Modeling Chemical Reactions

Intern: Shaun Tan *(me)*

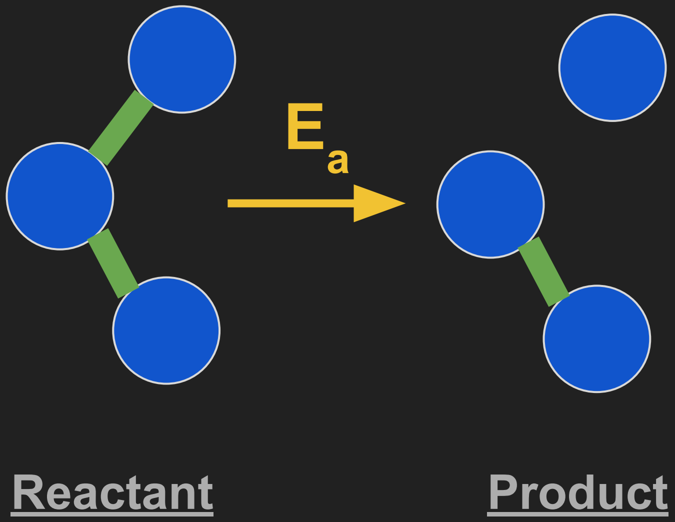
Supervisors:
Chieu Hai Leong, Chong Yihui, Alvin Liew

About me

- NUS, year 2 Computer Science student
- Came from Diploma of Applied Chemistry

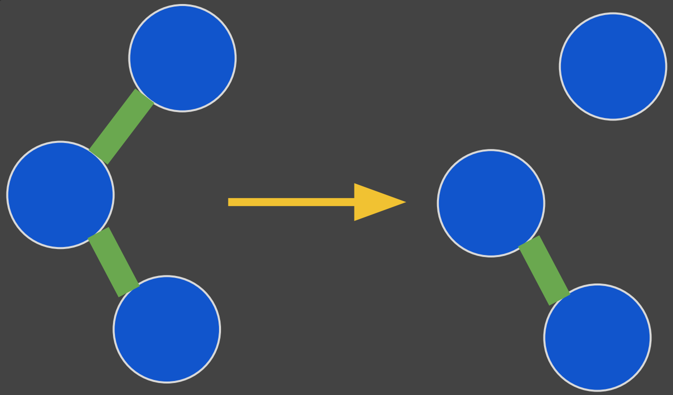
- Started internship with only introductory knowledge in neural networks
- Interested in doing **SWE** and/or **ML**

Objective



Predicting **activation energy** from the reaction's **molecules** *(ie. reactants & products)*

Objective

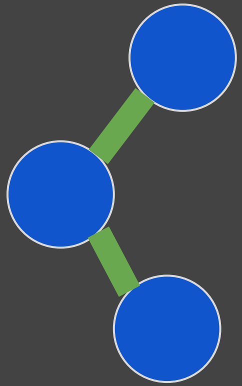


- Activation energy
- Rate of reaction
- Yield %

Reaction properties

- Multiple molecules
- Has reactants & products

VS



- Water solubility
- Toxicity

Molecular properties

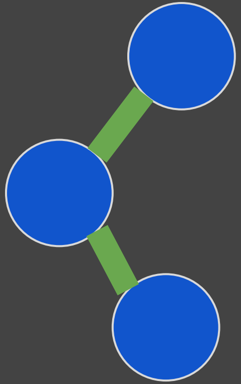
- 1 molecule only

Objective

Can we train on **molecular**-properties, then finetune on **reaction**-properties?

- **Reason:** there's a lot more molecular datasets than reaction datasets

Can a model for that's good at predicting molecular-property be **adapted** to predict reaction properties?



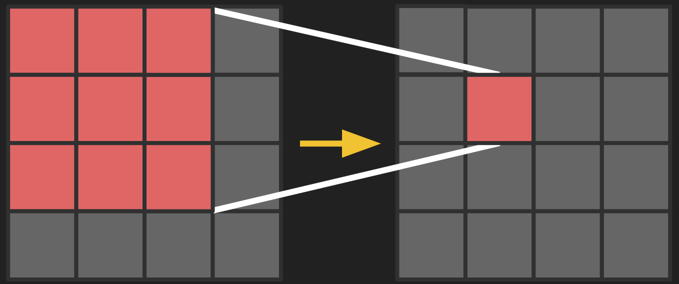
- Water solubility
- Toxicity

Molecular properties

- 1 molecule only

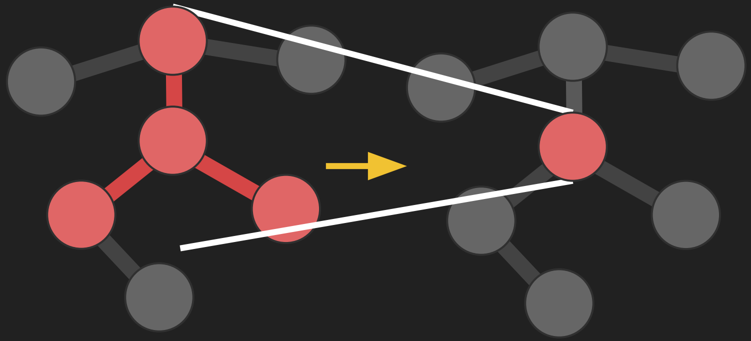
Context: Graph Neural Networks (GNN)

CNN



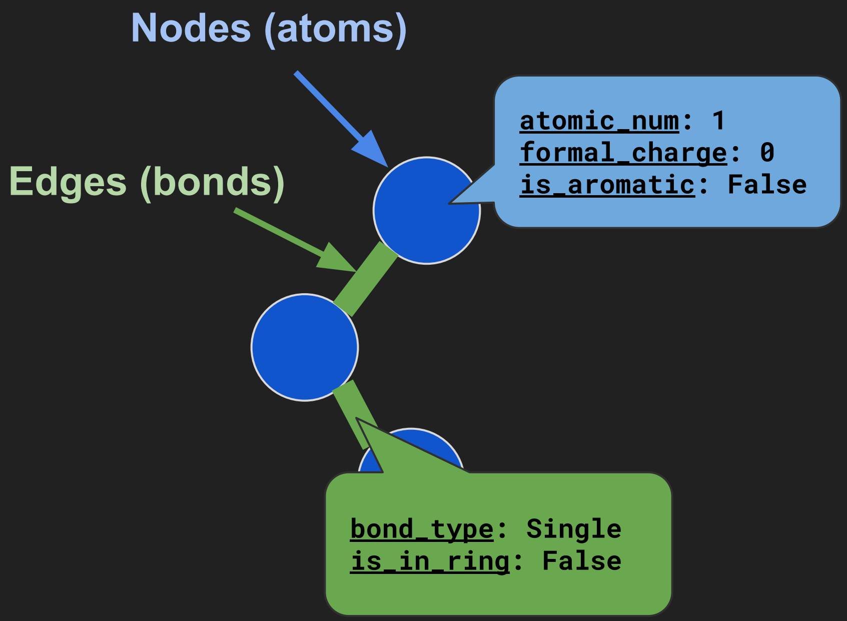
Like how **CNN** aggregate adjacent pixels in an image *(ie. convolving)*

GNN



GNN aggregate adjacent nodes & edges in a graph *(ie. message-passing)*

Context: GNN for molecules



Molecules can modelled as graphs

Using SOTA architectures

- We used **GeoGNN (aka GEM)**, a State-Of-The-Art (SOTA) architecture for predicting **molecular**-properties

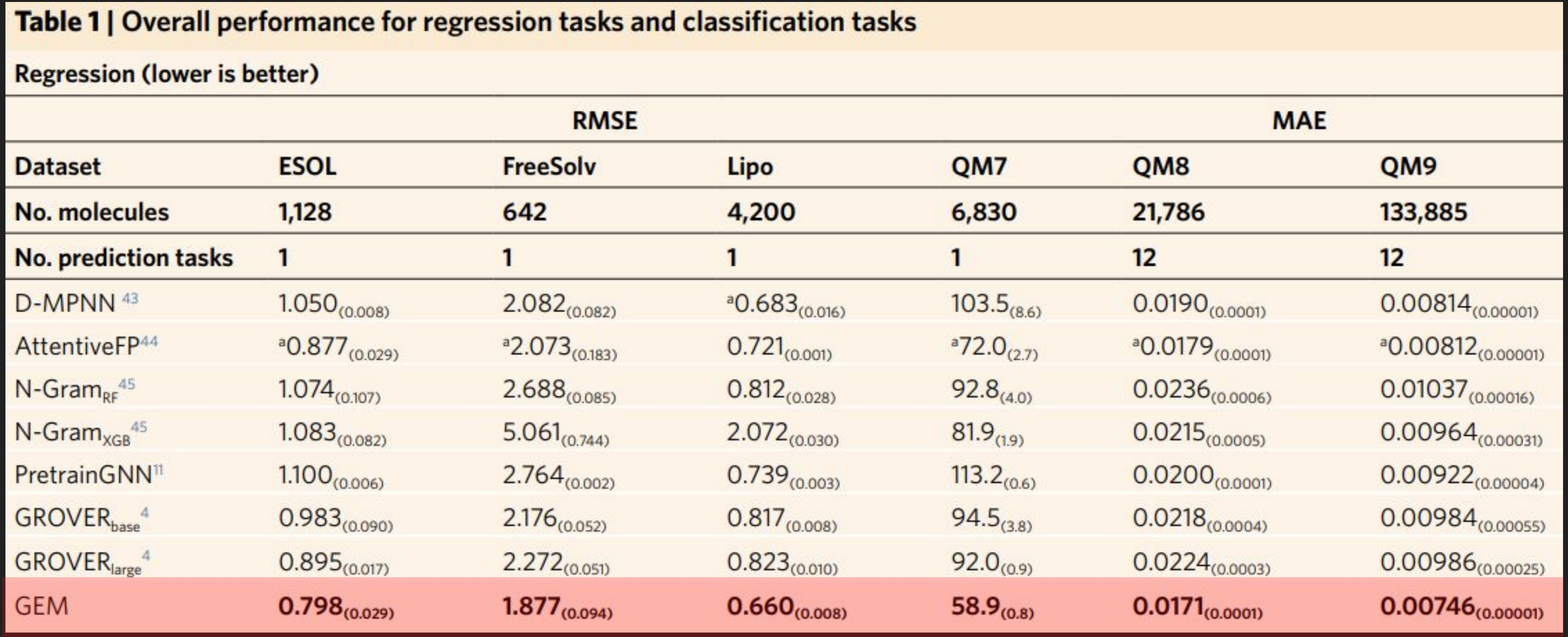


Table 1 | Overall performance for regression tasks and classification tasks

Regression (lower is better)

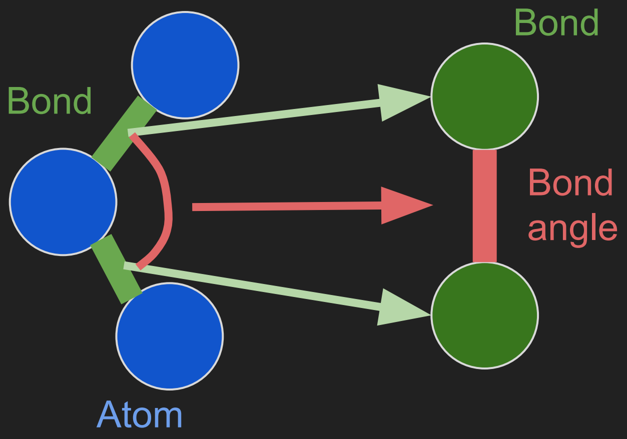
	RMSE				MAE	
Dataset	**ESOL**	**FreeSolv**	**Lipo**	**QM7**	**QM8**	**QM9**
No. molecules	**1,128**	**642**	**4,200**	**6,830**	**21,786**	**133,885**
No. prediction tasks	**1**	**1**	**1**	**1**	**12**	**12**
D-MPNN[43]	1.050$_{(0.008)}$	2.082$_{(0.082)}$	[a]0.683$_{(0.016)}$	103.5$_{(8.6)}$	0.0190$_{(0.0001)}$	0.00814$_{(0.00001)}$
AttentiveFP[44]	[a]0.877$_{(0.029)}$	[a]2.073$_{(0.183)}$	0.721$_{(0.001)}$	[a]72.0$_{(2.7)}$	[a]0.0179$_{(0.0001)}$	[a]0.00812$_{(0.00001)}$
N-Gram$_{RF}$[45]	1.074$_{(0.107)}$	2.688$_{(0.085)}$	0.812$_{(0.028)}$	92.8$_{(4.0)}$	0.0236$_{(0.0006)}$	0.01037$_{(0.00016)}$
N-Gram$_{XGB}$[45]	1.083$_{(0.082)}$	5.061$_{(0.744)}$	2.072$_{(0.030)}$	81.9$_{(1.9)}$	0.0215$_{(0.0005)}$	0.00964$_{(0.00031)}$
PretrainGNN[11]	1.100$_{(0.006)}$	2.764$_{(0.002)}$	0.739$_{(0.003)}$	113.2$_{(0.6)}$	0.0200$_{(0.0001)}$	0.00922$_{(0.00004)}$
GROVER$_{base}$[4]	0.983$_{(0.090)}$	2.176$_{(0.052)}$	0.817$_{(0.008)}$	94.5$_{(3.8)}$	0.0218$_{(0.0004)}$	0.00984$_{(0.00055)}$
GROVER$_{large}$[4]	0.895$_{(0.017)}$	2.272$_{(0.051)}$	0.823$_{(0.010)}$	92.0$_{(0.9)}$	0.0224$_{(0.0003)}$	0.00986$_{(0.00025)}$
GEM	**0.798$_{(0.029)}$**	**1.877$_{(0.094)}$**	**0.660$_{(0.008)}$**	**58.9$_{(0.8)}$**	**0.0171$_{(0.0001)}$**	**0.00746$_{(0.00001)}$**

(best)

Using SOTA architectures

Atom-bond graph

Bond-bondAngle graph



Bond

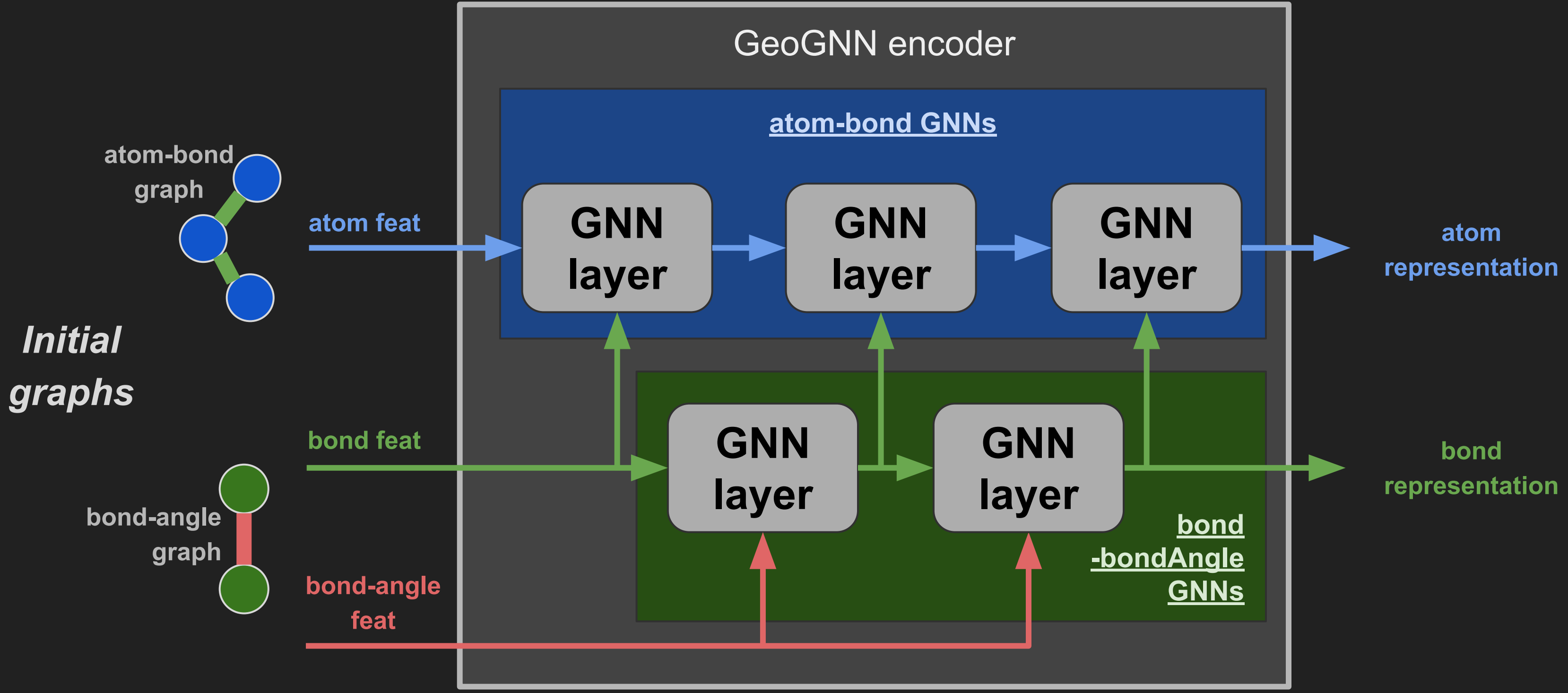
Bond

Bond angle

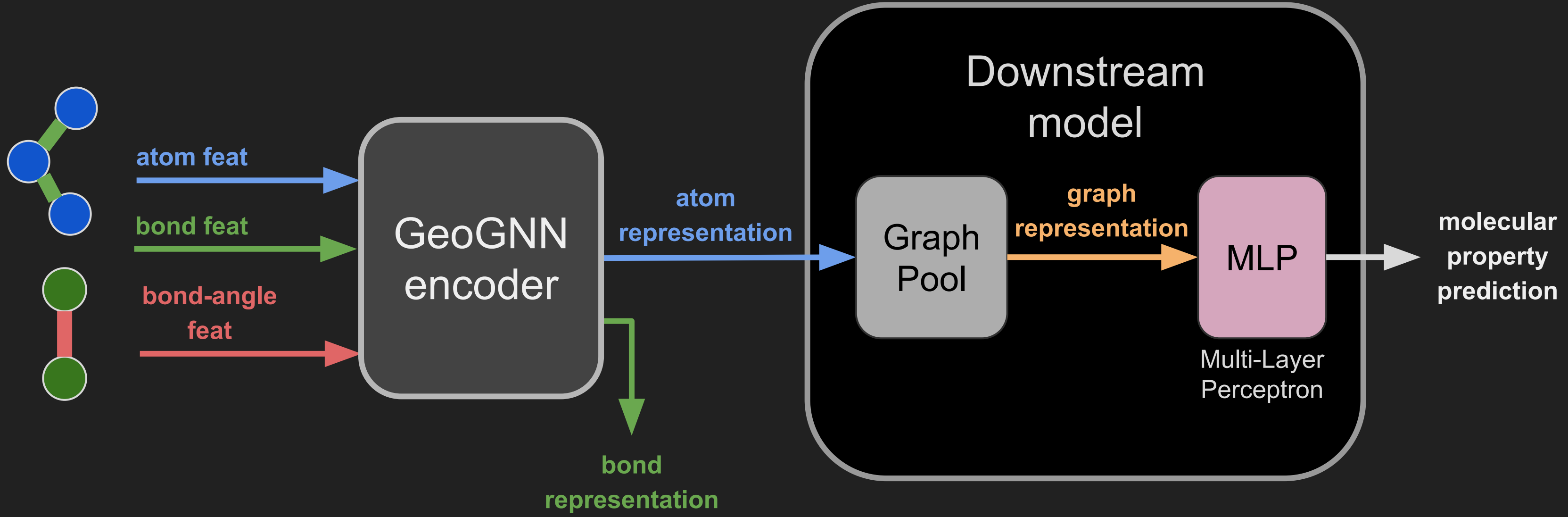
Atom

They proposed using 2 graphs to represent a molecule

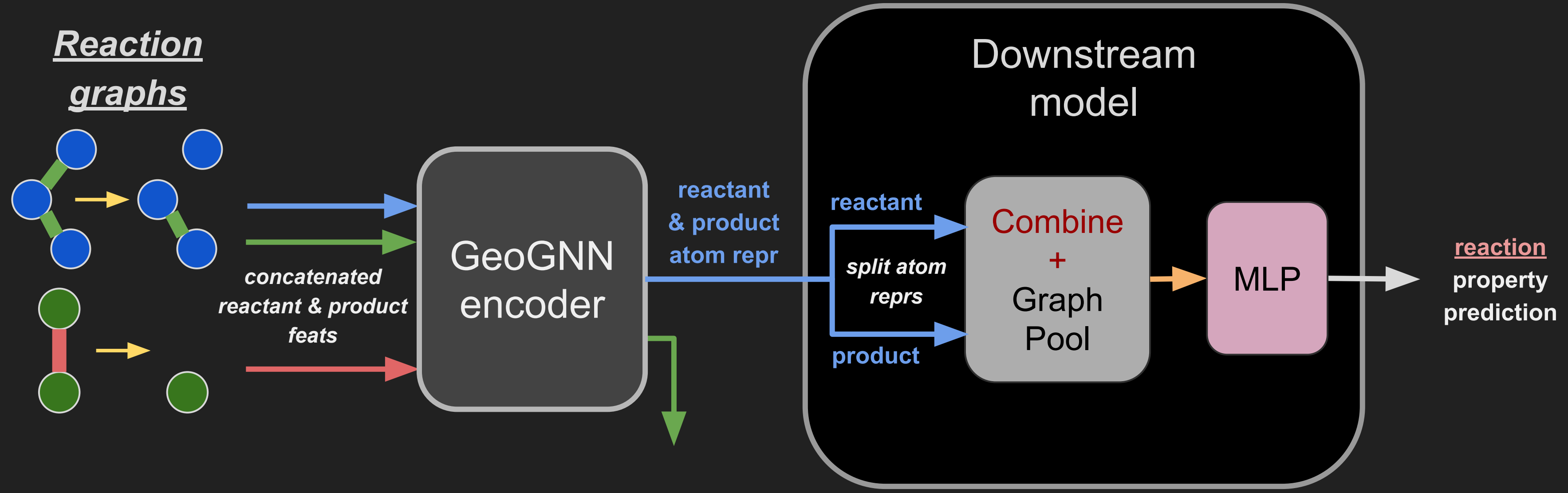
GeoGNN architecture



GeoGNN architecture

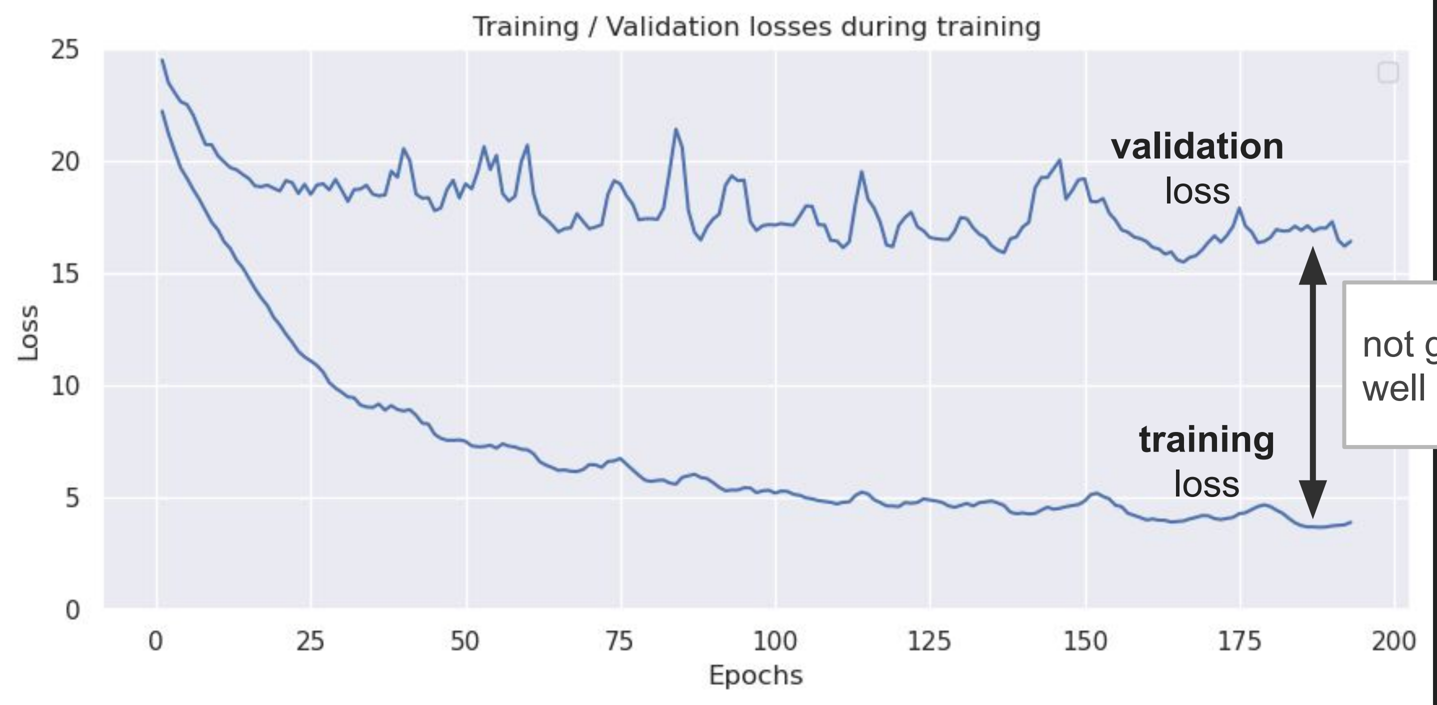


Adapting GeoGNN for reactions



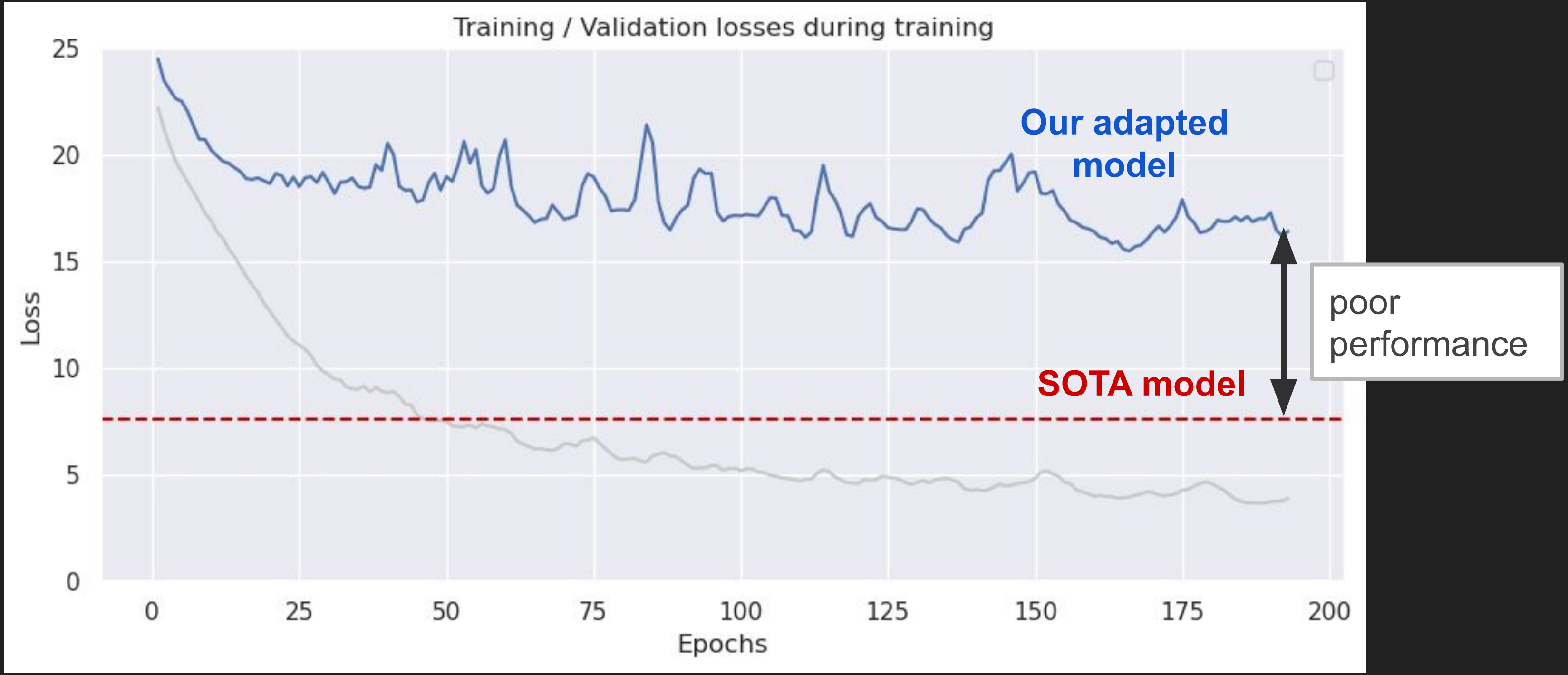
Results

Performance



not generalising well

Performance

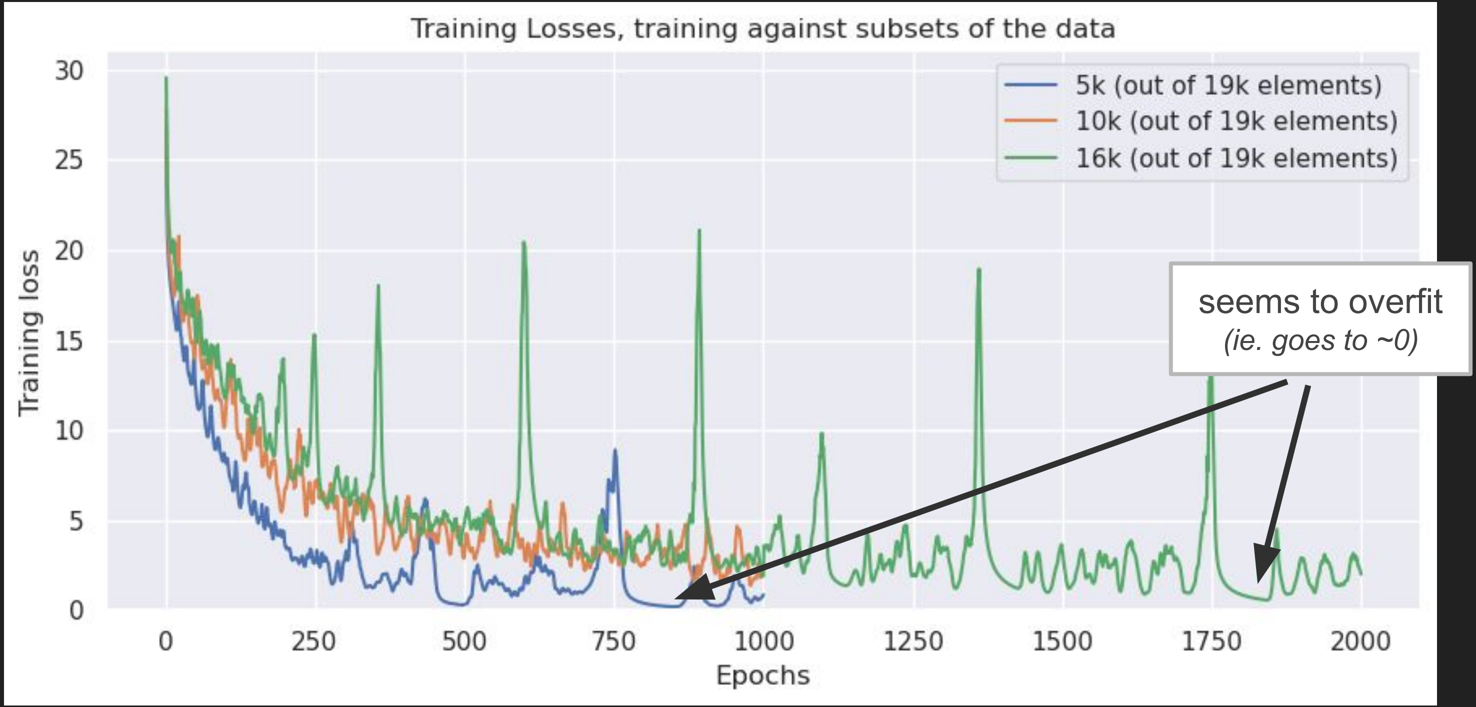


Debugging: Checking model's capacity

- **Intentionally** try to overfit *(ie. converges to ~0 loss)*
  smaller dataset sizes *(eg. 5, 10, 16k out of the 19k dataset)*

- If **can** overfit = model has the capacity to represent the dataset
- If **cannot** overfit = model is not powerful enough
  (or I coded something wrongly)

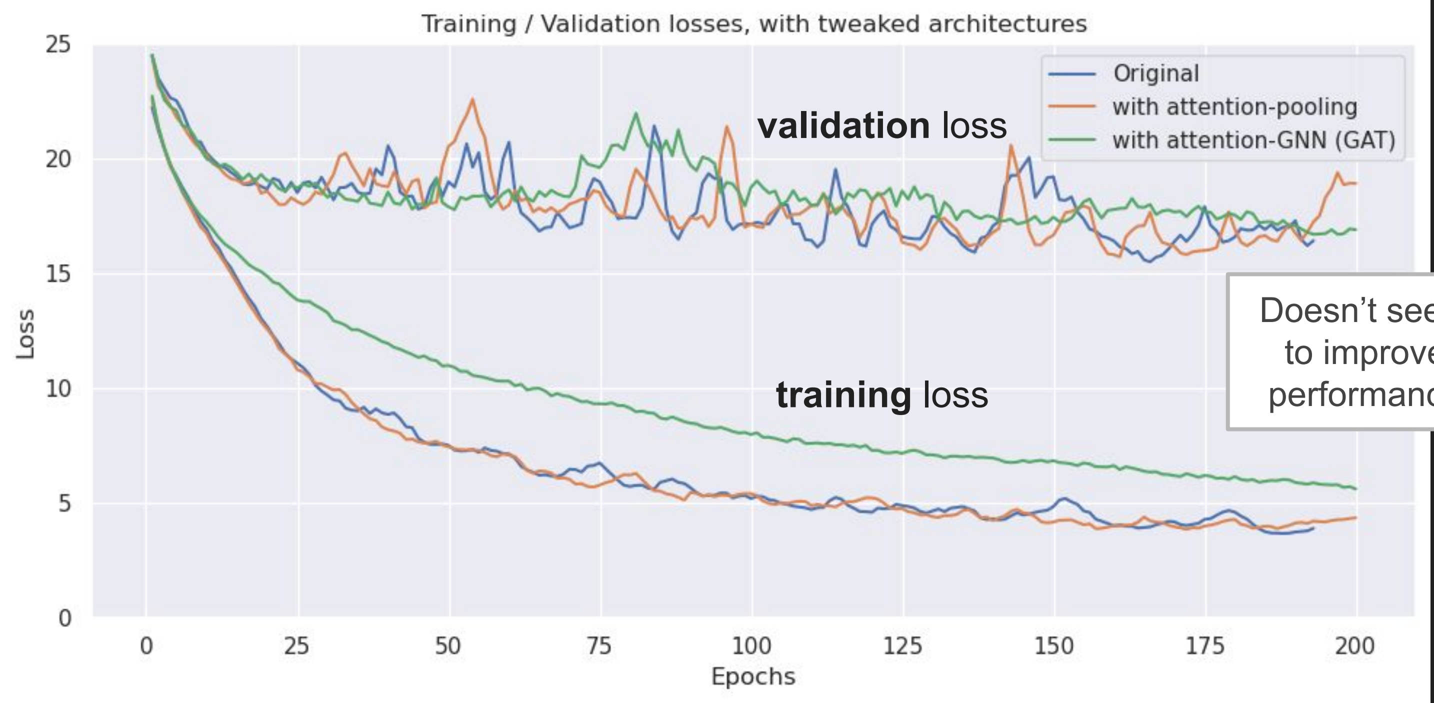
Debugging: Checking model’s capacity



seems to overfit
(ie. goes to ~0)

Thus:
model seems to have the capacity to represent the dataset

Tweaking the architecture



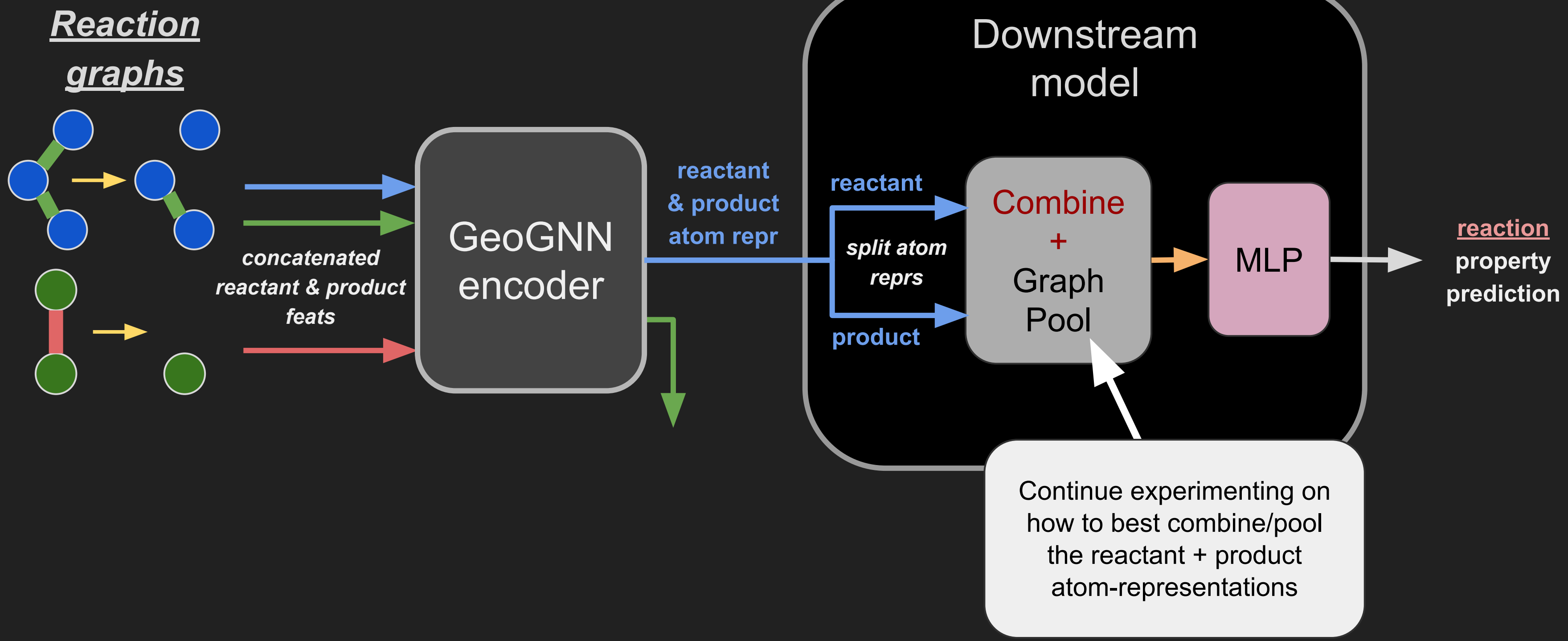
validation loss

training loss

Doesn't seem to improve performance

Next steps

Next steps: Tweaking combing/pooling



Continue experimenting on how to best combine/pool the reactant + product atom-representations

Next steps: Pretraining on molecular-properties

- Pretrain on molecular-property datasets
  then finetune on activation-energy reaction datasets
  to see if the pretraining helps

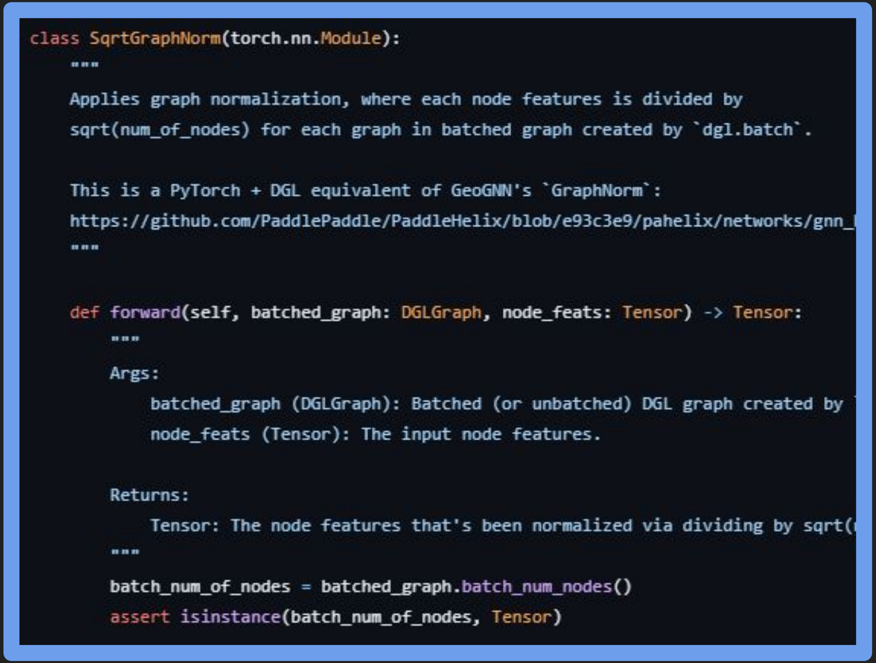
Challenges I faced

Challenges I faced

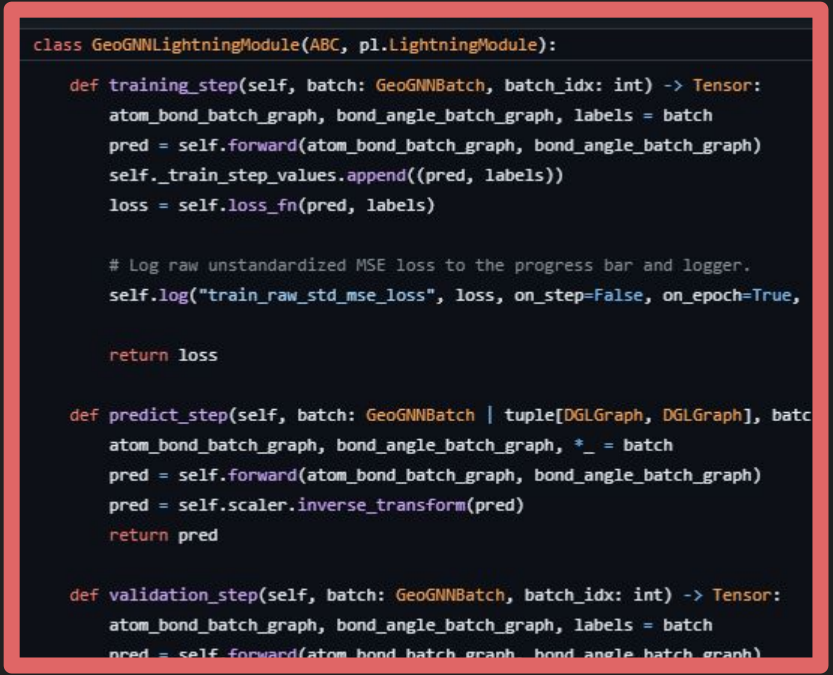
- Recoded GeoGNN’s architecture in PyTorch + DGL:
    - becuz their original code used a lesser-known library – paddlepaddle
    - and their code was very messy, w/o documentations
both of which made it hard to modify/adapt
    - recoded to gain experience in working with GNN / PyTorch

Challenges I faced

- Learnt PyTorch and DGL libraries
- To keep code clean, coded with type hintings, docstrings, PyTorch-Lightning



```python
class SqrtGraphNorm(torch.nn.Module):
    """
    Applies graph normalization, where each node features is divided by
    sqrt(num_of_nodes) for each graph in batched graph created by `dgl.batch`.

    This is a PyTorch + DGL equivalent of GeoGNN's `GraphNorm`:
    https://github.com/PaddlePaddle/PaddleHelix/blob/e93c3e9/pahelix/networks/gnn_
    """

    def forward(self, batched_graph: DGLGraph, node_feats: Tensor) -> Tensor:
        """
        Args:
            batched_graph (DGLGraph): Batched (or unbatched) DGL graph created by
            node_feats (Tensor): The input node features.

        Returns:
            Tensor: The node features that's been normalized via dividing by sqrt(
        """
        batch_num_of_nodes = batched_graph.batch_num_nodes()
        assert isinstance(batch_num_of_nodes, Tensor)
```

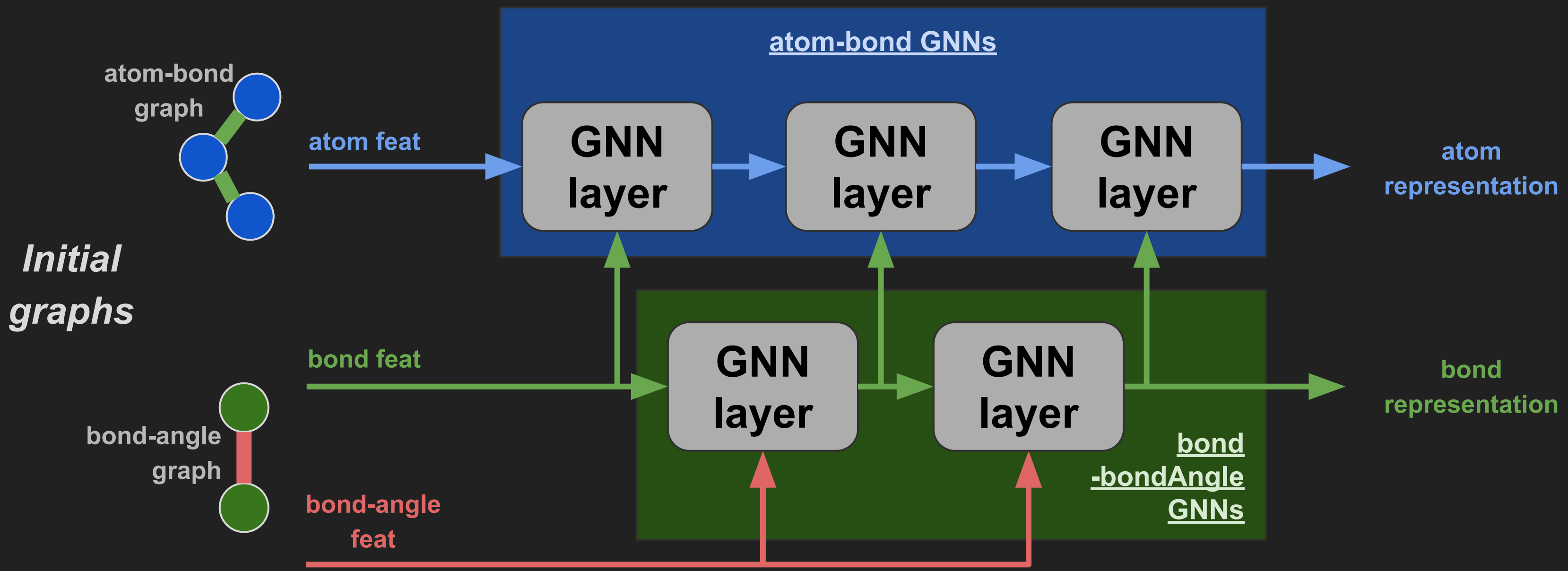


```python
class GeoGNNLightningModule(ABC, pl.LightningModule):

    def training_step(self, batch: GeoGNNBatch, batch_idx: int) -> Tensor:
        atom_bond_batch_graph, bond_angle_batch_graph, labels = batch
        pred = self.forward(atom_bond_batch_graph, bond_angle_batch_graph)
        self._train_step_values.append((pred, labels))
        loss = self.loss_fn(pred, labels)

        # Log raw unstandardized MSE loss to the progress bar and logger.
        self.log("train_raw_std_mse_loss", loss, on_step=False, on_epoch=True,

        return loss

    def predict_step(self, batch: GeoGNNBatch | tuple[DGLGraph, DGLGraph], batc
        atom_bond_batch_graph, bond_angle_batch_graph, *_ = batch
        pred = self.forward(atom_bond_batch_graph, bond_angle_batch_graph)
        pred = self.scaler.inverse_transform(pred)
        return pred

    def validation_step(self, batch: GeoGNNBatch, batch_idx: int) -> Tensor:
        atom_bond_batch_graph, bond_angle_batch_graph, labels = batch
        pred = self.forward(atom_bond_batch_graph, bond_angle_batch_graph)
```

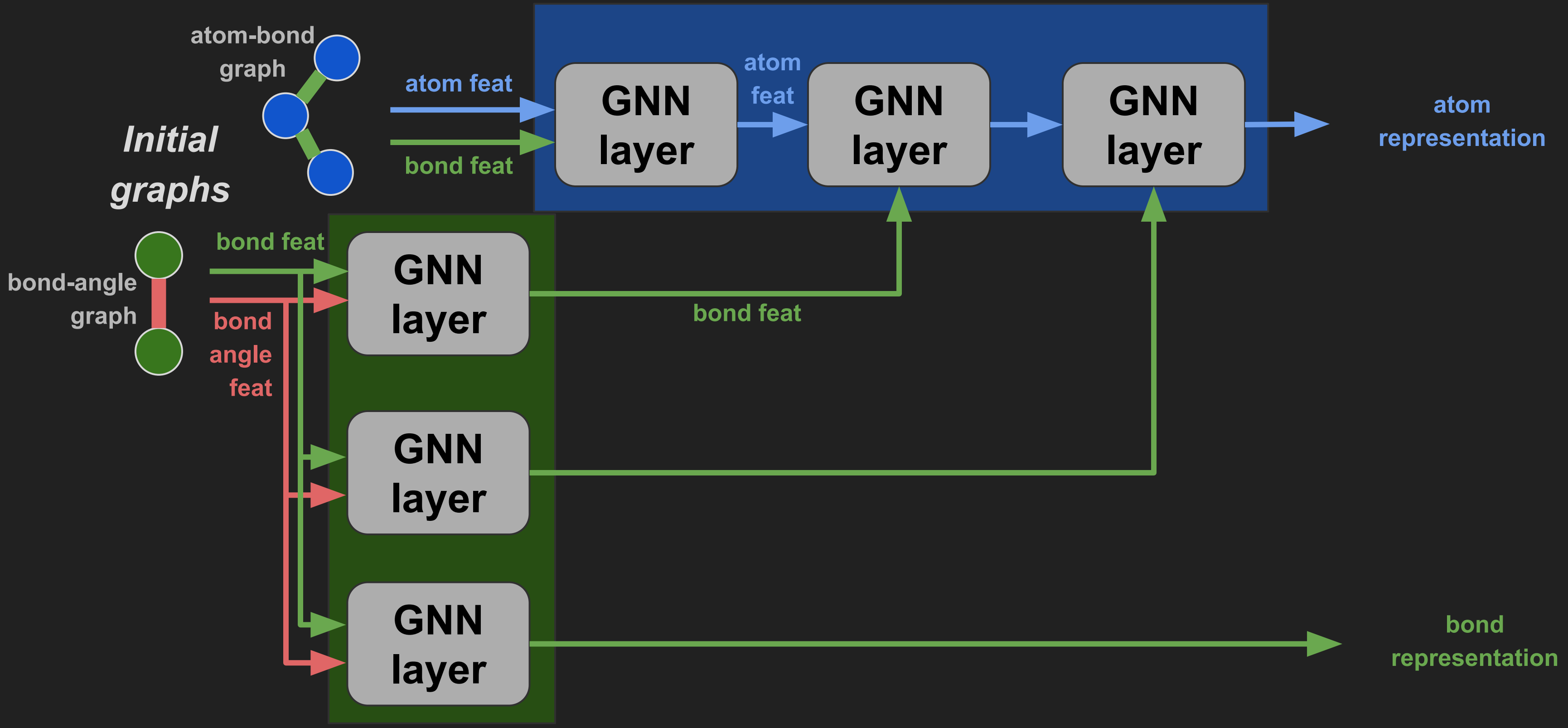
Challenges I faced

- The model GeoGNN coded wasn’t what they mentioned in the paper

What they **SAID** they did:



What they **ACTUALLY** did:



~ fin ~