ILC Weekly Meeting

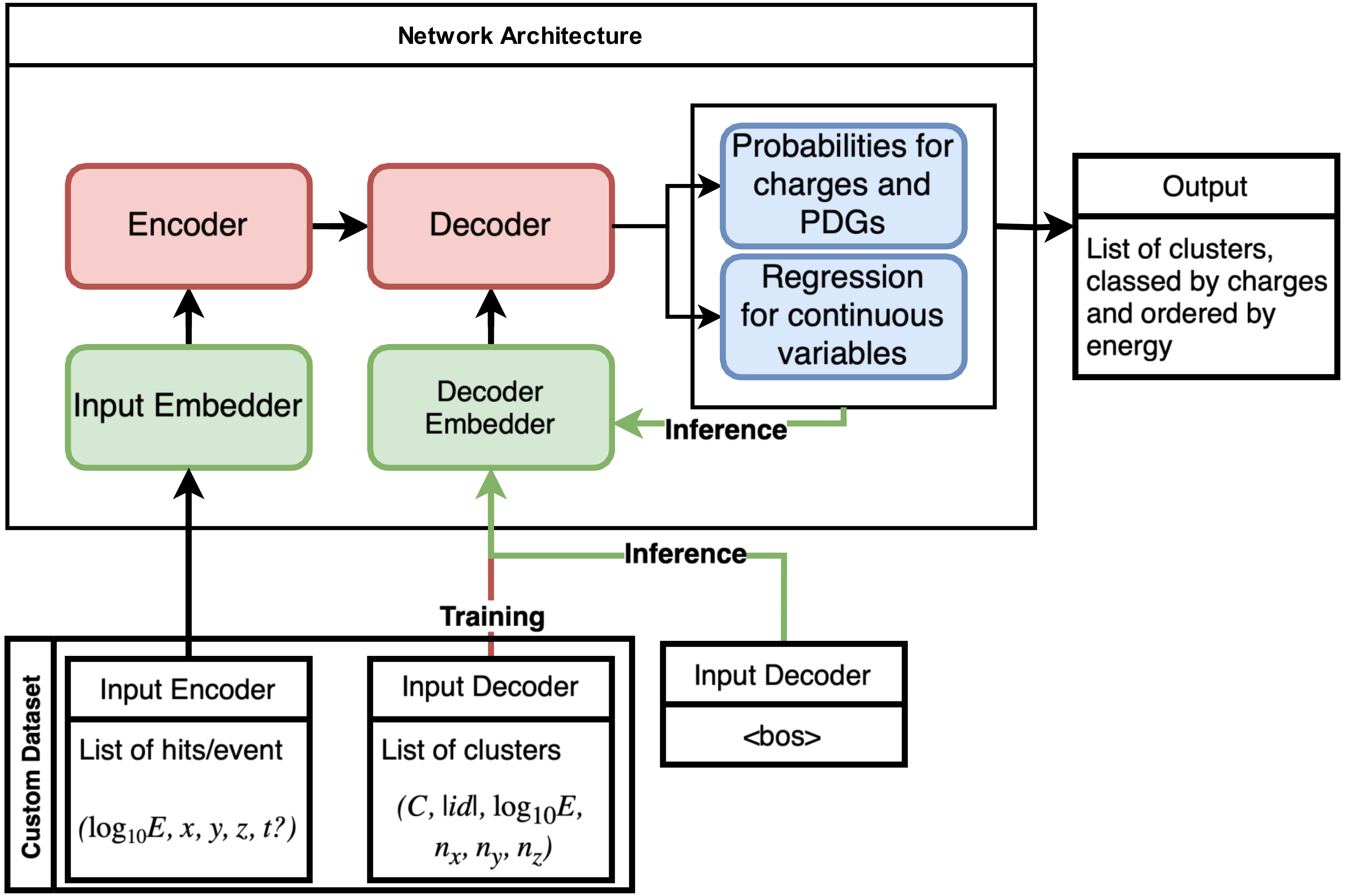
05.30.2024

Recap

Cluster information are obtained from MC Particle truth information.

3 loss functions, weighted by hyperparameters:

- Most common particle ids form vocabulary: $\gamma, K_S, K_L, K^+, \mu^-, p, n, \pi^\pm, e^-$ CrossEntropyLoss
- Charges form other vocabulary. -1, 0 ,1. Also CrossEntropyLoss
- Continuous variables are obtained by regression. MSE for the second loss function.



Network Architecture
Encoder
Decoder
Probabilities for charges and PDGs
Regression for continuous variables
Output
List of clusters, classed by charges and ordered by energy
Input Embedder
Decoder Embedder
Inference
Inference
Training
Custom Dataset
Input Encoder
List of hits/event
$(\log_{10}E, x, y, z, t?)$
Input Decoder
List of clusters
$(C, |id|, \log_{10}E, n_x, n_y, n_z)$
Input Decoder
<bos>

Since 2 weeks ago

- Implemented method to shrink labels from awkward arrays to 1 representative/ cluster

  ➡ Custom Dataset is now fully operational

- Implemented the vocabularies for PDGs and charges necessary for loss functions

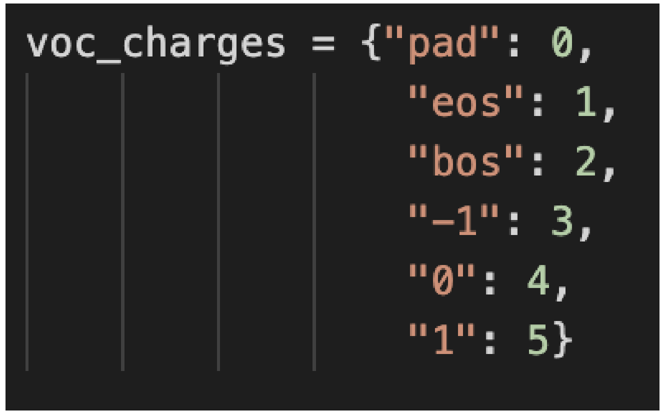
- Implemented the validate, train and inference function. Still needs testing

- Git repository at: https://github.com/Paul-3004/ILANCE_Transfo.git

Loss functions

- Pytorch’s CrossEntropyLoss for charges and particles pdg
  - Possibility to ignore a specified index, as long as the index is the same as target value

➡ Necessary to create vocabularies for charges and particles, then implemented in the Dataset’s special tokens



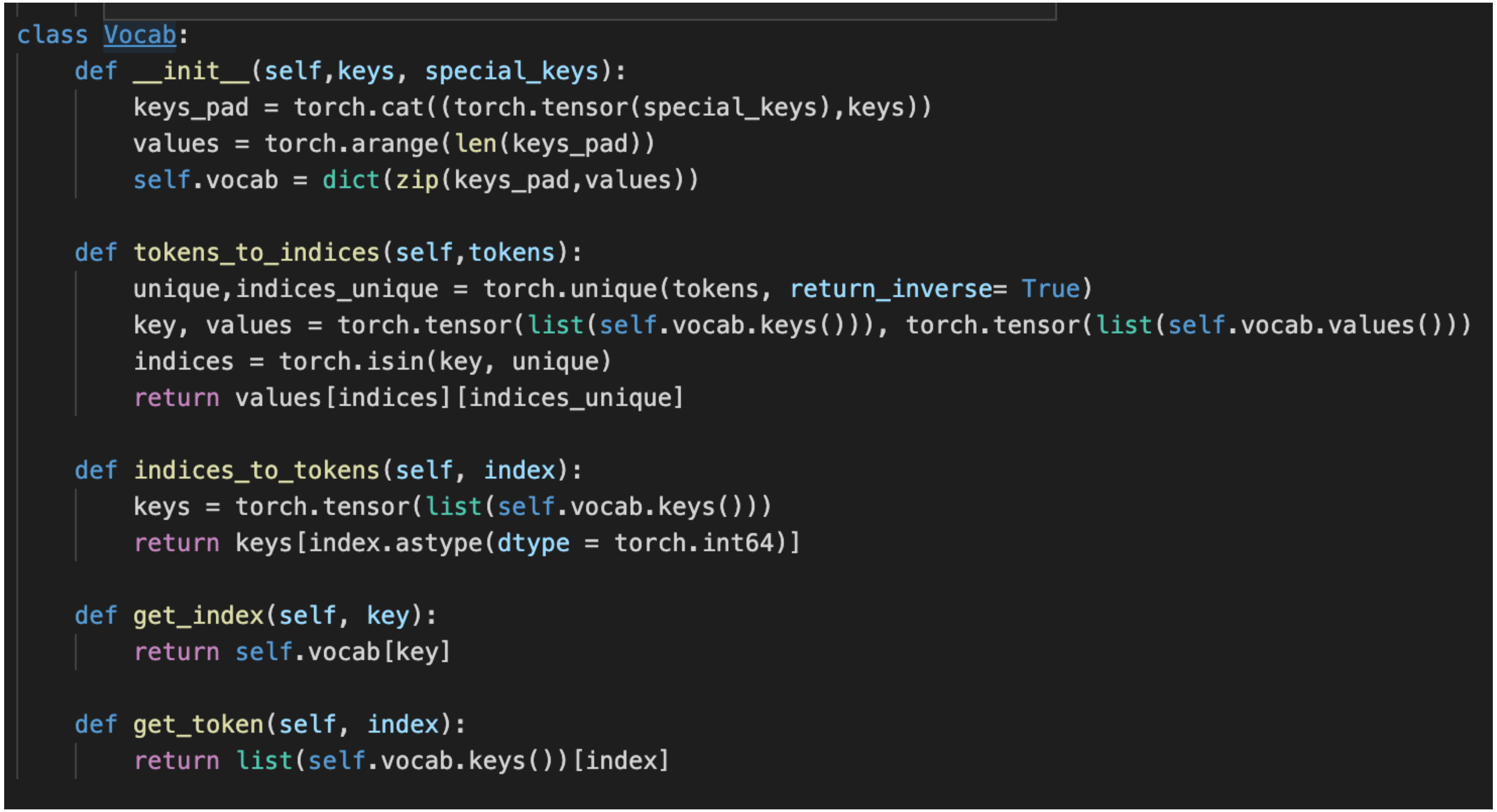
```
voc_charges = {"pad": 0,
               "eos": 1,
               "bos": 2,
               "-1": 3,
               "0": 4,
               "1": 5}
```

- Pytorch’s MSELoss for continous DOF.
  - No possibility of directly ignoring special tokens

➡ Can ignore the values by applying mask and setting their values to 0 manually.

Vocabs I

- One class for both instances
- Basically a dict that can do translation
- Keys: actual values of the charges/PDGs
- Values: unique indices associated to each particle
- 3 first entries are for the special keys (tokens)



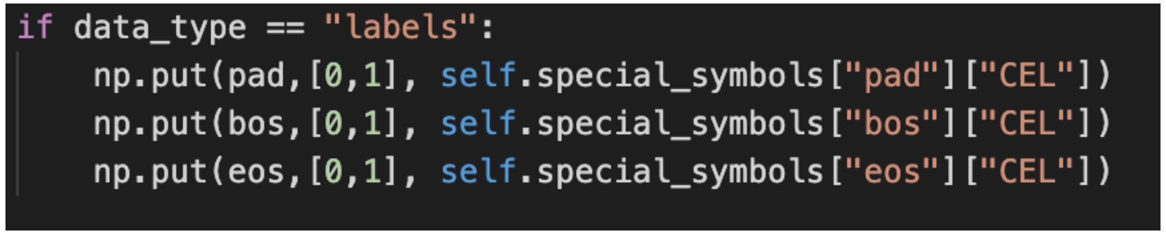
```
class Vocab:
    def __init__(self,keys, special_keys):
        keys_pad = torch.cat((torch.tensor(special_keys),keys))
        values = torch.arange(len(keys_pad))
        self.vocab = dict(zip(keys_pad,values))

    def tokens_to_indices(self,tokens):
        unique,indices_unique = torch.unique(tokens, return_inverse= True)
        key, values = torch.tensor(list(self.vocab.keys())), torch.tensor(list(self.vocab.values()))
        indices = torch.isin(key, unique)
        return values[indices][indices_unique]

    def indices_to_tokens(self, index):
        keys = torch.tensor(list(self.vocab.keys()))
        return keys[index.astype(dtype = torch.int64)]

    def get_index(self, key):
        return self.vocab[key]

    def get_token(self, index):
        return list(self.vocab.keys())[index]
```

Vocabs II

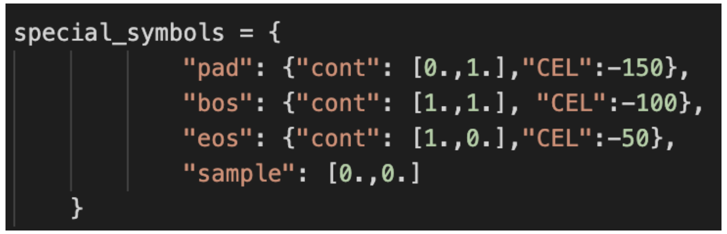
- Need to update special tokens during formatting

1. Adding values of keys in special tokens
2. Creating and using vocabs to translate

In AddSpecialSymbols::__call__

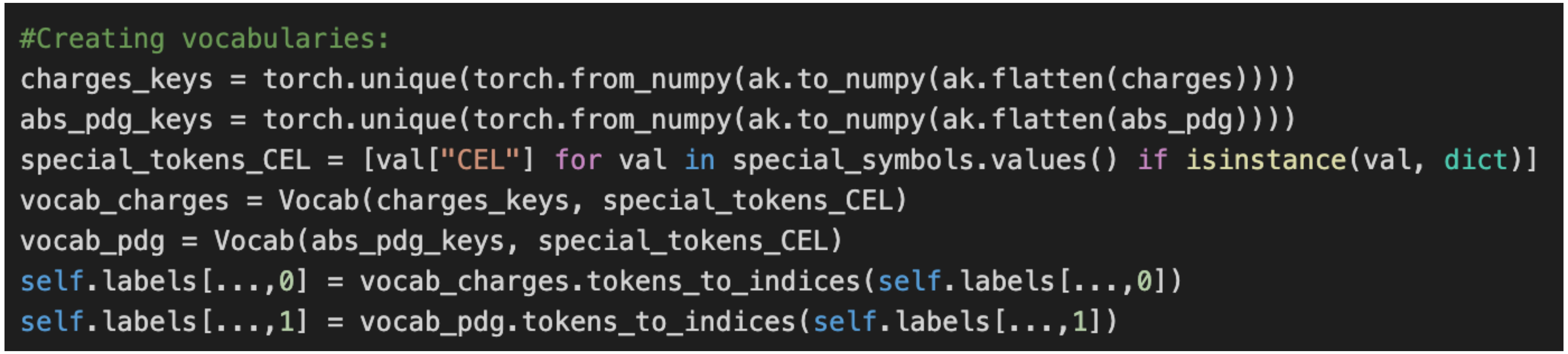


```
if data_type == "labels":
    np.put(pad,[0,1], self.special_symbols["pad"]["CEL"])
    np.put(bos,[0,1], self.special_symbols["bos"]["CEL"])
    np.put(eos,[0,1], self.special_symbols["eos"]["CEL"])
```

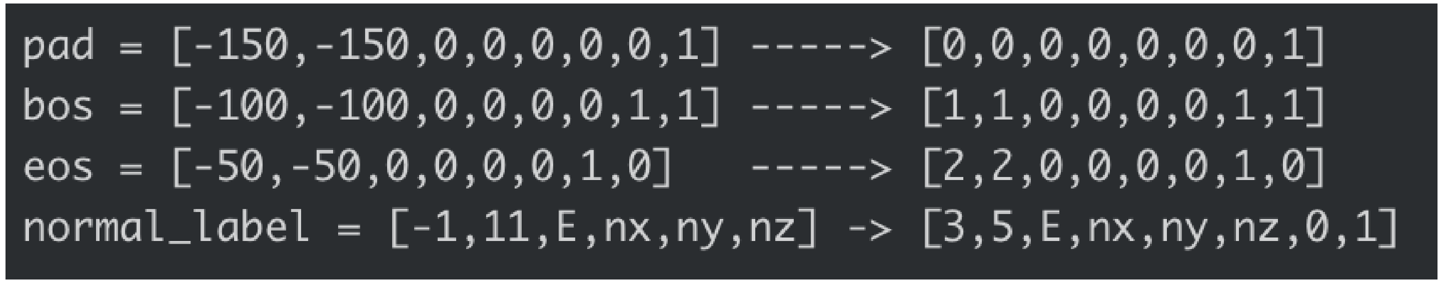


```
special_symbols = {
                "pad": {"cont": [0.,1.],"CEL":-150},
                "bos": {"cont": [1.,1.], "CEL":-100},
                "eos": {"cont": [1.,0.],"CEL":-50},
                "sample": [0.,0.]
    }
```

In CollectionHits::formatting

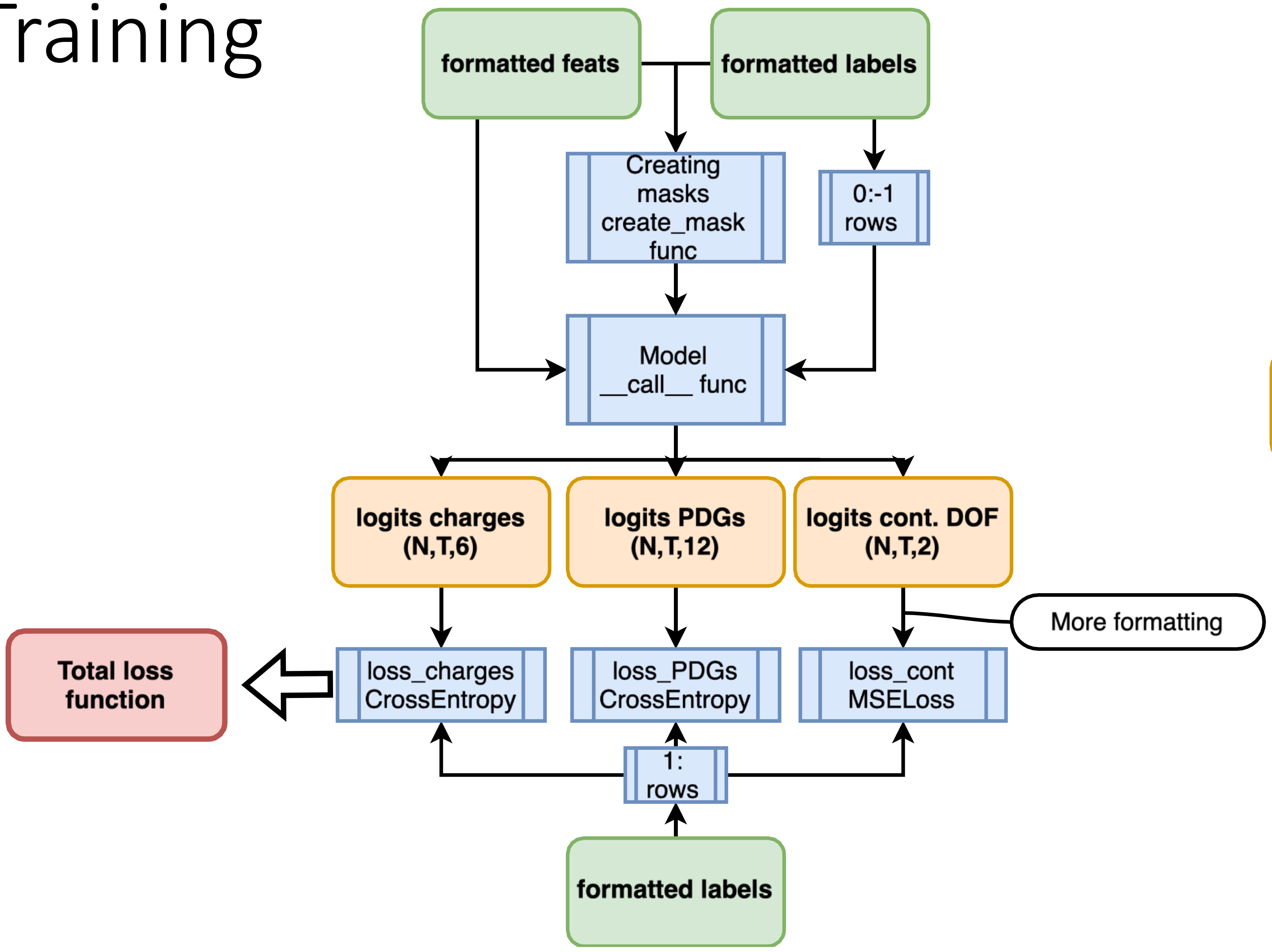


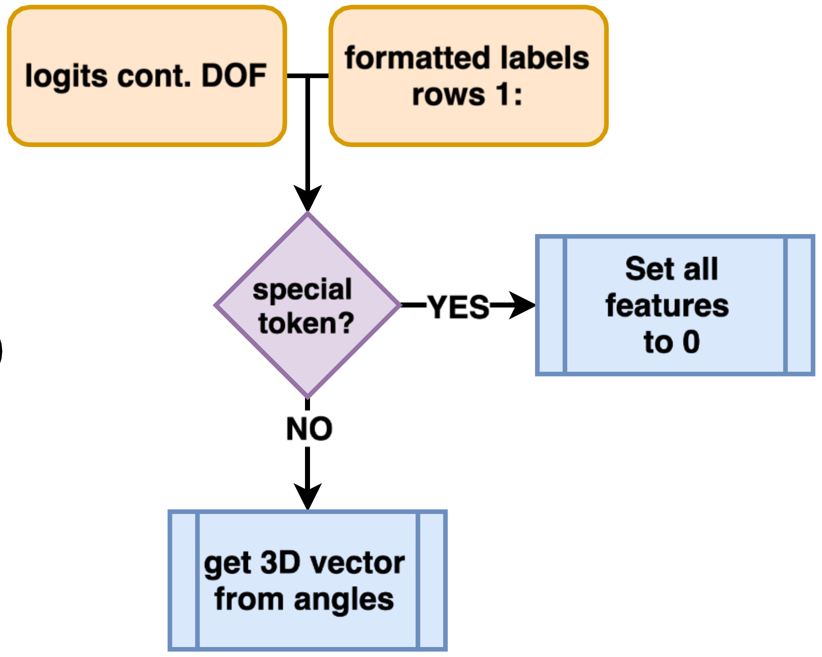
```
#Creating vocabularies:
charges_keys = torch.unique(torch.from_numpy(ak.to_numpy(ak.flatten(charges))))
abs_pdg_keys = torch.unique(torch.from_numpy(ak.to_numpy(ak.flatten(abs_pdg))))
special_tokens_CEL = [val["CEL"] for val in special_symbols.values() if isinstance(val, dict)]
vocab_charges = Vocab(charges_keys, special_tokens_CEL)
vocab_pdg = Vocab(abs_pdg_keys, special_tokens_CEL)
self.labels[...,0] = vocab_charges.tokens_to_indices(self.labels[...,0])
self.labels[...,1] = vocab_pdg.tokens_to_indices(self.labels[...,1])
```



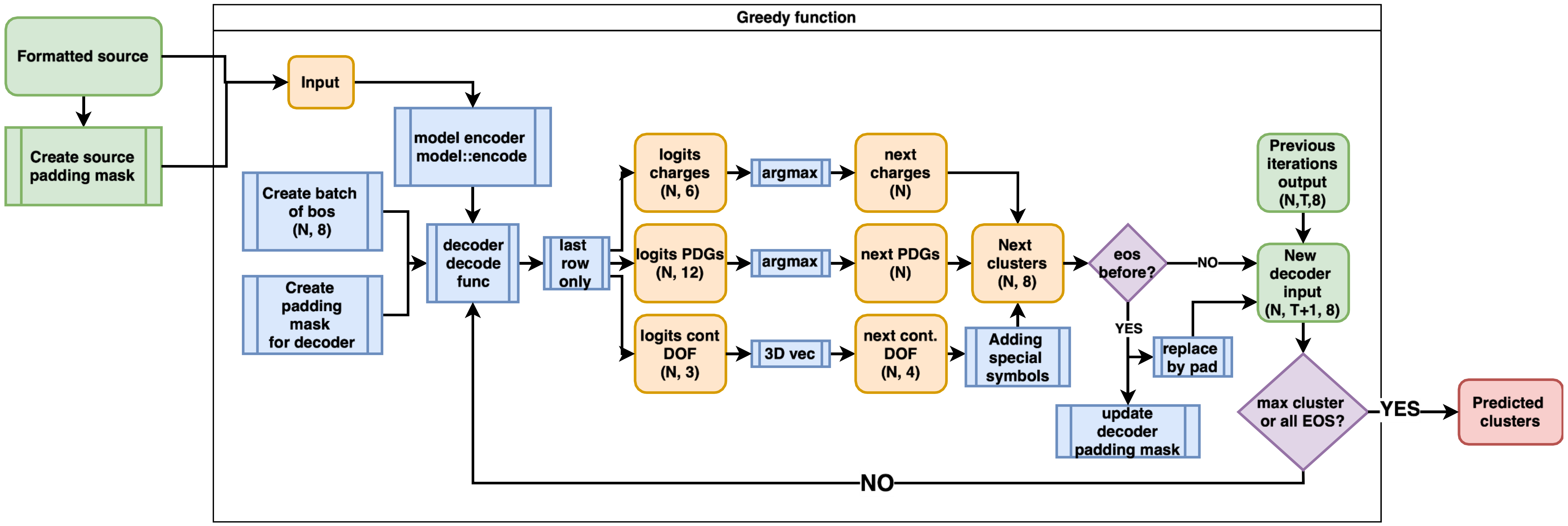
```
pad = [-150,-150,0,0,0,0,0,1] -----> [0,0,0,0,0,0,0,1]
bos = [-100,-100,0,0,0,0,1,1] -----> [1,1,0,0,0,0,1,1]
eos = [-50,-50,0,0,0,0,1,0]   -----> [2,2,0,0,0,0,1,0]
normal_label = [-1,11,E,nx,ny,nz] -> [3,5,E,nx,ny,nz,0,1]
```

Training


formatted feats
formatted labels
Creating masks create_mask func
0:-1 rows
Model __call__ func
logits charges (N,T,6)
logits PDGs (N,T,12)
logits cont. DOF (N,T,2)
More formatting
Total loss function
loss_charges CrossEntropy
loss_PDGs CrossEntropy
loss_cont MSELoss
1: rows
formatted labels


logits cont. DOF
formatted labels rows 1:
special token?
YES
Set all features to 0
NO
get 3D vector from angles

Inference


Greedy function
Formatted source
Create source padding mask
Input
model encoder model::encode
Create batch of bos (N, 8)
Create padding mask for decoder
decoder decode func
last row only
logits charges (N, 6)
logits PDGs (N, 12)
logits cont DOF (N, 3)
argmax
argmax
3D vec
next charges (N)
next PDGs (N)
next cont. DOF (N, 4)
Adding special symbols
Next clusters (N, 8)
eos before?
NO
YES
replace by pad
update decoder padding mask
Previous iterations output (N,T,8)
New decoder input (N, T+1, 8)
max cluster or all EOS?
YES
NO
Predicted clusters