Time Series Benchmarking

Crossformer

Mohit Dalal

Table of contents

Original Paper Details...

CROSSFORMER: TRANSFORMER UTILIZING CROSS-DIMENSION DEPENDENCY FOR MULTIVARIATE TIME SERIES FORECASTING

Yunhao Zhang & Junchi Yan*
MoE Key Lab of Artificial Intelligence, Shanghai Jiao Tong University and Shanghai AI Lab
{zhangyunhao, yanjunchi}@sjtu.edu.cn
Code: https://github.com/Thinklab-SJTU/Crossformer

ABSTRACT

Recently many deep models have been proposed for multivariate time series (MTS) forecasting. In particular, Transformer-based models have shown great potential because they can capture long-term dependency. However, existing Transformer-based models mainly focus on modeling the temporal dependency (cross-time dependency) yet often omit the dependency among different variables (cross-dimension dependency), which is critical for MTS forecasting. To fill the gap, we propose Crossformer, a Transformer-based model utilizing cross-dimension dependency for MTS forecasting. In Crossformer, the input MTS is embedded into a 2D vector array through the Dimension-Segment-Wise (DSW) embedding to preserve time and dimension information. Then the Two-Stage Attention (TSA) layer is proposed to efficiently capture the cross-time and cross-dimension dependency. Utilizing DSW embedding and TSA layer, Crossformer establishes a Hierarchical Encoder-Decoder (HED) to use the information at different scales for the final forecasting. Extensive experimental results on six real-world datasets show the effectiveness of Crossformer against previous state-of-the-arts.

Paper Experimental Setup

4 EXPERIMENTS

4.1 PROTOCOLS

Datasets We conduct experiments on six real-world datasets following Zhou et al. (2021); Wu et al. (2021a). **1) ETTh1** (Electricity Transformer Temperature-hourly), **2) ETTm1** (Electricity Transformer Temperature-minutely), **3) WTH** (Weather), **4) ECL** (Electricity Consuming Load), **5) ILI** (Influenza-Like Illness), **6) Traffic**. The train/val/test splits for the first four datasets are same as Zhou et al. (2021), the last two are split by the ratio of 0.7:0.1:0.2 following Wu et al. (2021a).

Baselines We use the following popular models for MTS forecasting as baselines:**1) LSTMa** (Bahdanau et al., 2015), **2) LSTnet** (Lai et al., 2018), **3) MTGNN** (Wu et al., 2020), and recent Transformer-based models for MTS forecasting: **4) Transformer** (Vaswani et al., 2017), **5) Informer** (Zhou et al., 2021), **6) Autoformer** (Wu et al., 2021a), **7) Pyraformer** (Liu et al., 2021a) and **8) FEDformer** (Zhou et al., 2022).

Setup We use the same setting as in Zhou et al. (2021): train/val/test sets are zero-mean normalized with the mean and std of training set. On each dataset, we evaluate the performance over the changing future window size τ. For each τ, the past window size T is regarded as a hyper-parameter to search which is a common protocol in recent MTS transformer literature (Zhou et al., 2021; Liu et al., 2021a). We roll the whole set with stride $= 1$ to generate different input-output pairs. The Mean Square Error (MSE) and Mean Absolute Error (MAE) are used as evaluation metrics. All experiments are repeated for 5 times and the mean of the metrics reported. Our Crossformer only utilize the past series to forecast the future, while baseline models use additional covariates such as hour-of-the-day. Details about datasets, baselines, implementation, hyper-parameters are shown in Appendix A.

Paper Results

Table 1: MSE/MAE with different prediction lengths. Bold/underline indicates the best/second. Results of LSTMa, LSTnet, Transformer, Informer on the first 4 datasets are from Zhou et al. (2021).

Models		LSTMa		LSTnet		MTGNN		Transformer		Informer		Autoformer		Pyraformer		FEDformer		Crossformer	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	24	0.650	0.624	1.293	0.901	0.336	0.393	0.620	0.577	0.577	0.549	0.439	0.440	0.493	0.507	0.318	0.384	**0.305**	**0.367**
	48	0.720	0.675	1.456	0.960	0.386	0.429	0.692	0.671	0.685	0.625	0.429	0.442	0.554	0.544	**0.342**	0.396	0.352	**0.394**
	168	1.212	0.867	1.997	1.214	0.466	0.474	0.947	0.797	0.931	0.752	0.493	0.479	0.781	0.675	0.412	0.449	**0.410**	**0.441**
	336	1.424	0.994	2.655	1.369	0.736	0.643	1.094	0.813	1.128	0.873	0.509	0.492	0.912	0.747	0.456	0.474	**0.440**	**0.461**
	720	1.960	1.322	2.143	1.380	0.916	0.750	1.241	0.917	1.215	0.896	0.539	0.537	0.993	0.792	0.521	**0.515**	**0.519**	0.524
ETTm1	24	0.621	0.629	1.968	1.170	0.260	0.324	0.306	0.371	0.323	0.369	0.410	0.428	0.310	0.371	0.290	0.364	**0.211**	**0.293**
	48	1.392	0.939	1.999	1.215	0.386	0.408	0.465	0.470	0.494	0.503	0.485	0.464	0.465	0.464	0.342	0.396	**0.300**	**0.352**
	96	1.339	0.913	2.762	1.542	0.428	0.446	0.681	0.612	0.678	0.614	0.502	0.476	0.520	0.504	0.366	0.412	**0.320**	**0.373**
	288	1.740	1.124	1.257	2.076	0.469	0.488	1.162	0.879	1.056	0.786	0.604	0.522	0.729	0.657	**0.398**	0.433	0.404	**0.427**
	672	2.736	1.555	1.917	2.941	0.620	0.571	1.231	1.103	1.192	0.926	0.607	0.530	0.980	0.678	**0.455**	**0.464**	0.569	0.528
WTH	24	0.546	0.570	0.615	0.545	0.307	0.356	0.349	0.397	0.335	0.381	0.363	0.396	0.301	0.359	0.357	0.412	**0.294**	**0.343**
	48	0.829	0.677	0.660	0.589	0.388	0.422	0.386	0.433	0.395	0.459	0.456	0.462	0.376	0.421	0.428	0.458	**0.370**	**0.411**
	168	1.038	0.835	0.748	0.647	0.498	0.512	0.613	0.582	0.608	0.567	0.574	0.548	0.519	0.521	0.564	0.541	**0.473**	**0.494**
	336	1.657	1.059	0.782	0.683	0.506	0.523	0.707	0.634	0.702	0.620	0.600	0.571	0.539	0.543	0.533	0.536	**0.495**	**0.515**
	720	1.536	1.109	0.851	0.757	**0.510**	**0.527**	0.834	0.741	0.831	0.731	0.587	0.570	0.547	0.553	0.562	0.557	0.526	0.542
ECL	48	0.486	0.572	0.369	0.445	0.173	0.280	0.334	0.399	0.344	0.393	0.241	0.351	0.478	0.471	0.229	0.338	**0.156**	**0.255**
	168	0.574	0.602	0.394	0.476	0.236	0.320	0.353	0.420	0.368	0.424	0.299	0.387	0.452	0.455	0.263	0.361	**0.231**	**0.309**
	336	0.886	0.795	0.419	0.477	0.328	0.373	0.381	0.439	0.381	0.431	0.375	0.428	0.463	0.456	**0.305**	0.386	0.323	**0.369**
	720	1.676	1.095	0.556	0.565	0.422	**0.410**	0.391	0.438	0.406	0.443	0.377	0.434	0.480	0.461	**0.372**	0.434	0.404	0.423
	960	1.591	1.128	0.605	0.599	0.471	0.451	0.492	0.550	0.460	0.548	**0.366**	**0.426**	0.550	0.489	0.393	0.449	0.433	0.438
ILI	24	4.220	1.335	4.975	1.660	4.265	1.387	3.954	1.323	4.588	1.462	3.101	1.238	3.970	1.338	**2.687**	**1.147**	3.041	1.186
	36	4.771	1.427	5.322	1.659	4.777	1.496	4.167	1.360	4.845	1.496	3.397	1.270	4.377	1.410	**2.887**	**1.160**	3.406	1.232
	48	4.945	1.462	5.425	1.632	5.333	1.592	4.746	1.463	4.865	1.516	2.947	1.203	4.811	1.503	**2.797**	**1.155**	3.459	1.221
	60	5.176	1.504	5.477	1.675	5.070	1.552	5.219	1.553	5.212	1.576	3.019	1.202	5.204	1.588	**2.809**	**1.163**	3.640	1.305
Traffic	24	0.668	0.378	0.648	0.403	0.506	0.278	0.597	0.332	0.608	0.334	0.550	0.363	0.606	0.338	0.562	0.375	**0.491**	**0.274**
	48	0.709	0.400	0.709	0.425	**0.512**	0.298	0.658	0.369	0.644	0.359	0.595	0.376	0.619	0.346	0.567	0.374	0.519	**0.295**
	168	0.900	0.523	0.713	0.435	0.521	0.319	0.664	0.363	0.660	0.391	0.649	0.407	0.635	0.347	0.607	0.385	**0.513**	**0.289**
	336	1.067	0.599	0.741	0.451	0.540	0.335	0.654	0.358	0.747	0.405	0.624	0.388	0.641	0.347	0.624	0.389	**0.530**	**0.300**
	720	1.461	0.787	0.768	0.474	**0.557**	0.343	0.685	0.370	0.792	0.430	0.674	0.417	0.670	0.364	0.623	0.378	0.573	**0.313**

Effects of hyper parameters

4.4 Effect of Hyper-Parameters

We evaluate the effect of two hyper-parameters: segment length (L_{seg} in Eq. 1) and number of routers in TSA (c in Cross-Dimension Stage of TSA) on the ETTh1 dataset. **Segment Length:** In Fig. 4(a), we prolong the segment length from 4 to 24 and evaluate MSE with different prediction windows. For short-term forecasting ($\tau = 24, 48$), smaller segment yields relevantly better results, but the prediction accuracy is stable. For long-term forecasting ($\tau \geq 168$), prolonging the segment length from 4 to 24 causes the MSE to decrease. This indicates that long segments should be used for long-term forecasting. We further prolong the segment length to 48 for $\tau = 336, 720$, the MSE is slightly larger than that of 24. The possible reason is that 24 hours exactly matches the daily period of this dataset, while 48 is too coarse to capture fine-grained information. **Number of Routers in TSA Layer:** Number of Routers c controls the information bandwidth among all dimensions. As Fig. 4(b) shows, the performance of Crossformer is stable w.r.t to c for $\tau \leq 336$. For $\tau = 720$, the MSE is large when $c = 3$ but decreases and stabilizes when $c \geq 5$. In pratice, we set $c = 10$ to balance the prediction accuracy and computation efficiency.

Reproducing the results

Following the readme provided by paper, I reproduced the results for the below setting (snapshot from original paper code repo):

Reproducibility

1. Put datasets to conduct experiments into folder `datasets/`. We have already put `ETTh1` and `ETTm1` into it. `WTH` and `ECL` can be downloaded from https://github.com/zhouhaoyi/Informer2020. `ILI` and `Traffic` can be downloaded from https://github.com/thuml/Autoformer. Note that the `WTH` we used in the paper is the one with 12 dimensions from Informer, not the one with 21 dimensions from Autoformer.
2. To get results of Crossformer with $T = 168, \tau = 24, L_{seg} = 6$ on ETTh1 dataset, run:

```
python main_crossformer.py --data ETTh1 --in_len 168 --out_len 24 --seg_len 6 --itr 1
```

The model will be automatically trained and tested. The trained model will be saved in folder `checkpoints/` and evaluated metrics will be saved in folder `results/`.

3. You can also evaluate a trained model by running:

```
python eval_crossformer.py --checkpoint_root ./checkpoints --setting_name
Crossformer_ETTh1_il168_ol24_sl6_win2_fa10_dm256_nh4_el3_itr0
```

Both the files main_crossformer.py and eval_crossformer.py are working as intended. Results are shown on the next slide.

#	Setting Name	Description	Results MSE	Results MAE	Comment
1	Crossformer_ETTh1_il168_ol24_sl6_win2_fa10_dm256_nh4_el3_itr0	Original setting of the code. Trying to reproduce the results	0.3445	0.3929	Comparable to the numbers mentioned in the paper. MSE = 0.410 | MAE = 0.441
2	Crossformer_ETTh1_il168_ol24_sl6_win2_fa10_dm256_nh4_el3_itr0	Changed the normalization to MinMaxScaler	0.0312	0.1369	
3	Crossformer_ETTh1_il168_ol1_sl6_win2_fa10_dm256_nh4_el3_itr0	Predicting only one time stamp into the future.	0.0114	0.0731	Error dropped as we are only trying to predict for only one time stamp in the future.
4	Crossformer_stock_AKAM_il168_ol1_sl6_win2_fa10_dm256_nh4_el3_itr0	Changed the data to stock market data. AKAM 2023 train file. For this run I had simply split the data into 70:20:10.	0.0030	0.0344	Numbers looks good. But the plots for the ret column are a bit weird.
5	Crossformer_stock_AKAM_il168_ol1_sl6_win2_fa10_dm256_nh4_el3_itr0	Calculated the results by taking the inverse of final output. This is done to bring everything back to its original scale.	128.7794	4.4847	Numbers don't look good. I have added plot for further analysis.

Results

MinMaxScaler Implementation

```python
    def train(self, setting):
        train_data, train_loader = self._get_data(flag = 'train')
        vali_data, vali_loader = self._get_data(flag = 'val')
        test_data, test_loader = self._get_data(flag = 'test')

        path = os.path.join(self.args.checkpoints, setting)
        if not os.path.exists(path):
            os.makedirs(path)
        with open(os.path.join(path, "args.json"), 'w') as f:
            json.dump(vars(self.args), f, indent=True)

        scale_statistic = {'min': train_data.scaler.data_min_, 'max': train_data.scaler.data_max_} # MinMaxScaler - New code
        # scale_statistic = {'mean': train_data.scaler.mean, 'std': train_data.scaler.std} # StandardScaler - Original code

        with open(os.path.join(path, "scale_statistic.pkl"), 'wb') as f:
            pickle.dump(scale_statistic, f)

        train_steps = len(train_loader)
        early_stopping = EarlyStopping(patience=self.args.patience, verbose=True)

        model_optim = self._select_optimizer()
        criterion =  self._select_criterion()
```

Change 1 – train function in the exp_crossformer.py

Description: Just saving the min, max stats of train data in a pickle.

```python
    def __read_data__(self):
        df_raw = pd.read_csv(os.path.join(self.root_path,
                                          self.data_path))
        if (self.data_split[0] > 1):
            train_num = self.data_split[0]; val_num = self.data_split[1]; test_num = self.data_split[2];
        else:
            train_num = int(len(df_raw)*self.data_split[0]);
            test_num = int(len(df_raw)*self.data_split[2])
            val_num = len(df_raw) - train_num - test_num;
        border1s = [0, train_num - self.in_len, train_num + val_num - self.in_len]
        border2s = [train_num, train_num+val_num, train_num + val_num + test_num]

        border1 = border1s[self.set_type]
        border2 = border2s[self.set_type]

        cols_data = df_raw.columns[1:]
        df_data = df_raw[cols_data]

        # New way of scalling - MinMaxScaler
        if self.scale:
            self.scaler = MinMaxScaler()
            train_data = df_data[border1s[0]:border2s[0]]
            self.scaler.fit(train_data.values)

            data = self.scaler.transform(df_data.values)
        else:
            data = df_data.values

        """
        # Original code snippet - commented out
        # This was responsible for scaling the data
        if self.scale:
            if self.scale_statistic is None:
                self.scaler = StandardScaler()
                train_data = df_data[border1s[0]:border2s[0]]
                self.scaler.fit(train_data.values)
            else:
                self.scaler = StandardScaler(mean = self.scale_statistic['mean'], std = self.scale_statist
            data = self.scaler.transform(df_data.values)
        else:
            data = df_data.values

        """

        self.data_x = data[border1:border2]
        self.data_y = data[border1:border2]
```

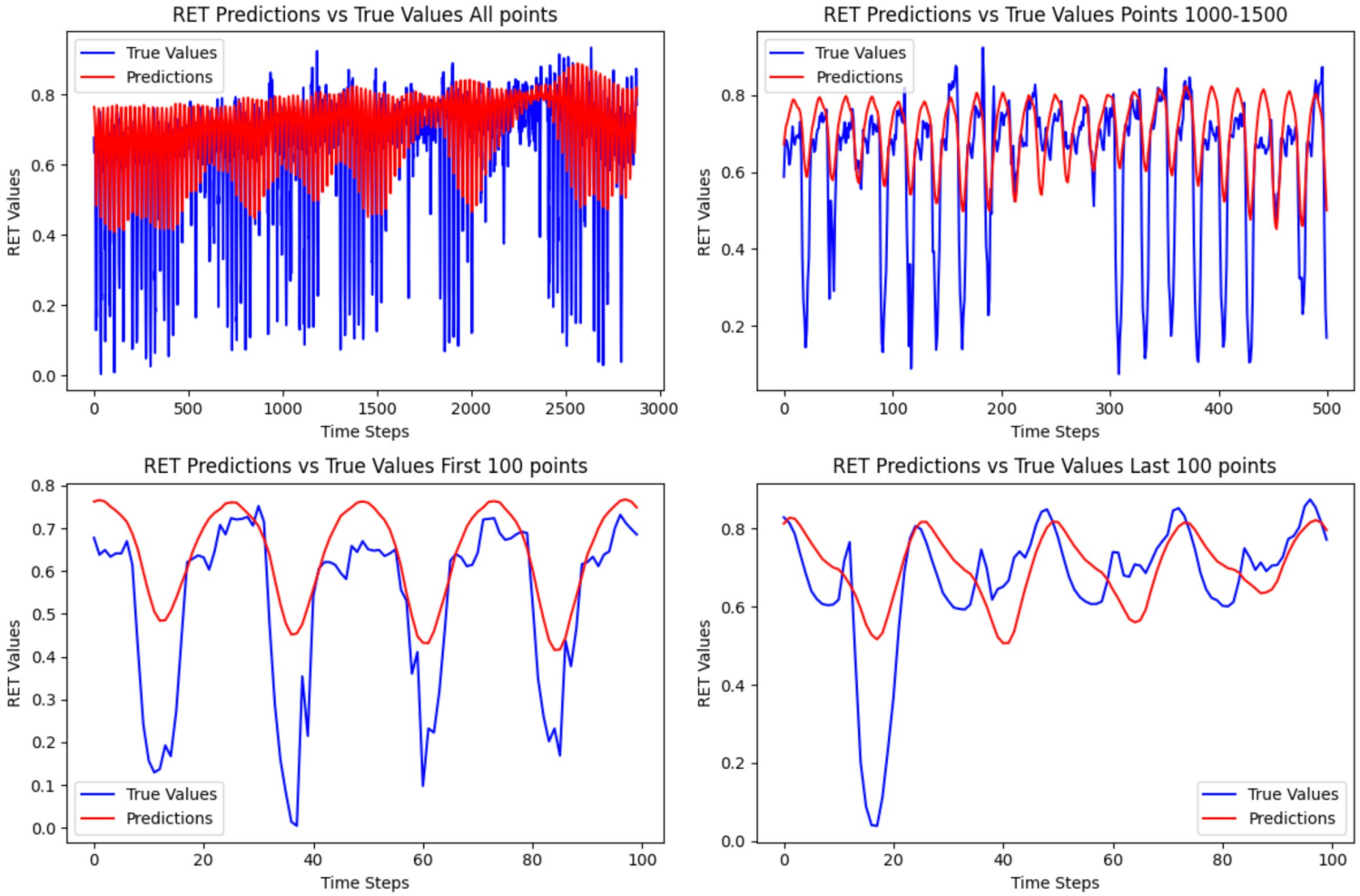
Change 2 - __read_data__ function in the data_loader.py

Description: transformed the data using the MinMaxSacler().

Sanity check
Plots of the output!
Result #3

- These plots are for dataset used by authors in original paper.
- We have just changed the Normalization to MinMaxScaler.
- Here we are only predicting for 1 timestamp in the future.

```
Predictions shape: (2880, 1)
True values shape: (2880, 1)
Selected setting: Crossformer_ETTh1_il168_ol1_sl6_win2_fa10_dm256_nh4_el3_itr0
```



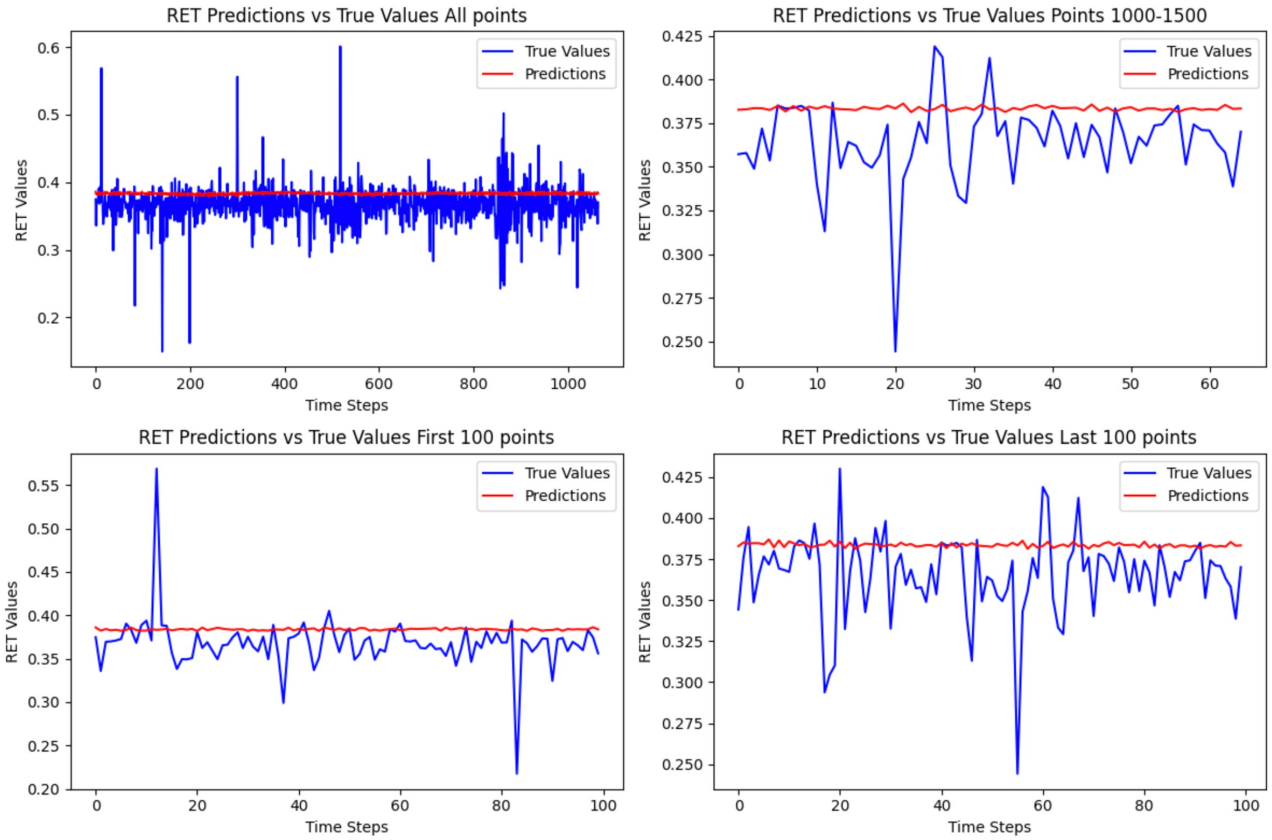
Inverse Implementation

```
207
208     def _process_one_batch(self, dataset_object, batch_x, batch_y, inverse = False):
209         batch_x = batch_x.float().to(self.device)
210         batch_y = batch_y.float().to(self.device)
211
212         outputs = self.model(batch_x)
213
214         ## Original code
215         # if inverse:
216         #     outputs = dataset_object.inverse_transform(outputs)
217         #     batch_y = dataset_object.inverse_transform(batch_y)
218
219         ## New code
220         if inverse:
221             original_shape_outputs = outputs.shape
222             outputs = outputs.detach().cpu().numpy().reshape(-1, original_shape_outputs[-1])
223             outputs = dataset_object.inverse_transform(outputs)
224             outputs = torch.from_numpy(outputs.reshape(original_shape_outputs)).to(self.device)
225
226             original_shape_batch_y = batch_y.shape
227             batch_y = batch_y.detach().cpu().numpy().reshape(-1, original_shape_batch_y[-1])
228             batch_y = dataset_object.inverse_transform(batch_y)
229             batch_y = torch.from_numpy(batch_y.reshape(original_shape_batch_y)).to(self.device)
230
231         return outputs, batch_y
```

Plot data – AKAM stock data 2023 | Result #4

Normalized Data

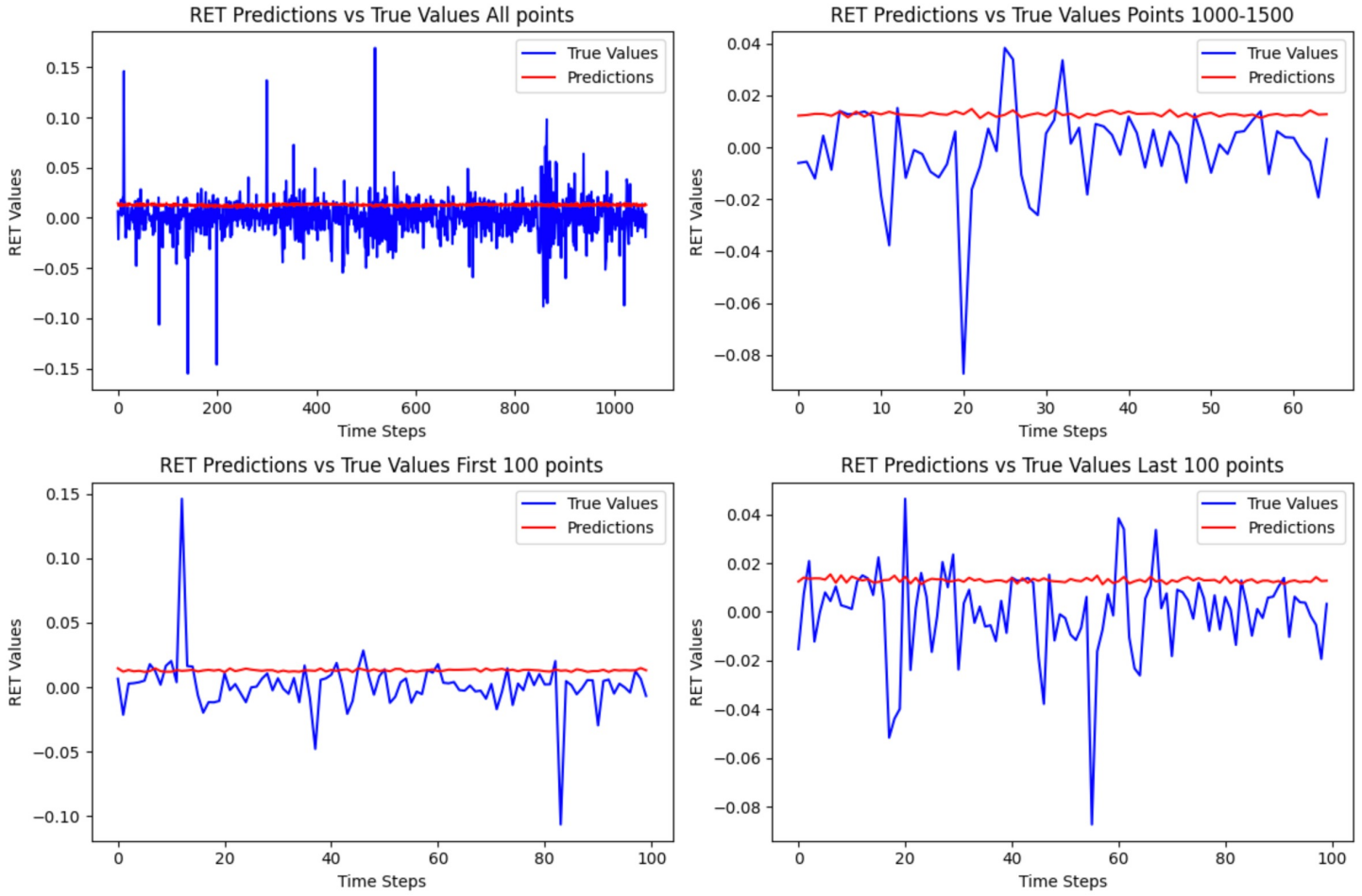
```
Predictions shape: (1065, 1)
True values shape: (1065, 1)
Selected setting: Crossformer_stock_AKAM_il168_ol1_sl6_win2_fa10_dm256_nh4_el3_itr0
```



Plot data – AKAM stock data 2023 | Result #5

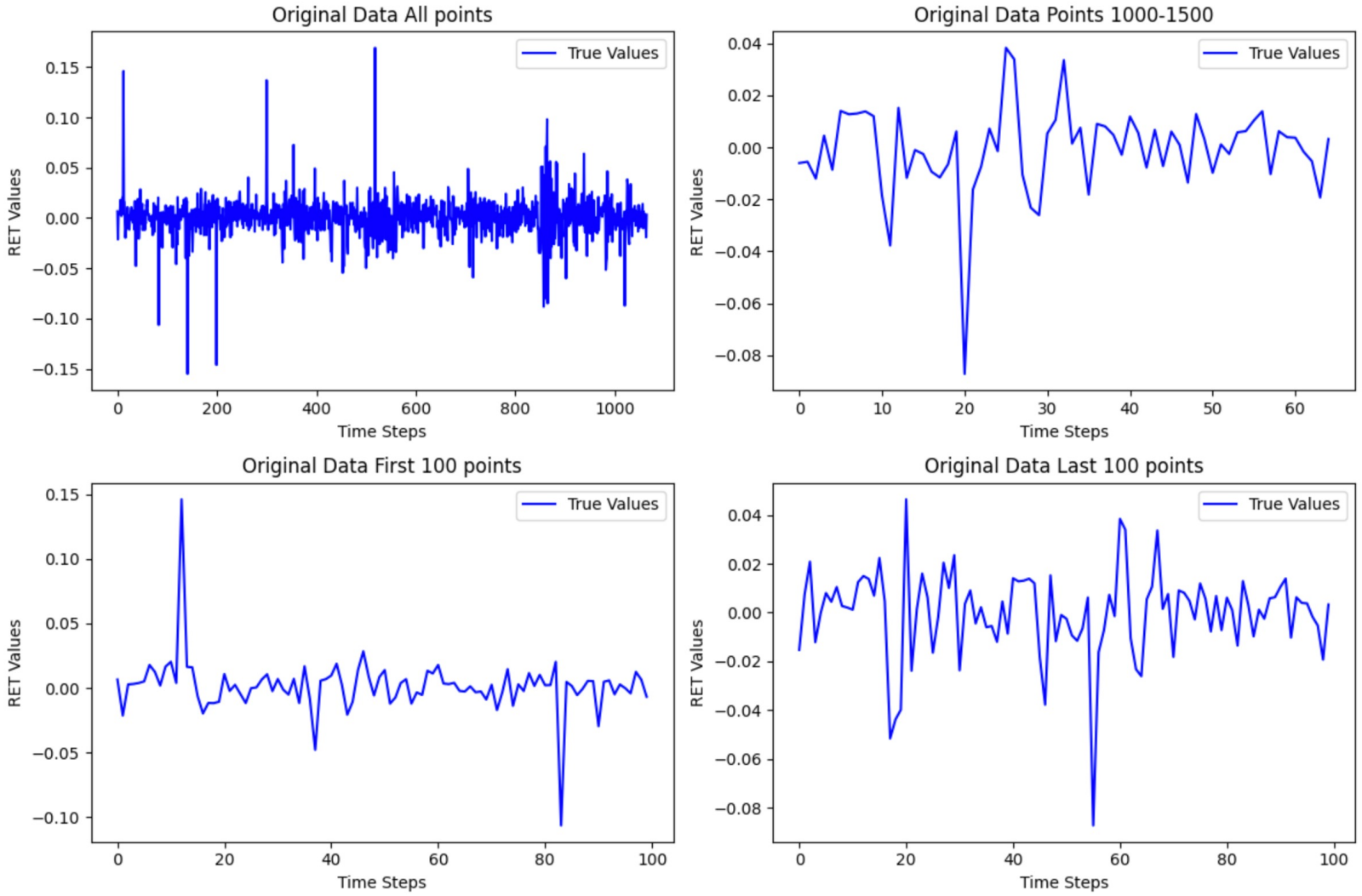
De-Normalized Data

```
Predictions shape: (1065, 1)
True values shape: (1065, 1)
Selected setting: Crossformer_stock_AKAM_il168_ol1_sl6_win2_fa10_dm256_nh4_el3_itr0
```



Original data – AKAM stock data 2023

Original Data



Results Screenshots

Result #1

```
(trade) md@Mohits-MacBook-Pro time_series_benchmarking % python crossformer/OriginalPaperCode/main_crossformer.py --data ETTh1 --in_len 168 --out_len 24 --seg_len 6 --itr 1
/Users/md/opt/anaconda3/envs/trade/lib/python3.9/site-packages/threadpoolctl.py:1214: RuntimeWarning:
Found Intel OpenMP ('libiomp') and LLVM OpenMP ('libomp') loaded at
the same time. Both libraries are known to be incompatible and this
can cause random crashes or deadlocks on Linux when loaded in the
same Python program.
Using threadpoolctl may cause crashes or deadlocks. For more
information and possible workarounds, please see
    https://github.com/joblib/threadpoolctl/blob/master/multiple_openmp.md

  warnings.warn(msg, RuntimeWarning)
Args in experiment:
Namespace(data='ETTh1', root_path='./datasets/', data_path='/Users/md/Library/CloudStorage/OneDrive-Personal/Mohit/RIT/trading/codebases/stockPredict/time_series_benchmarking/crossformer/Original
PaperCode/datasets/ETTh1.csv', data_split=[8640, 2880, 2880], checkpoints='./checkpoints/', in_len=168, out_len=24, seg_len=6, win_size=2, factor=10, data_dim=7, d_model=256, d_ff=512, n_heads=4,
 e_layers=3, dropout=0.2, baseline=False, num_workers=0, batch_size=32, train_epochs=1, patience=3, learning_rate=0.0001, lradj='type1', itr=1, save_pred=False, use_gpu=False, gpu=0, use_multi_gp
u=False, devices='0,1,2,3')
Use CPU
>>>>>>>start training : Crossformer_ETTh1_il168_ol24_sl6_win2_fa10_dm256_nh4_el3_itr0>>>>>>>>>>>>>>>>>>>>>>>>>
train 8449
val 2857
test 2857
        iters: 100, epoch: 1 | loss: 0.4136730
        speed: 1.5291s/iter; left time: 253.8256s
        iters: 200, epoch: 1 | loss: 0.3579630
        speed: 1.5232s/iter; left time: 100.5308s
Epoch: 1 cost time: 374.6694848537445
Epoch: 1, Steps: 265 | Train Loss: 0.4415135 Vali Loss: 0.4979840 Test Loss: 0.3438014
Validation loss decreased (inf --> 0.497984).  Saving model ...
>>>>>>>testing : Crossformer_ETTh1_il168_ol24_sl6_win2_fa10_dm256_nh4_el3_itr0<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
test 2857
mse:0.34453585743904114, mae:0.3929487466812134
(trade) md@Mohits-MacBook-Pro time_series_benchmarking % Python crossformer/OriginalPaperCode/eval_crossformer.py --checkpoint_root ./checkpoints --setting_name Crossformer_ETTh1_il168_ol24_sl6_w
in2_fa10_dm256_nh4_el3_itr0
/Users/md/opt/anaconda3/envs/trade/lib/python3.9/site-packages/threadpoolctl.py:1214: RuntimeWarning:
Found Intel OpenMP ('libiomp') and LLVM OpenMP ('libomp') loaded at
the same time. Both libraries are known to be incompatible and this
can cause random crashes or deadlocks on Linux when loaded in the
same Python program.
Using threadpoolctl may cause crashes or deadlocks. For more
information and possible workarounds, please see
    https://github.com/joblib/threadpoolctl/blob/master/multiple_openmp.md

  warnings.warn(msg, RuntimeWarning)
Use CPU
mse:0.34453585743904114, mae:0.3929487466812134
```

Result #2

```
(trade) md@Mohits-MacBook-Pro time_series_benchmarking % python crossformer/OriginalPaperCode/main_crossformer.py --data ETTh1 --in_len 168 --out_len 24 --seg_len 6 --itr 1
/Users/md/opt/anaconda3/envs/trade/lib/python3.9/site-packages/threadpoolctl.py:1214: RuntimeWarning:
Found Intel OpenMP ('libiomp') and LLVM OpenMP ('libomp') loaded at
the same time. Both libraries are known to be incompatible and this
can cause random crashes or deadlocks on Linux when loaded in the
same Python program.
Using threadpoolctl may cause crashes or deadlocks. For more
information and possible workarounds, please see
    https://github.com/joblib/threadpoolctl/blob/master/multiple_openmp.md

  warnings.warn(msg, RuntimeWarning)
Args in experiment:
Namespace(data='ETTh1', root_path='./datasets/', data_path='/Users/md/Library/CloudStorage/OneDrive-Personal/Mohit/RIT/trading/codebases/stockPredict/time_series_benchmarking/crossformer/Original
PaperCode/datasets/ETTh1.csv', data_split=[8640, 2880, 2880], checkpoints='./checkpoints/', in_len=168, out_len=24, seg_len=6, win_size=2, factor=10, data_dim=7, d_model=256, d_ff=512, n_heads=4,
 e_layers=3, dropout=0.2, baseline=False, num_workers=0, batch_size=32, train_epochs=1, patience=3, learning_rate=0.0001, lradj='type1', itr=1, save_pred=False, use_gpu=False, gpu=0, use_multi_gp
u=False, devices='0,1,2,3')
Use CPU
>>>>>>>start training : Crossformer_ETTh1_il168_ol24_sl6_win2_fa10_dm256_nh4_el3_itr0>>>>>>>>>>>>>>>>>>>>>>>>
train 8449
val 2857
test 2857
        iters: 100, epoch: 1 | loss: 0.0498740
        speed: 1.3402s/iter; left time: 222.4775s
        iters: 200, epoch: 1 | loss: 0.0312173
        speed: 1.2864s/iter; left time: 84.9055s
Epoch: 1 cost time: 343.82427287101746
Epoch: 1, Steps: 265 | Train Loss: 0.0662261 Vali Loss: 0.0390280 Test Loss: 0.0310992
Validation loss decreased (inf --> 0.039028).  Saving model ...
>>>>>>>testing : Crossformer_ETTh1_il168_ol24_sl6_win2_fa10_dm256_nh4_el3_itr0<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
test 2857
mse:0.031207000836730003, mae:0.13691513240337372
(trade) md@Mohits-MacBook-Pro time_series_benchmarking % Python crossformer/OriginalPaperCode/eval_crossformer.py --checkpoint_root ./checkpoints --setting_name Crossformer_ETTh1_il168_ol24_sl6_w
in2_fa10_dm256_nh4_el3_itr0
/Users/md/opt/anaconda3/envs/trade/lib/python3.9/site-packages/threadpoolctl.py:1214: RuntimeWarning:
Found Intel OpenMP ('libiomp') and LLVM OpenMP ('libomp') loaded at
the same time. Both libraries are known to be incompatible and this
can cause random crashes or deadlocks on Linux when loaded in the
same Python program.
Using threadpoolctl may cause crashes or deadlocks. For more
information and possible workarounds, please see
    https://github.com/joblib/threadpoolctl/blob/master/multiple_openmp.md

  warnings.warn(msg, RuntimeWarning)
Use CPU
mse:0.031207000836730003, mae:0.13691513240337372
```

Result #3

```
(trade) md@Mohits-MacBook-Pro time_series_benchmarking % python crossformer/OriginalPaperCode/main_crossformer.py --data ETTh1 --in_len 168 --out_len 1 --seg_len 6 --itr 1
/Users/md/opt/anaconda3/envs/trade/lib/python3.9/site-packages/threadpoolctl.py:1214: RuntimeWarning:
Found Intel OpenMP ('libiomp') and LLVM OpenMP ('libomp') loaded at
the same time. Both libraries are known to be incompatible and this
can cause random crashes or deadlocks on Linux when loaded in the
same Python program.
Using threadpoolctl may cause crashes or deadlocks. For more
information and possible workarounds, please see
    https://github.com/joblib/threadpoolctl/blob/master/multiple_openmp.md

  warnings.warn(msg, RuntimeWarning)
Args in experiment:
Namespace(data='ETTh1', root_path='./datasets/', data_path='/Users/md/Library/CloudStorage/OneDrive-Personal/Mohit/RIT/trading/codebases/stockPredict/time_series_benchmarking/crossformer/Original
PaperCode/datasets/ETTh1.csv', data_split=[8640, 2880, 2880], checkpoints='./checkpoints/', in_len=168, out_len=1, seg_len=6, win_size=2, factor=10, data_dim=7, d_model=256, d_ff=512, n_heads=4,
e_layers=3, dropout=0.2, baseline=False, num_workers=0, batch_size=32, train_epochs=1, patience=3, learning_rate=0.0001, lradj='type1', itr=1, save_pred=False, use_gpu=False, gpu=0, use_multi_gpu
=False, devices='0,1,2,3')
Use CPU
>>>>>>>start training : Crossformer_ETTh1_il168_ol1_sl6_win2_fa10_dm256_nh4_el3_itr0>>>>>>>>>>>>>>>>>>>>>>>>>
train 8472
val 2880
test 2880
        iters: 100, epoch: 1 | loss: 0.0534162
        speed: 1.5196s/iter; left time: 252.2523s
        iters: 200, epoch: 1 | loss: 0.0435354
        speed: 1.1388s/iter; left time: 75.1637s
Epoch: 1 cost time: 333.4297790527344
Epoch: 1, Steps: 265 | Train Loss: 0.1093438 Vali Loss: 0.0176277 Test Loss: 0.0113550
Validation loss decreased (inf --> 0.017628).  Saving model ...
>>>>>>>testing : Crossformer_ETTh1_il168_ol1_sl6_win2_fa10_dm256_nh4_el3_itr0<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
test 2880
mse:0.01135498657822609, mae:0.07314461469650269
(trade) md@Mohits-MacBook-Pro time_series_benchmarking % Python crossformer/OriginalPaperCode/eval_crossformer.py --checkpoint_root ./checkpoints --setting_name Crossformer_ETTh1_il168_ol1_sl6_wi
n2_fa10_dm256_nh4_el3_itr0
/Users/md/opt/anaconda3/envs/trade/lib/python3.9/site-packages/threadpoolctl.py:1214: RuntimeWarning:
Found Intel OpenMP ('libiomp') and LLVM OpenMP ('libomp') loaded at
the same time. Both libraries are known to be incompatible and this
can cause random crashes or deadlocks on Linux when loaded in the
same Python program.
Using threadpoolctl may cause crashes or deadlocks. For more
information and possible workarounds, please see
    https://github.com/joblib/threadpoolctl/blob/master/multiple_openmp.md

  warnings.warn(msg, RuntimeWarning)
Use CPU
mse:0.01135498657822609, mae:0.07314461469650269
```

Result #4

```
(trade) md@Mohits-MacBook-Pro time_series_benchmarking % python crossformer/OriginalPaperCode/main_crossformer.py --data stock_AKAM --in_len 168 --out_len 1 --seg_len 6 --itr 1
/Users/md/opt/anaconda3/envs/trade/lib/python3.9/site-packages/threadpoolctl.py:1214: RuntimeWarning:
Found Intel OpenMP ('libiomp') and LLVM OpenMP ('libomp') loaded at
the same time. Both libraries are known to be incompatible and this
can cause random crashes or deadlocks on Linux when loaded in the
same Python program.
Using threadpoolctl may cause crashes or deadlocks. For more
information and possible workarounds, please see
    https://github.com/joblib/threadpoolctl/blob/master/multiple_openmp.md

  warnings.warn(msg, RuntimeWarning)
Args in experiment:
Namespace(data='stock_AKAM', root_path='./datasets/', data_path='/Users/md/Library/CloudStorage/OneDrive-Personal/Mohit/RIT/trading/codebases/stockPredict/time_series_benchmarking/crossformer/Ori
ginalPaperCode/datasets/my_data/AKAM_filtered_data.csv', data_split=[0.7, 0.1, 0.2], checkpoints='./crossformer/OriginalPaperCode/checkpoints/', in_len=168, out_len=1, seg_len=6, win_size=2, fact
or=10, data_dim=6, d_model=256, d_ff=512, n_heads=4, e_layers=3, dropout=0.2, baseline=False, num_workers=0, batch_size=32, train_epochs=1, patience=3, learning_rate=0.0001, lradj='type1', itr=1,
 save_pred=True, use_gpu=False, gpu=0, use_multi_gpu=False, devices='0,1,2,3')
Use CPU
>>>>>>>start training : Crossformer_stock_AKAM_il168_ol1_sl6_win2_fa10_dm256_nh4_el3_itr0>>>>>>>>>>>>>>>>>>>>>>>>>
train 3560
val 534
test 1065
        iters: 100, epoch: 1 | loss: 0.0506057
        speed: 0.9407s/iter; left time: 12.2297s
Epoch: 1 cost time: 105.32556796073914
Epoch: 1, Steps: 112 | Train Loss: 0.1699368 Vali Loss: 0.0023518 Test Loss: 0.0031919
Validation loss decreased (inf --> 0.002352).  Saving model ...
>>>>>>>testing : Crossformer_stock_AKAM_il168_ol1_sl6_win2_fa10_dm256_nh4_el3_itr0<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
test 1065
mse:0.0030088727362453938, mae:0.03436887636780739
```

Result #5

```
(trade) md@Mohits-MacBook-Pro time_series_benchmarking % Python crossformer/OriginalPaperCode/eval_crossformer.py --checkpoint_root ./checkpoints --setting_name Crossformer_stock_AKAM_il168_ol1_sl6_win2_fa10_dm256_nh4_el3_itr0
/Users/md/opt/anaconda3/envs/trade/lib/python3.9/site-packages/threadpoolctl.py:1214: RuntimeWarning:
Found Intel OpenMP ('libiomp') and LLVM OpenMP ('libomp') loaded at
the same time. Both libraries are known to be incompatible and this
can cause random crashes or deadlocks on Linux when loaded in the
same Python program.
Using threadpoolctl may cause crashes or deadlocks. For more
information and possible workarounds, please see
    https://github.com/joblib/threadpoolctl/blob/master/multiple_openmp.md

  warnings.warn(msg, RuntimeWarning)
Args: Namespace(checkpoint_root='./checkpoints', setting_name='Crossformer_stock_AKAM_il168_ol1_sl6_win2_fa10_dm256_nh4_el3_itr0', num_workers=0, batch_size=32, different_split=False, data_split='0.7,0.1,0.2', inverse=True, save_pred=True, use_gpu=False, gpu=0, use_multi_gpu=False)
Args: Namespace(checkpoint_root='./crossformer/OriginalPaperCode/checkpoints/', setting_name='Crossformer_stock_AKAM_il168_ol1_sl6_win2_fa10_dm256_nh4_el3_itr0', num_workers=0, batch_size=32, different_split=False, data_split='0.7,0.1,0.2', inverse=True, save_pred=True, use_gpu=False, gpu=0, use_multi_gpu=False)
Use CPU
mse:128.7794647216797, mae:4.484711170196533
```