Third agent: ***Trading Agent.***

The third agent is an intelligent trading agent that learns a policy π to map an observed state $\mathbf{s}_{t-1}$ to an optimal trading action a_t. The goal is to maximize a cumulative reward signal over time. This learning problem is formulated as a Markov Decision Process (MDP) and solved using a Reinforcement Learning (RL) approach, specifically, a Deep Q-Network (DQN).

State Space. At the beginning of each time step t, before an action is taken, the agent observes a state vector $\mathbf{s}_{t-1} \in \mathcal{S}$. This vector is constructed from information available up to the end of period $t-1$:

$$\mathbf{s}_{t-1} = \left[CMI_{t-1}^{\text{trgt}|\text{synth}}, CMI_{t-1}^{\text{synth}|\text{trgt}}, a_{t-1}, \tau_{t-1}^{\text{norm}}, \{r_\ell^{\text{trgt}}\}_{\ell=t-w}^{t-1}, \{r_\ell^{\text{synth}}\}_{\ell=t-w}^{t-1}\right]^\top$$

where:

- $CMI_{t-1}^{\text{trgt}|\text{synth}}$ and $CMI_{t-1}^{\text{synth}|\text{trgt}}$ are the cumulative mispricing indices calculated after observing returns at $t-1$, as detailed by the second agent. In the implementation, these values are typically bounded (e.g., within $[-20, 20]$).
- $a_{t-1} \in \{-1, 0, 1\}$ is the trading position taken by the agent in the previous period, $t-1$.
- τ_{t-1}^{norm} is the normalized duration for which the position a_{t-1} has been held. It is computed as $\min(\text{dur}(a_{t-1})/H, 1.0)$, where $\text{dur}(a_{t-1})$ is the number of consecutive periods a_{t-1} has been active, and H is a normalization horizon (e.g., $H = 100$ periods in the code). If $a_{t-1} = 0$, $\text{dur}(a_{t-1}) = 0$. If $a_{t-1} \neq 0$ and was initiated at $t-1$, $\text{dur}(a_{t-1}) = 1$.
- $\{r_\ell^{\text{trgt}}\}_{\ell=t-w}^{t-1}$ and $\{r_\ell^{\text{synth}}\}_{\ell=t-w}^{t-1}$ are sequences of w most recent log-returns for the target and synthetic assets, respectively, up to period $t-1$. The window size w is a hyperparameter (e.g., $w = 100$). These returns are also typically bounded in the state representation (e.g., within $[-2, 2]$).

Action Space. Based on the observed state $\mathbf{s}_{t-1}$, the agent selects an action $a_t \in \mathcal{A} = \{-1, 0, 1\}$ for period t:

- $a_t = 1$: Long position on the target asset, short position on the synthetic asset.
- $a_t = -1$: Short position on the target asset, long position on the synthetic asset.
- $a_t = 0$: Neutral (no position).

Reward Function. After action a_t is taken, the market returns r_t^{trgt} and r_t^{synth} for period t are realized. The agent's performance is evaluated through a reward signal R_t. The fundamental component of this reward is the portfolio return for period t, $r_t^{\mathcal{P}}$, which accounts for trading profits and transaction costs:

$$r_t^{\mathcal{P}} = a_t(r_t^{\text{trgt}} - r_t^{\text{synth}}) - \kappa|a_t - a_{t-1}|$$

Here, κ represents the per-unit transaction cost rate (e.g., $\kappa = 0.005$ or 0.5% in the implementation), applied to the absolute change in position size (since positions are unit-sized, this is effectively on entering or exiting/flipping a position).

While $r_t^{\mathcal{P}}$ itself can be used as a reward, the implementation primarily employs a reward R_t based on the rolling Sharpe ratio of the portfolio returns. This encourages the agent to learn a policy that yields consistent risk-adjusted returns. The reward is defined as:

$$R_t = \begin{cases} \frac{\text{mean}(\{r_k^{\mathcal{P}}\}_{k=t-W_S+1}^t)}{\text{std}(\{r_k^{\mathcal{P}}\}_{k=t-W_S+1}^t)+\epsilon_{\text{stab}}} \sqrt{N_{\text{ann}}} \cdot c_{\text{scale}} & \text{if } N_h \geq W_S \\ r_t^{\mathcal{P}} & \text{if } N_h < W_S \end{cases}$$

where $\{r_k^{\mathcal{P}}\}_{k=t-W_S+1}^t$ is the series of the W_S most recent portfolio returns (with $W_S = 100$ in the code), N_h is the count of available historical portfolio returns, ϵ_{stab} is a small constant for numerical stability (e.g., 10^{-6}), N_{ann} is an annualization factor (e.g., 252 for daily data), and c_{scale} is a scaling factor (e.g., 0.01) to adjust the reward magnitude. If fewer than W_S historical portfolio returns are available, the reward defaults to the current period's portfolio return $r_t^{\mathcal{P}}$. The framework also supports other reward structures, such as raw returns or returns penalized by maximum drawdown.

Learning Algorithm: Deep Q-Network (DQN). The trading agent utilizes a Deep Q-Network (DQN) algorithm **?** to learn the optimal trading policy. DQN aims to approximate the optimal action-value function, $Q^*(\mathbf{s}, a)$, which represents the maximum expected cumulative discounted future reward obtainable by taking action a in state $\mathbf{s}$ and following the optimal policy thereafter:

$$Q^*(\mathbf{s}, a) = \mathbb{E}_{\pi^*}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid \mathbf{s}_t = \mathbf{s}, a_t = a\right]$$

where $\gamma \in [0, 1]$ is the discount factor (set to 0.99 in the implementation), which prioritizes immediate versus future rewards.

The Q-function is parameterized by a neural network, $Q(\mathbf{s}, a; \theta)$ (a Multi-Layer Perceptron, MLP, in this case). The network parameters θ are learned by minimizing the temporal difference error, typically using a loss function based on the Bellman equation:

$$L(\theta) = \mathbb{E}_{(\mathbf{s}_j, a_j, R_{j+1}, \mathbf{s}_{j+1}) \sim \mathcal{B}}\left[(Y_j - Q(\mathbf{s}_j, a_j; \theta))^2\right]$$

where $Y_j = R_{j+1} + \gamma \max_{a'} Q(\mathbf{s}_{j+1}, a'; \theta^-)$ is the target value. Key components of the DQN implementation include:

- **Experience Replay**: Transitions $(\mathbf{s}_j, a_j, R_{j+1}, \mathbf{s}_{j+1})$ are stored in a replay buffer $\mathcal{B}$ (e.g., size 50,000). Training samples are drawn randomly from this buffer, which helps to break correlations between consecutive samples and improves learning stability.
- **Target Network**: A separate target network $Q(\cdot, \cdot; \theta^-)$ is used to generate the target values Y_j. The weights θ^- of this target network are periodically updated with the weights θ of the online Q-network (e.g., every 1,000 training steps), providing a stable learning target.
- **Optimization**: The online network's weights θ are updated using an optimizer (e.g., Adam) with a specified learning rate (e.g., 10^{-4}). Updates are typically performed on mini-batches sampled from the replay buffer (e.g., batch size of 64).
- **Exploration Strategy**: To ensure adequate exploration of the state-action space, an ϵ-greedy policy is employed during training. With probability ϵ, a random action is selected; otherwise, the action with the highest Q-value is chosen: $a_t = \arg\max_a Q(\mathbf{s}_{t-1}, a; \theta)$. The value of ϵ is typically annealed from an initial value (e.g., 1.0) to a final value (e.g., 0.05) over a portion of the training steps (e.g., the first 20% of total timesteps).

Through iterative interaction with the trading environment and updates to its Q-network, the agent learns a policy $\pi(\mathbf{s}) = \arg\max_a Q(\mathbf{s}, a; \theta)$ that aims to maximize long-term cumulative rewards.

We implement a Deep Q-Network (DQN) to learn the optimal trading policy. The Q-function approximator $Q(s, a; \theta)$ is parameterized by a multilayer perceptron with parameters θ. The learning objective is to minimize the loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right]$$

where $\mathcal{D}$ is the replay buffer, γ is the discount factor, and θ^- are the parameters of the target network.

Key hyperparameters of our implementation include:

- Learning rate: 10^{-4}
- Buffer size: 50,000 transitions
- Exploration strategy: ϵ-greedy with linear decay from 1.0 to 0.05

- Batch size: 64
- Discount factor (γ): 0.99
- Target network update frequency: 1,000 steps

Reward Function Our reward for period t is R_t