model.py

```python
from tensorflow.keras.models import Model
from tensorflow.keras.layers import LSTM, Bidirectional, TimeDistributed, Dense, RepeatVector, ReLU
import tensorflow as tf


class FCEncoder(Model):
    def __init__(self, n_features):
        super().__init__()
        self.FC = []
        for level, n_feature in enumerate(n_features):
            if level == 0:
                pass
            elif level == len(n_features) - 1:
                dense = Dense(n_feature, name=f"encoder{level}")
                self.FC.append(dense)
                relu = ReLU(name=f"activation{level}")
                self.FC.append(relu)

            else:
                dense = Dense(n_feature, name=f"encoder{level}")
                self.FC.append(dense)
                relu = ReLU(name=f"activation{level}")
                self.FC.append(relu)

    def call(self, x):
        out = x
        for layer in self.FC:
            out = layer(out)
            # print(layer.name, out.shape)
        return out


class FCDecoder(Model):
    def __init__(self, n_features):
        super().__init__()
        self.FC = []
        for level, n_feature in enumerate(n_features):
            if level == 0:
                pass
            elif level == len(n_features) - 1:
                dense = Dense(n_feature, name=f"decoder{level}")
                self.FC.append(dense)
            else:
                dense = Dense(n_feature, name=f"decoder{level}")
                self.FC.append(dense)
                relu = ReLU(name=f"activation{level}")
                self.FC.append(relu)

    def call(self, x):
        out = x
        for layer in self.FC:
            out = layer(out)
            # print(layer.name,out.shape)
        return out


class AE(Model):
    def __init__(self, n_features):
        super().__init__()
        self.encoder = FCEncoder(n_features)
        n_features.reverse()
        self.decoder = FCDecoder(n_features)
        self.decoder2 = FCDecoder(n_features)

    def call(self, x):
        z = self.encoder(x)
        w1 = self.decoder(z)
        w2 = self.decoder2(z)
```

```
69            w3 = self.decoder2(self.encoder(w1))
70
71            return w1, w2, w3
72
```