these are my training and testing scripts

so my basic problem statement is my input features are range, positionx,y,z, velocity x,y,z and my output features are pipx1,pipy1,pipz1....pipx4,y4,z4
so in my inference script i am generating the plots of actual and predicted patterns .. but my predicted patterns are not so proper it has spikes and not following the actual pattern for few test files and for other test files it is following the pattern
so how should i fix this issue plz help

Based on your LSTM training and testing scripts, the issue of spiky and inconsistent predictions on some test files can stem from several factors. Here are the key problems and solutions to fix your LSTM model:[1] [2]

Normalization Mismatch

Your training script uses **MinMax scaling** on inputs but **Z-score normalization** on outputs, which creates an inconsistency. In testing, you're applying MinMax to inputs and Z-score inverse to outputs, but the scales may not match properly across different test files.[2] [3]

Fix: Use consistent normalization for both inputs and outputs:

```
# In training - use MinMax for both
Y_data = (Y_data - y_min) / (y_max - y_min + delta)

# In testing - inverse transform accordingly
data_predict = data_predict * (y_max - y_min) + y_min
```

Overfitting and Insufficient Regularization

Your model has a complex architecture (bidirectional LSTM + attention + residual connections) but limited dropout (0.2) that's not being applied in the forward pass. This can cause the model to memorize training patterns and fail to generalize, leading to spikes on unseen data.[4] [5] [2]

Fixes:

- Actually use the dropout layer in your forward pass: `out = self.dropout(out)`
- Add dropout after each LSTM layer and before the MLP
- Increase dropout rate to 0.3-0.5 for better regularization[6]
- Reduce model complexity or add L2 regularization weight (currently commented out)

Sequence Length and Stride Issues

Your training uses `stride=2` but testing uses `stride=1`, creating a distribution shift between training and inference. The model learns patterns from every-other-sample sequences but is tested on consecutive sequences.[1] [2]

Fix: Use the same stride for both training and testing:

```
# In test script, change:
stride = 2  # Match training stride
```

Learning Rate and Training Duration

With 150 epochs and learning rate 0.001, the model might be converging to a local minimum that doesn't generalize well. The lack of learning rate scheduling (currently commented out) prevents adaptation during training.[7]

Fixes:

- Uncomment and use the learning rate scheduler:

```
scheduler = ReduceLROnPlateau(optimizer, 'min', patience=10, factor=0.5)
scheduler.step(val_loss)  # After validation
```

- Monitor validation loss curves to detect overfitting early
- Implement early stopping if validation loss plateaus

Data Scaling Statistics

Your model saves normalization statistics from the **entire dataset** before train-test split, which can cause data leakage. Test files may have different value ranges that weren't captured during training.[2]

Fix: Calculate statistics only on training data:

```
# After train_test_split
x_mean = np.mean(X_train.reshape(-1, input_size), axis=0)
x_std = np.std(X_train.reshape(-1, input_size), axis=0)
x_min = np.min(X_train.reshape(-1, input_size), axis=0)
x_max = np.max(X_train.reshape(-1, input_size), axis=0)

y_mean = np.mean(y_train)
y_std = np.std(y_train)
y_min = np.min(y_train, axis=0)
y_max = np.max(y_train, axis=0)
```

Model Evaluation Mode

Ensure your model is in evaluation mode during testing to disable dropout and batch normalization effects properly:[1]

```
model1.eval()  # Already present but verify dropout layers behave correctly
```

Additional Recommendations

- Add gradient clipping (currently commented) to prevent exploding gradients: `torch.nn.utils.clip_grad_norm_(lstm.parameters(), max_norm=1.0)`[8]
- Try simpler architectures first - your current model with 4 stacked LSTM+attention layers may be too complex for the task[4]
- Validate that test files have similar statistical distributions as training data
- Plot training/validation loss curves to diagnose overfitting vs underfitting[7]

The most critical fixes are ensuring consistent normalization across train/test, using the same stride, and properly applying dropout throughout the network.[3] [6]

⁂

1. test_lstm_only_inputs_PIP1.py
2. LSTM-only_inputs_v2.py
3. http://jatit.org/volumes/Vol102No24/8Vol102No24.pdf
4. https://stackoverflow.com/questions/48034625/keras-lstm-predicted-timeseries-squashed-and-shifted
5. https://stackoverflow.com/questions/67149124/unstable-results-using-lstm-for-high-frequency-stock-forecasting
6. https://codesignal.com/learn/courses/time-series-forecasting-with-lstms/lessons/optimizing-lstm-performance-for-time-series-forecasting
7. https://www.machinelearningmastery.com/diagnose-overfitting-underfitting-lstm-models/
8. https://towardsdatascience.com/lstm-for-time-series-prediction-de8aeb26f2ca/
9. https://stackoverflow.com/questions/50054419/extremely-poor-prediction-lstm-time-series
10. https://www.tandfonline.com/doi/full/10.1080/08839514.2024.2377510
11. https://thesai.org/Downloads/Volume14No1/Paper_33-Time_Series_Forecasting_using_LSTM_and_ARIMA.pdf
12. https://www.machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/