Blockhouse Work Trial Report

Sai Sena Chinnakonduru

Saisena314@gmail.com

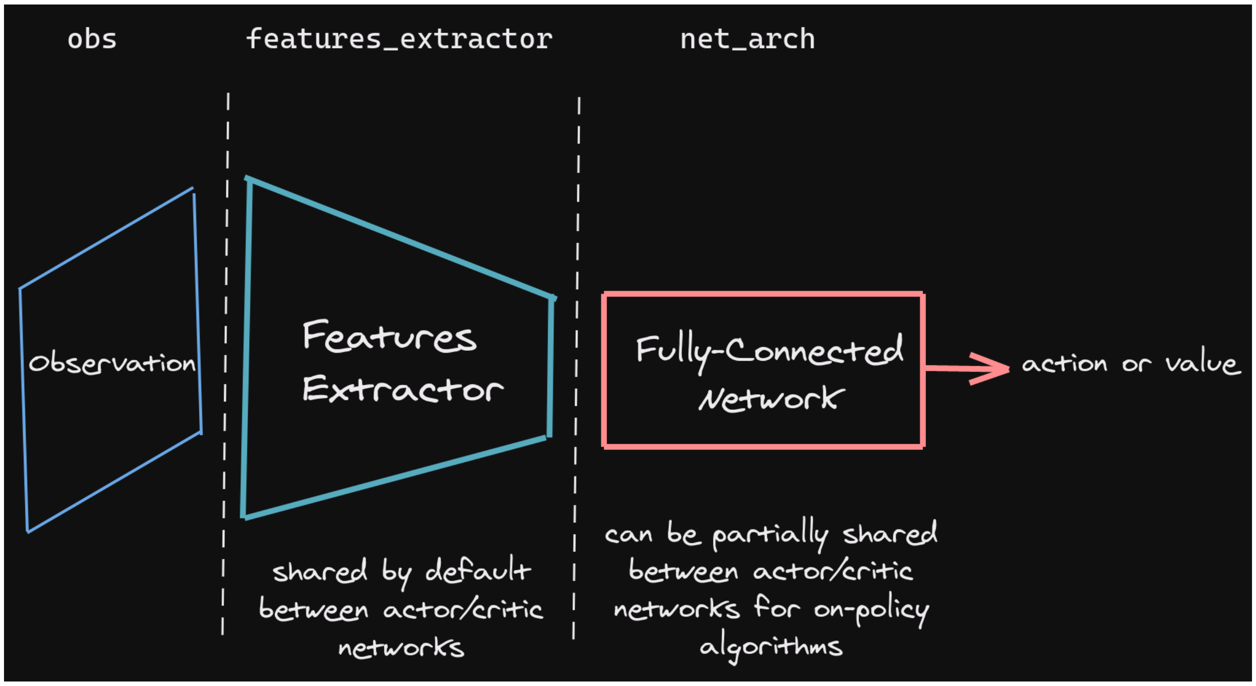
https://github.com/saisena-ss/Transformer-Agent-for-Trade-Optimization

Objective:

The goal of this task is to use a transformer to generate appropriate signals for buy, sell and hold for trade optimization.

Implementation:

The given code has the implementation of PPO method to generate signals. My idea of using transformer is to use it as a feature extractor in the PPO model as shown in the figure below.


obs
features_extractor
net_arch
Observation
Features Extractor
Fully-Connected Network
action or value
shared by default between actor/critic networks
can be partially shared between actor/critic networks for on-policy algorithms

The default PPO model uses a simple neural network as a feature extractor. I replaced this with a custom transformer model that learns features by paying attention to previous time ticks (the number of time ticks depends on the configuration). By analyzing previous time ticks, the model can learn patterns and predict future prices or volumes.

Transformers are primarily designed for language tasks, so I modified the architecture accordingly. I removed the token embedding layer and replaced it with a linear layer that takes each tick's features as input and projects them to the n_embed dimensions. Below is the model configuration:

- n_embed dimension: 1024
- block_size (number of time ticks): 20
- n_layers (number of attention blocks): 6

- output_dim: 256
- n_head(number of heads in attention layer): 16

I used a cosine scheduler for the learning rate, initialized at 3e-4, and ran all models for 10,000 timesteps.

Results:

I evaluated three models on both train and test sets:

1. Given PPO model
2. PPO model with transformer as a feature extractor
3. Simple trading blotter (provided)

For training, I used the provided file. For testing, to check how well the model generalizes, I extracted data for "AAPL" (May '24) from Databento and evaluated it on 10,000 records (file uploaded on GitHub).Below are the results:

Train Set:

Metric	Base PPO	PPO – with Transformer	Trading Blotter
Balance	$ 38,311	$9,969,528.95	-
Shares held	51,450	160	-
Total shares traded	51,450	160	-
Total portfolio value	$ 10,022,184.24	$10,000,576.95	$10,038,906.85
Cumulative reward	- 9,437.92	-2,950.75	-11,806.66
Total ticks in the data		59,236	
# ticks with buy or sell generated from model	11,329	12,858	32,035

Test Set:

Metric	Base PPO	PPO – with Transformer	Trading Blotter
Balance	$9,891,718.07	$9,737,182.04	-
Shares held	313	1377	-
Total shares traded	313	1377	-
Total portfolio value	$9,951,407.17	$9,999,775.94	$9,979,032.57
Cumulative reward	-1,248.42	-52.85	-1,196.80
Total ticks in the data		9,965	
# ticks with buy or sell generated from model	5,983	226	3,181

The transformer-based PPO model demonstrates a significant enhancement over the base PPO model in generating trade signals (for trade optimization). The results show that the transformer-based approach's reward is maximum for both training data and unseen test data. By leveraging the attention mechanism inherent in transformer architectures, the model effectively captures temporal dependencies and patterns in market data, which are crucial for making informed trading decisions.

Fine-tuning:

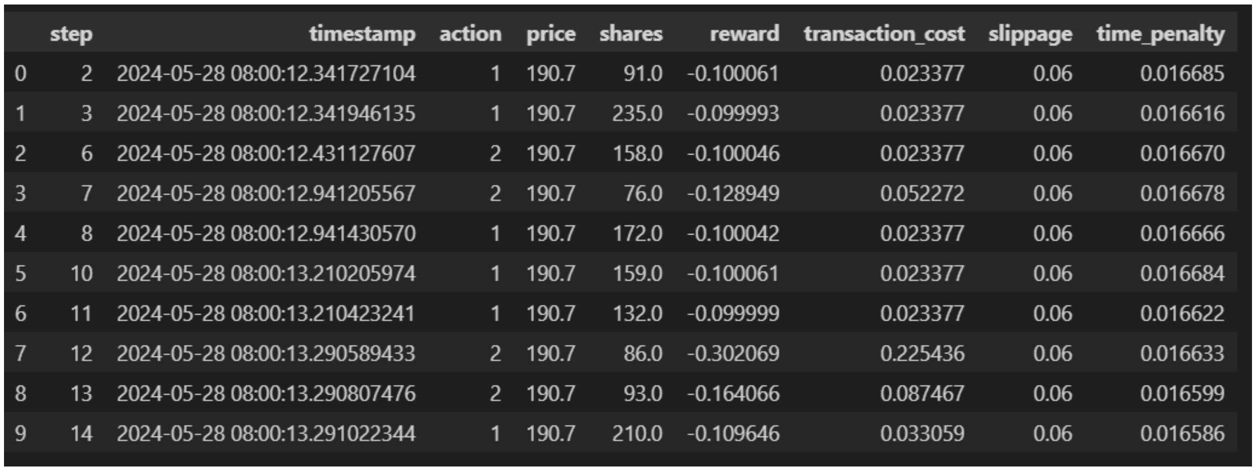
For transformer, I scaled the input features using log transformation and implementing robust standard scaler. I experimented with different configurations of n_embed dimensions (512, 1024) and block size (ranging from 5 to 300 ticks). For all configurations, the transformer model as a feature extractor consistently outperformed the simple RL model and the trading blotter. The final hyperparameters were chosen to balance both cumulative reward and portfolio balance. These parameters can be further tweaked to achieve better results tailored to specific scenarios, such as trade optimization or portfolio optimization.

My custom transformer code is located in customtransformer.py, and its configuration is set in the main notebook (RL PPO v2.ipynb) using the config class.

Examples:

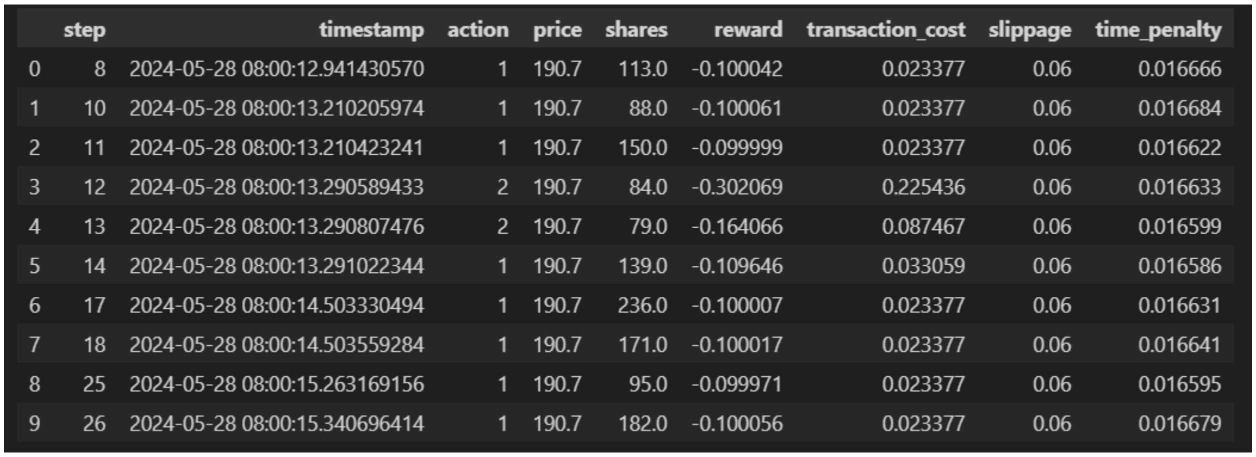
All the examples are with test_data.csv

PPO Model:



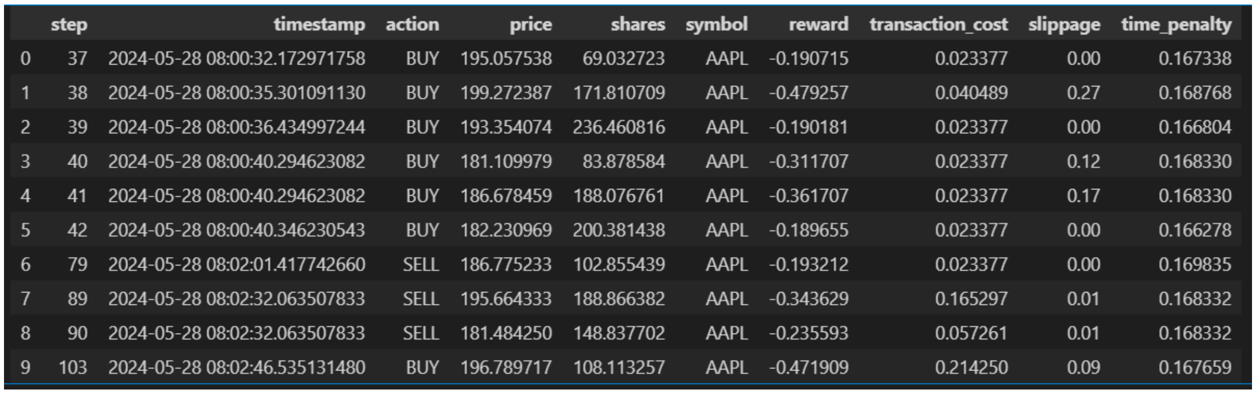
	step	timestamp	action	price	shares	reward	transaction_cost	slippage	time_penalty
0	2	2024-05-28 08:00:12.341727104	1	190.7	91.0	-0.100061	0.023377	0.06	0.016685
1	3	2024-05-28 08:00:12.341946135	1	190.7	235.0	-0.099993	0.023377	0.06	0.016616
2	6	2024-05-28 08:00:12.431127607	2	190.7	158.0	-0.100046	0.023377	0.06	0.016670
3	7	2024-05-28 08:00:12.941205567	2	190.7	76.0	-0.128949	0.052272	0.06	0.016678
4	8	2024-05-28 08:00:12.941430570	1	190.7	172.0	-0.100042	0.023377	0.06	0.016666
5	10	2024-05-28 08:00:13.210205974	1	190.7	159.0	-0.100061	0.023377	0.06	0.016684
6	11	2024-05-28 08:00:13.210423241	1	190.7	132.0	-0.099999	0.023377	0.06	0.016622
7	12	2024-05-28 08:00:13.290589433	2	190.7	86.0	-0.302069	0.225436	0.06	0.016633
8	13	2024-05-28 08:00:13.290807476	2	190.7	93.0	-0.164066	0.087467	0.06	0.016599
9	14	2024-05-28 08:00:13.291022344	1	190.7	210.0	-0.109646	0.033059	0.06	0.016586

PPO with transformer as a feature extractor:



	step	timestamp	action	price	shares	reward	transaction_cost	slippage	time_penalty
0	8	2024-05-28 08:00:12.941430570	1	190.7	113.0	-0.100042	0.023377	0.06	0.016666
1	10	2024-05-28 08:00:13.210205974	1	190.7	88.0	-0.100061	0.023377	0.06	0.016684
2	11	2024-05-28 08:00:13.210423241	1	190.7	150.0	-0.099999	0.023377	0.06	0.016622
3	12	2024-05-28 08:00:13.290589433	2	190.7	84.0	-0.302069	0.225436	0.06	0.016633
4	13	2024-05-28 08:00:13.290807476	2	190.7	79.0	-0.164066	0.087467	0.06	0.016599
5	14	2024-05-28 08:00:13.291022344	1	190.7	139.0	-0.109646	0.033059	0.06	0.016586
6	17	2024-05-28 08:00:14.503330494	1	190.7	236.0	-0.100007	0.023377	0.06	0.016631
7	18	2024-05-28 08:00:14.503559284	1	190.7	171.0	-0.100017	0.023377	0.06	0.016641
8	25	2024-05-28 08:00:15.263169156	1	190.7	95.0	-0.099971	0.023377	0.06	0.016595
9	26	2024-05-28 08:00:15.340696414	1	190.7	182.0	-0.100056	0.023377	0.06	0.016679

Trading Blotter



	step	timestamp	action	price	shares	symbol	reward	transaction_cost	slippage	time_penalty
0	37	2024-05-28 08:00:32.172971758	BUY	195.057538	69.032723	AAPL	-0.190715	0.023377	0.00	0.167338
1	38	2024-05-28 08:00:35.301091130	BUY	199.272387	171.810709	AAPL	-0.479257	0.040489	0.27	0.168768
2	39	2024-05-28 08:00:36.434997244	BUY	193.354074	236.460816	AAPL	-0.190181	0.023377	0.00	0.166804
3	40	2024-05-28 08:00:40.294623082	BUY	181.109979	83.878584	AAPL	-0.311707	0.023377	0.12	0.168330
4	41	2024-05-28 08:00:40.294623082	BUY	186.678459	188.076761	AAPL	-0.361707	0.023377	0.17	0.168330
5	42	2024-05-28 08:00:40.346230543	BUY	182.230969	200.381438	AAPL	-0.189655	0.023377	0.00	0.166278
6	79	2024-05-28 08:02:01.417742660	SELL	186.775233	102.855439	AAPL	-0.193212	0.023377	0.00	0.169835
7	89	2024-05-28 08:02:32.063507833	SELL	195.664333	188.866382	AAPL	-0.343629	0.165297	0.01	0.168332
8	90	2024-05-28 08:02:32.063507833	SELL	181.484250	148.837702	AAPL	-0.235593	0.057261	0.01	0.168332
9	103	2024-05-28 08:02:46.535131480	BUY	196.789717	108.113257	AAPL	-0.471909	0.214250	0.09	0.167659

Conclusion:

The transformer-based PPO model effectively enhances the performance of trade signal generation compared to the base PPO model and the simple trading blotter. By capturing temporal dependencies and patterns in the data, the transformer-based model provides more accurate and actionable trade recommendations. This implementation can be further improved with additional fine-tuning and tailored to specific trading objectives, offering a robust solution for trade optimization.

The detailed implementation, fine-tuning process, and evaluation results are thoroughly documented in the accompanying GitHub repository. The repository includes all code files, configuration details, and comprehensive results from the training and testing phases, providing a clear and accessible overview of the project.