list
+ registerObserver(Observer)
+ removeObserver(Observer)
+ notifyObservers() : void
observers
*
<<Interface>>
Observer
+ update() : self
SystemFactory
+ placement: dict
+ potentials: dict
+ central_potentials: dict
+boundary_conditions: dict
+ build_system(kwargs): Sys
+ get_dofs(System, string): \
+ add_central_potential(Pote
+ lattice_placement(int, np.n
+ random_placement(int, np
build_system
<<Interface>>
Thermostat
+ integrator: Integrator
+ T: float
+count: int
+ apply(): void
<<Interface>>
Integrator
+ system: System
+ dt: float
+ thermostat: Thermostat
+thermostatFactory: Thermos
+ integrate(): void
+ get_thermostat(string, int): v
System
+ particles: list
+ bonds: list
+ box: np.ndarry
+ int_fact : IntegratorFactory
+ integrator : Integrator
+ centralPotential : Potential
+ bc : BoundaryCondition
+ dim : int
+ dof : int
+ getIntegrator(string): void
+ getThermostat(string):void
+ run(int): void
+ addParticle(Particle): void
+ addBond(Bond): void
+ applyBC(np.ndarray) : np.r
+ getVelocities(): np.ndarray
+ setVelocities(np.ndarray): \
+ getCoordinates() : np.ndar
+ setCoordinates(np.ndarray
+ getForces(): np.ndarray
+ setForces(np.ndarray): voi
+ getMasses(): void
+ setMasses(np.ndarray): vc
+ getEnergy(): void
+ setEnergy(np.ndarray): voi
CoorindateLogger
+ system: System
+ coordinates: list
+ steps: list
+ freq: int
+ update(int): void
EnergyLogger
+ system: System
+ energy: list
+ steps: list
+ freq: int
+ update(int): void
DistanceLogger
+ system: System
+ distance: list
+ steps: list
+ freq: int
+ update(int): void
bonds
particles
boundary condition
AndersonThermostat
+ freq: int
+colisionL float
+ apply(): void
VerletIntegrator
+ integrate(): void
+ calculateForces(): np.ndar
+ calculateEnergy(): float
+ calculatePressure(): float
+ calculateTemperature(): flc
VerletCellList
+ r_cell: float
+ neighbor_indexes: list
+ cell+indexes: list
+ integrate(): void
+ calculateForces(): np.ndar
+ calculateEnergy(): float
+ calculatePressure(): float
+ calculateTemperature(): flc
+ init_cell_list(): void
+ update_neightbors(): void
+ get_cell_neighbors(): void
VerletNeighbors
+ r_nl: float
+ integrate(): void
+ calculateForces(): np.ndar
+ calculateEnergy(): float
+ calculatePressure(): float
+ calculateTemperature(): flc
+ init_particle_history(): void
+ update_neightbors(): void
+ check_delta_r(): void
MetropolisIntegrator
+ temp: float
+ sigma: float
+ adjust_sigma: bool
+ adjust_freq: int
+ accepts: int
+ steps: int
+ energy: float
+ integrate(): void
+ calculate_energy(): float
Particle
+ potential: Potential
+ loc: np.ndarray
+ dim: int
+ mass: float
+ vel: np.ndarray
+force: np.ndarray
Bond
+ particle_1: Particle
+ particle_2: Particle
+ potential: Potential
+ particleInteractions: bool
+ bc: BoundaryCondition
+ getDistance() : float
+ getRij(): np.ndarray
+ getForce(): np.ndarray
central_potential
potential
potential
<<Interface>>
BoundaryCondition
+ system: System
+ __call__(np.ndarray): np.nd
PeriodicBoundaryConditions
+ __call__(np.ndarray): np.nda
NoBoundaryConditions
+ __call__(np.ndarray): np.n
<<Interface>>
Potential
+ __call__(np.ndarray): float
+ derivative(np.ndarray): np.n
LJPotential
+ r_c: float
+ sigma: float
+ epsilon: float
+ E_c: float
+ __call__(np.ndarray): float
+ derivative(np.ndarray): np.
WCAPotential
+ r_c: float
+ sigma: float
+ epsilon: float
+ E_c: float
+ __call__(np.ndarray): float
+ derivative(np.ndarray): np.
DoubleWellPotential
+ d0: float
+ a: float
+ b: float
+ c: float
+ d: float
+ __call__(np.ndarray): float
+ derivative(np.ndarray): np.
MuellerPotential
+ alpha: float
+ A: list
+ a: list
+ b: list
+ c: list
+ xj: list
+ yj: list
+ __call__(np.ndarray): float
+ derivative(np.ndarray): np.
LJRepulsion
+ r_c: float
+ sigma: float
+ epsilon: float
+ __call__(np.ndarray): float
+ derivative(np.ndarray): np.
DoubleWellPotential1D
+ d0: float
+ a: float
+ b: float
+ c: float
+ __call__(np.ndarray): float
+ derivative(np.ndarray): np.
HarmonicPotential
+ k: float
+ x_o: float
+ __call__(np.ndarray): float
+ derivative(np.ndarray): np.
HarmonicBox
+ l_box: float
+ k_box: float
+ __call__(np.ndarray): float
+ derivative(np.ndarray): np.


LossLogging
+ loss: float
+ setState(): void
LearningRateLogging
+ learningRate: float
+ setState(): void
- updateLearningRate(): void
MeanGradientLogging
+ meanGrad: float
+ setState(): void
BasicLogging
+ accuracy: float
+ setState(): void
LoggingDecorator
+ setState(): void
Network
<<Interface>>
+ method(type): type
Subject
<<Interface>>
+ registerObserver(Observer): void
+ removeObserver(Observer): void
+ notifyObverser(): void)
Observer
Observer
<<Interface>>
+ update(): void
Component
getOpt
+ adam(): Adam
+ RMSProp(): RMSProp
+ SGD(): SGD
1
getOptimizer
RealNVP
+ loc: array
+ scale: array
+model_name: string
+ loss: Loss
+ optimizer: Optimizer
+ chain: array
+ in_shape: array
+ nn_layers: array
+ chain_length: int
+ log: tensorboard_writer
+ flow: tfp.bijectors.TransformedDistribution
+ state: dictionary
+ epoch: int
+ batch_iter: int
+ training_iter: int
+ loss: tf.Tensor
- generate_multivariate_norm(array, array): tf.Tensor
- set_up_logging(string)
+ summary()
+ forward_sample(int): tf.Tensor
+ backward_sample(tf.Tensor): tf.Tensor
+ train(tf.Tensor, tuple)
+ get_state(): void
+ update(): void
tensorboardWriter
+ state: dictionary
- writetoBoard()
Subject
Optimizer
<<Interface>>
+ optimizer(): tf.keras.opt
Optimizer
Adam
+ learning_rate: float
+ beta_1: float
+ beta_2: float
+ kwargs: dict
+ optimizer(): tf.keras.opt.Adam
RMSProp
+ learning_rate: float
+ rho: float
+ momentum: float
+ kwargs: dict
+ optimizer(): tf.keras.opt.RMSProp
SGD
+ learning_rate: float
+ nestrov: boolean
+ optimizer(): tf.keras.opt.SGD
getLoss
getLoss
+ loss_parameters: dictionary
+ get_loss(): loss
1
Loss
Loss
<<Interface>>
+ loss(pred: tensor, true: tensor): float
SimulationData
- simulation : Simulation
+ simulationName : String
- simulationData : np.ndarray
+ getSimulation(string) : void
+ getData(): np.ndarray
+ evaluateEnergy(np.ndarray) : flc
<<Interface>>
Simulation
Text
+ configFile: String
+ runSimulation()
+ getEnergy(np.ndarray) : float
DoubleWellSim
+ runSimulation():
MullerWellSim
+ runSimulation():
LoadSimulationAdapter
+ runSimulation():
DimerSim
+ runSimulation():
MolDymSim
+ runSimulation():
<<Interface>>
LoadSimulation
+ file: String
+ readData():
FromCSV
+ readData():
FromGromacsSim
+ readData():
DiscreteLoss
+ loss(pred: tensor, true: tensor): float
ContinuousLoss
+ loss(pred: tensor, true: tensor): float
RCLoss
+ loss(pred: tensor, true: tensor): float
Trainer
+ model: Model
+ data: SimulationData
+ hyperparameters: dictionary
+ train(epochs: int): Model
streamlitBuilder
+ model: Model
+ data: SimulationData
+ buildStreamLit(): void
RealNVPLayer
+ in_shape: array
+ nn: tf.keras.Model
# _forward(tf.Tensor): tf.Tensor
# _inverse (tf.Tensor): tf.Tensor
# _forward_log_det_jacobian(tf.Tensor): tf.Tensor
builds nn model
NN(tf.keras.layers.Layer)
+ layer_list: array
+ log_s_layer: tf.keras.layers.Dense
+ t_layer: tf.keras.layers.Dense()
+ call(tf.Tensor): tf.Tensor, tf.Tensor


Subject
<<Interface>>
+ registerObserver(Observer): void
+ removeObserver(Observer): void
+ notifyObverser(): void)
Observer
Observer
<<Interface>>
+ update(): void
tensorboardWriter
+ state: dictionary
- writetoBoard()
Subject
getOpt
+ optimizer_parameters: dictionary
+ init_optimizer(): Optimizer
1
getOptimizer
Model
+ num_features: int
+ optimizer: Optimizer
+ logging_directory: string
+ state: dictionary
+ loss: Loss
+ training: boolean
+ forward(tensorflow.tensor): tensorflow.tensor
+ backward(tensorflow.tensor): tensorflow.tensor
+ train(batch): void
+ getState(): dictionary
+ setState(): void
- compile_loss(): void
- compile_optimizer(): void
+ save(string): void
+ build_model(): void
Optimizer
<<Interface>>
+ optimize(): void
Optimizer
Adam
+ optimize(): void
RMSProp
+ optimize(): void
SGD
+ optimize(): void
getLoss
getLoss
+ loss_parameters: dictionary
+ get_loss(): loss
1
Loss
Component
Loss
<<Interface>>
+ loss(pred: tensor, true: tensor): float
SimulationData
+ retrieveData : RetrieveData
+ simulationName : String
+ simulationData : np.ndarray
- setRetrieveData(RetrieveData) :
+ retrieveData(): void
<<Interface>>
RetrieveData
+ file: String
+ readData():
BasicLogging
+ accuracy: float
+ setState(): void
LoggingDecorator
+ setState(): void
DiscreteLoss
+ loss(pred: tensor, true: tensor): float
ContinuousLoss
+ loss(pred: tensor, true: tensor): float
RCLoss
+ loss(pred: tensor, true: tensor): float
SimulationAdapter
+ readData():
- runSimulation():
ReadCSV
+ readData():
ReadPDB
+ readData():
ReadBinary
+ readData():
LossLogging
+ loss: float
+ setState(): void
LearningRateLogging
+ learningRate: float
+ setState(): void
- updateLearningRate(): void
MeanGradientLogging
+ meanGrad: float
+ setState(): void
<<Interface>>
Simulation
+ configFile: String
+ runSimulation()
DoubleWellSim
+ runSimulation():
MullerWellSim
+ runSimulation():
DimerSim
+ runSimulation():
MolDymSim
+ runSimulation():
Trainer
+ model: Model
+ data: SimulationData
+ hyperparameters: dictionary
+ train(epochs: int): Model
streamlitBuilder
+ model: Model
+ data: SimulationData
+ buildStreamLit(): void