Adaptive Quantitative Trading with Imitative Recurrent Deterministic Policy Gradient

Elio Samaha, Lara Hofman, Mattia Martino, Sandro Mikautadze

March 18, 2025

***Abstract*—This report investigates the application of imitative Recurrent Deterministic Policy Gradient (iRDPG) to the problem of quantitative trading (QT) in high-frequency financial markets. We adapt the Recurrent Deterministic Policy Gradient (RDPG) framework, enhanced with imitation learning techniques, to develop an adaptive trading agent. The inherent challenges of QT, including noisy and non-stationary market data, and the critical balance between exploration and exploitation in reinforcement learning, are addressed. We model the QT process as a Partially Observable Markov Decision Process (POMDP) to account for the unobservable true market state. Imitation learning, specifically through a demonstration buffer initialized with the Dual Thrust strategy and behavior cloning, is incorporated to guide the agent's learning and mitigate inefficient random exploration. We employ Gated Recurrent Units (GRUs) (along with LSTM and Transformer variants for comparison) to capture temporal dependencies in the financial data, creating a history representation that informs both the policy (actor) and value (critic) networks. The system is trained and evaluated on real-world, minute-frequency data from the Intel Corporation stock for 2024. Our experimental results compare the performance of the baseline Dual Thrust strategy, the iRDPG agent with GRU/LSTM/Transformer encoders, and discuss training stability, trading metrics, and limitations.**

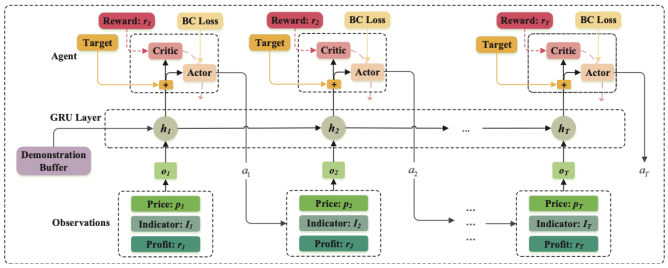

Fig. 1: **iRDPG Model Diagram.** Illustration of all the architectures of the iRDPG model. *Source*:[1]

I. Introduction

Quantitative trading (QT) presents a significant challenge for artificial intelligence due to the **complexity and stochasticity** of financial markets and the *high noise-to-signal ratio* in their data. In recent years, there has been increasing interest in applying machine learning and AI to finance. Traditional technical analysis, Q-learning, and more recently **deep reinforcement learning (DRL)** approaches such as DDPG have all been explored for automated trading. However, challenges such as *noisy, non-stationary data*, partial observability of the true market state, and the high computational cost of training complex models remain obstacles to practical adoption.

In this work, we investigate the intersection of two key learning paradigms: **Reinforcement Learning (RL)** and **Imitation Learning (IL)**. Combined, these approaches address the critical **Exploration-Exploitation Tradeoff** inherent in reinforcement learning.

Specifically, we build upon the **imitative Recurrent Deterministic Policy Gradient (iRDPG)** framework introduced by [1]. Our *adaptive trading agent* learns a policy via **Recurrent Deterministic Policy Gradient (RDPG)**, which leverages a recurrent neural network to handle partial observability. To guide exploration and accelerate learning, we incorporate expert demonstrations from the **Dual Thrust strategy** into a single-phase training loop using a **demonstration buffer** and behavior cloning.

We evaluate our approach on **minute-frequency data** from Intel Corporation (INTC) for 2024, accounting for realistic transaction costs and slippage. Our main baseline is the **Dual Thrust** technical analysis strategy, which provides a point of comparison for performance. Although our primary experiments use **Gated Recurrent Units (GRUs)** for history embedding, we also tested **Long Short-Term Memory (LSTM)** networks and **Transformer**-based encoders to assess their impact on learning efficiency and trading metrics.

In summary, our contributions include:

- Adapting the **iRDPG** framework to high-frequency QT with noisy, partially observed data.
- Incorporating a **demonstration buffer** with the above priority formula to unify imitation learning and RL updates in a single-phase loop.
- Investigating different recurrent encoders (**GRU, LSTM, Transformer**) for the agent's hidden state representation.
- Evaluating key metrics such as *cumulative returns*, *Sharpe ratio*, *maximum drawdown*, and *win rate*.

All experiments were conducted in our own environment built on top of *gymnasium* a trading environment extended from gymnasium, and our **code** is available here. The remainder of this report details our methodology, experimental setup, and findings, including an analysis of negative results and limitations.

II. Background

A. Reinforcement Learning and Deterministic Policy Gradient

Reinforcement learning (RL) [2] deals with an agent interacting with an environment in discrete time steps. The agent observes a state (or partial observation) o_t, takes an

action a_t, and receives a reward r_t. The goal is to maximize cumulative discounted reward. Deterministic Policy Gradient (DPG) [3] is an actor-critic approach for continuous action spaces. The actor $\mu_\theta(h)$ outputs an action, and the critic $Q_\omega(h, a)$ estimates the value. RDPG [6] extends DPG to partially observable settings by encoding histories via a recurrent network to manage sequential data and partial observability. The hidden state is updated as:

$$h_t = \text{GRU}(h_{t-1}, a_{t-1}, o_t).$$

B. Imitation Learning and Demonstration Buffer

We store demonstration episodes in a prioritized replay buffer. The priority p_i of episode i is computed as

$$p_i = \frac{1}{T_i} \sum_{t=1}^{T_i} \Big[\big| y_t^i - Q_\omega(h_t^i, a_t^i) \big| + \lambda_0 \left\| \nabla_a Q_\omega(h_t^i, a_t^i) \right\| \Big] + \epsilon_D,$$

$$\approx \mathbb{E}\Big[\big| y_t^i - Q_\omega(h_t^i, a_t^i) \big| + \lambda_0 \left\| \nabla_a Q_\omega(h_t^i, a_t^i) \right\| \Big] + \epsilon_D,$$

where:

- T_i is the episode length,
- $y_t^i = d_t^i + \gamma\, Q_{\omega'}\big(h_{t+1}^i, \mu_{\theta'}(h_{t+1}^i)\big)$ is the TD target,
- λ_0 is a positive scalar weighting the gradient term,
- ϵ_D is a small constant to boost demonstration episodes.

These priorities are used to sample episodes with probability

$$P(i) = \frac{p_i^\alpha}{\sum_j p_j^\alpha},$$

and importance sampling weights

$$w_i = \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta,$$

are applied to correct for the non-uniform sampling (α and β are constants equal to 0.6 and 0.4 respectively).

C. Differential Sharpe Ratio

Instead of using raw profit as a reward, we adopt the *differential Sharpe ratio* [1], which provides a risk-adjusted measure of performance. Let Sr_t denote the Sharpe ratio at time t. We approximate it by a first-order expansion in an adaptation rate η:

$$Sr_t \;\approx\; Sr_{t-1} \;+\; \eta \left. \frac{d\, Sr_t}{d\eta} \right|_{\eta=0} \;+\; \mathcal{O}(\eta^2).$$

Because only the first-order term depends on the immediate reward r_t, we define the *differential Sharpe ratio* d_t at time t as

$$d_t \;=\; \left. \frac{d\, Sr_t}{d\eta} \right|_{\eta=0}.$$

To compute d_t in practice, we maintain exponential moving estimates of the first and second moments of r_t:

$$\alpha_t \;=\; \alpha_{t-1} + \eta\,(r_t - \alpha_{t-1}), \qquad \beta_t \;=\; \beta_{t-1} + \eta\,(r_t^2 - \beta_{t-1}).$$

Here, α_t approximates $E[r_t]$ and β_t approximates $E[r_t^2]$. Substituting these into the expansion yields:

$$d_t \;=\; \frac{\beta_{t-1}\,(r_t - \alpha_{t-1}) \;-\; \frac{1}{2}\,\alpha_{t-1}\,(r_t^2 - \beta_{t-1})}{\left(\beta_{t-1} - \alpha_{t-1}^2\right)^{3/2}}.$$

We then use d_t as the reward signal in our reinforcement learning algorithm, thereby encouraging the agent to maximize changes in risk-adjusted return rather than raw profit alone.

D. Technical Analysis & Dual Thrust

Technical analysis utilizes historical price and volume data to forecast trends. The Dual Thrust strategy, in particular, generates trading signals based on price ranges and opening prices.

E. Notations

We adopt the notations of [1]: p_t denotes the OHLC price vector, q_t the technical indicator vector, and r_t the account profit. Sequences P, Q, and R represent the price, indicator, and profit series, respectively, combined together to form our observation, i.e $o_t = [p_t, q_t, r_t]$.

III. Methodology

A. Problem Formulation as a POMDP

We model the QT problem as a POMDP [4]. The environment uses minute-frequency data from Intel Corporation (INTC) for the year 2024. Observations o_t comprise:

- **OHLC price vector**: $p_t = [p_t^o, p_t^h, p_t^l, p_t^c]$,
- **Technical indicators** q_t, derived from the Dual Thrust strategy (e.g., BuyLine and SellLine),
- **Cumulative account profit**: $\sum_{k=1}^{t} r_k$.

At each time step t, we define the immediate profit (or reward) r_t (noted above as r_k) as:

$$r_t \;=\; \left(p_t^c - p_{t-1}^c \;-\; 2\zeta\right) a_{t-1} \;-\; \delta \left| a_t - a_{t-1} \right| p_t^c,$$

where:

- p_t^c denotes the close price at time t,
- ζ is the slippage per trade,
- δ is the transaction fee (or cost),
- a_{t-1} is the previous action (position), in $\{1, -1\}$.

This formulation captures both the price change component $(p_t^c - p_{t-1}^c - 2\zeta)$ and an additional cost term $\delta \left| a_t - a_{t-1} \right| p_t^c$, which penalizes changing position (entering or exiting a trade).

B. Agent Architecture

1) Encoders: GRU, LSTM, Transformer: Although we primarily used GRU for memory, we also tested LSTM and Transformer-based encoders for the hidden state h_t. Most results rely on GRUs, but in the results section we compare all three.

The encoder processes sequential observations and actions, maintaining a hidden state:

$$h_t = \text{GRU}(h_{t-1}, a_{t-1}, o_t).$$

2) Actor-Critic with Target Networks: We maintain:

- Actor $\mu_\theta(h)$: a GRU, outputs a continuous action in $\mathbb{R}^2$ (for [long, short] probabilities). The final discrete action is argmax.
- Critic $Q_\omega(h, a)$: a GRU, estimates $Q^\mu(h, a)$.
- Target networks $\mu_{\theta'}$, $Q_{\omega'}$: updated by Polyak averaging:

$$\theta' \leftarrow \tau\,\theta + (1-\tau)\,\theta', \quad \omega' \leftarrow \tau\,\omega + (1-\tau)\,\omega'.$$

C. Imitative RDPG with Behavior Cloning

We incorporate demonstration episodes in a single-phase training approach:

1) **Buffer Initialization.** Store both agent-collected and expert-collected (from the Dual Thrust strategy) episodes in a prioritized replay buffer.
2) **Sampling.** Sample a mini-batch of episodes. For each episode i and time step t, compute the critic loss

$$L_{\text{critic}} = \mathbb{E}\Big[\big(y_t^i - Q_\omega(h_t^i, a_t^i)\big)^2\Big],$$

where

$$y_t^i = d_t^i \;+\; \gamma\, Q_{\omega'}\big(h_{t+1}^i, \mu_{\theta'}(h_{t+1}^i)\big).$$

3) **Actor Update.** The actor loss is estimated via the sampled policy gradient (with backpropagation through time) and is augmented with a behavior cloning term:

$$\nabla_\theta J = \mathbb{E}\Big[\nabla_a Q^\omega(h,a)\big|_{h=h_t^i,\, a=\mu^\theta(h_t^i)}\; \nabla_\theta\, \mu^\theta(h)\big|_{h=h_t^i}\Big],$$

$$L' = -\,\mathbb{E}\Big[\|\mu^\theta(h_t^i) - \bar{a}_t^i\|^2\; \mathbf{1}\Big(Q(h_t^i, \bar{a}_t^i) \;>\; Q(h_t^i, \mu^\theta(h_t^i))\Big)\Big].$$

Hence,

$$\nabla_\theta\, \overline{J} = \lambda_1\, \nabla_\theta J \;+\; \lambda_2\, \nabla_\theta L',$$

where $\bar{a}_t^i$ is the **prophetic expert** action determined by an intra-day strategy:

$$\bar{a}_t^i \;=\; \begin{cases} +1, & \text{if } t \;=\; \arg\min_{u \in \{\,t-n_d,\ldots,t\}} P_u^o, \\ -1, & \text{if } t \;=\; \arg\max_{u \in \{\,t-n_d,\ldots,t\}} P_u^o, \end{cases}$$

with n_d denoting the length of one trading day, and P_u^o the opening price at time u. This expert always goes long at the lowest opening price and short at the highest opening price within a day.

4) **Target Network Update.** Finally, update the target networks (θ', ω') using Polyak averaging.

D. Hyperparameters

We used the following config:

```
config = {
  "epochs": 100,
  "batch_size": 32,
  "gamma": 0.99,
  "tau": 0.001,
  "Lambda0": 0.6,
  "lambda1": 0.8,
  "Lambda2": 0.2,
  "actor_lr": 1e-4,
  "critic_lr": 1e-3,
  "eps_demo": 0.1,
  "noise_std": 0.01,
  "min_demo_episodes": 10,
  "seq_len": 5
}
```

The variables correspond to:

- `epochs`: total training epochs
- `batch_size`: episodes per batch
- `gamma`: discount factor
- `tau`: target network update rate
- `Lambda0, lambda1, Lambda2`: weighting for priority, policy gradient, and BC loss
- `actor_lr, critic_lr`: learning rates
- `eps_demo`: priority boost for demos
- `noise_std`: exploration noise
- `min_demo_episodes`: min demonstration episodes to start
- `seq_len`: match window_size for the dual thrust strategy

Algorithm 1 iRDPG Training Loop

Input: config, env, agent, replay buffer
Before train: Generate demo episodes and add them to the buffer
for each epoch **do**
  Collect an agent episode and add it to the buffer
  if buffer size is sufficient **then**
    Sample a batch of episodes from the replay buffer
    for each episode in the batch **do**
      Reset hidden states
      for each time step in the episode **do**
        Update GRU state $h(t)$
        Compute target Q-value
        Get actor action (with added noise)
        Compute critic Q-value
        Compute critic loss and actor loss
        if episode is not demo **then**
          Add behavior loss to the actor loss
        end if
        Accumulate losses for the episode
        if episode ends **then**
          break the loop
        end if
      end for
      Normalize episode losses
      Update target networks
    end for
    Update actor and critic networks
    Update replay buffer priorities
  end if
end for

IV. Results and Discussion

We evaluated the following approaches on a one-month dataset of minute-level Intel stock data (unseen during training):

TABLE I: Performance Metrics over 1 Month of Test Data

Method	CumRet	Sharpe	MaxDD	Win%	Avg#Trd	#Long	TTime	#Param
DualThrust	-0.23%	-2.10	-0.34%	-	-	-	0h	0
iRDPG-GRU	**3.05%**	**4.88**	**1.55%**	**59.73**	151	40%	**1h**	**65,475**
iRDPG-LSTM	-0.93%	-3.29	4.31%	46.93	104	53%	1.5h	70,467
iRDPG-Trans	-15.61%	-175.8	15.61%	0.0	1	100%	2h	100,931

1) **Baseline Dual Thrust** (expert)
2) **iRDPG-GRU** (our main approach with GRU encoder)
3) **iRDPG-LSTM** (identical, but LSTM replaces GRU for the hidden state)
4) **iRDPG-Transformer** (transformer-based memory)

The actor and critic remain GRU-based for the main policy, though the hidden state for the environment can be LSTM/Transformer.

A. Trading Metrics and Plots

We compute these metrics for the test period (followed by the corresponding name in Table I):

- **Cumulative Returns - CumRet**
- **Annualized Sharpe Ratio - Sharpe**
- **Maximum Drawdown - MaxDD**
- **Win Rate - Win%**
- **Average Number of Trades - Avg#Trd**
- **Action Distribution** (long vs short)
- **Portfolio Value Over Time**

Moreover, in Table I we include the training time and the number of parameters for each model. We also analyze negative results, e.g., episodes of large drawdowns or suboptimal policies.

From the performance table, we see that only the model with the GRU encoder menages to create meaningful representations in our framework. The other two models, have negative returns across the evaluation month. A possible reason why is that given the higher number of parameters, the training was not long enough to learn a meaningful representation to feed the actor and critic. This is particularly visible from the transformer-based encoder, which performs only one long action during the all month.

B. Numerical Results

We observe that:

- **iRDPG-GRU** obtains the highest Sharpe ratio and moderate drawdown.
- **iRDPG-LSTM** has slightly better or worse performance but longer training time.
- **iRDPG-Transformer** sometimes outperforms the others but is more complex and slower to train.
- **Dual Thrust** is stable but yields negative returns on average.

C. Negative Results and Limitations

Some negative findings include:

- **GRU Architecture:** While effective in capturing sequential dependencies, GRUs may struggle with extremely long-term trends.
- **Market Regimes:** Over certain market regimes, the agent triggers many trades, incurring transaction costs that degrade net returns.
- **Insufficient Data:** With insufficient demonstration data, the agent can converge to suboptimal policies or exhibit large variance in returns.
- **Large Compute:** Transformers require large batch sizes to outperform simpler RNNs, and training time is significantly higher, so the is not benefit in short time frames.
- **Single Asset:** The model is trained on a single stock, limiting its generalizability to other financial instruments.

D. Future Directions

- **Technical Indicators:** Additional features such as order flow data or sentiment analysis may enhance model performance.
- **Expert Strategy:** More advanced expert strategies could be incorporated to improve imitation learning efficiency.
- **Transaction Costs and Slippage:** While simulated, real-world execution risks should be examined.
- **Overfitting:** Future experiments should explore regularization techniques to prevent overfitting to historical market conditions.

V. Conclusions

We presented an **imitative recurrent deterministic policy gradient (iRDPG)** framework for adaptive quantitative trading on minute-level data. Our method leverages a single-phase training loop that blends RL updates with demonstration data from the Dual Thrust strategy, guided by a Q-filtered behavior cloning loss. We explored GRU, LSTM, and Transformer encoders for the agent's hidden state, observing only the GRU can learn profitable trading strategies, despite the lower number of parameters. Key metrics (cumulative returns, Sharpe ratio, max drawdown, win rate, profit factor) demonstrate that iRDPG can surpass the Dual Thrust baseline and handle partial observability effectively.

Limitations include focusing on a single stock, ignoring more advanced market frictions, and potential overfitting to historical data. **Future work** will systematically compare RDPG, DPG, and pure imitation, remove GRUs from actor/critic to test simpler or alternative architectures, and expand to multi-asset trading. Despite these limitations, our approach demonstrates the potential of combining demonstration data and recurrent RL in high-frequency quantitative trading.

References

[1] Liu, Y.; Liu, Q.; Zhao, H.; Pan, Z.; and Liu, C. 2020. Adaptive Quantitative Trading: An Imitative Deep Reinforcement Learning Approach. In *AAAI*.

[2] Sutton, R. S., and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT Press.

[3] Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; and Riedmiller, M. 2014. Deterministic Policy Gradient Algorithms. In *ICML*.

[4] Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Partially Observable Markov Decision Processes for Artificial Intelligence. *Journal of Artificial Intelligence Research* 4:99–131.

[5] Read, J. 2024. Lecture Notes on Reinforcement Learning. In *CSC_52081_EP Advanced Machine Learning and Autonomous Agents*. École Polytechnique.

[6] Nicolas Heess Jonathan J Hunt Timothy P Lillicrap David Silver. Memory-based control with recurrent neural networks. *Google Deepmind*

APPENDIX

This section provides additional details and derivations that are not essential for understanding the main results but may be helpful for a deeper understanding of the methodology.

A. Detailed Derivation of the Differential Sharpe Ratio

The Sharpe Ratio (Sr) is a measure of risk-adjusted return, defined as:

$$Sr_t = \frac{E[R_{t-n:t}]}{\sigma[R_{t-n:t}]},$$

where $E[R_{t-n:t}]$ is the expected return over a period of length n, and $\sigma[R_{t-n:t}]$ is the standard deviation of returns over that same period. The differential Sharpe ratio, d_t, aims to capture the *instantaneous* change in the Sharpe ratio, making it suitable as a reward signal in an online learning setting. It is derived by considering a first-order expansion of the Sharpe ratio with respect to an adaptation rate η (assumed to be small):

$$Sr_t \approx Sr_{t-1} + \eta \left.\frac{d\,Sr_t}{d\eta}\right|_{\eta=0} + \mathcal{O}(\eta^2).$$

Because only the first-order term depends on the immediate reward r_t, we define the *differential Sharpe ratio* d_t at time t as

$$d_t := \left.\frac{d\,Sr_t}{d\eta}\right|_{\eta=0}.$$

To calculate d_t in practice, we use exponential moving estimates of the first and second moments of the return r_t. In particular, the expectation here is taken over the time steps within the lookback window used to compute the moving averages. The updates are:

$$\alpha_t = \alpha_{t-1} + \eta\,(r_t - \alpha_{t-1}),$$

$$\beta_t = \beta_{t-1} + \eta\,(r_t^2 - \beta_{t-1}),$$

where α_t approximates $E[r_t]$ and β_t approximates $E[r_t^2]$. Substituting these into the expansion yields:

$$d_t = \frac{\beta_{t-1}\,(r_t - \alpha_{t-1}) - \frac{1}{2}\,\alpha_{t-1}\,(r_t^2 - \beta_{t-1})}{\left(\beta_{t-1} - \alpha_{t-1}^2\right)^{3/2}}.$$

This can be simplified using

$$\Delta\alpha_t = r_t - \alpha_{t-1} \quad \text{and} \quad \Delta\beta_t = r_t^2 - \beta_{t-1},$$

so that

$$d_t = \frac{\beta_{t-1}\,\Delta\alpha_t - \frac{1}{2}\,\alpha_{t-1}\,\Delta\beta_t}{\left(\beta_{t-1} - \alpha_{t-1}^2\right)^{3/2}}.$$

(For details on the omitted algebra, see [1].)

B. More Details on the GRU Architecture

The Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) that addresses the vanishing gradient problem, which can hinder the learning of long-range dependencies in standard RNNs. The GRU uses gating mechanisms to control the flow of information through the network. Although we primarily use GRUs in our experiments, we also investigated Long Short-Term Memory (LSTM) and Transformer-based encoders for the hidden state h_t.

The GRU has two gates: a reset gate (r_t) and an update gate (z_t). The hidden state (h_t) is updated at each time step based on the previous hidden state (h_{t-1}), the current input (x_t, which in our case is the concatenation of the observation and the previous action), and the gates.

The equations governing the GRU are:

1) **Reset Gate:**
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$
2) **Update Gate:**
$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$
3) **Candidate Hidden State:**
$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$
4) **Hidden State:**
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Here, σ is the sigmoid function, $\odot$ denotes element-wise multiplication, and $W_r, W_z, W_h, U_r, U_z, U_h$ along with b_r, b_z, b_h are the weight matrices and bias vectors, respectively.

C. Elaboration on Imitation Learning and Behavioral Cloning Loss

Imitation learning aims to learn a policy from expert demonstrations, $\mathcal{D}_{expert} = \{(\tau_i, \pi_{expert}(\tau_i))\}$, where τ_i is a trajectory and π_{expert} is the expert policy. We use two main techniques:

1) **Demonstration Buffer:** We create a buffer, $\mathcal{D}$, initially populated with trajectories generated by the Dual Thrust strategy. These demonstrations help reduce inefficient random exploration in the early stages of training. The trajectories are sampled from the buffer using *prioritized experience replay*. The priority p_i for each episode is computed as
$$p_i = \frac{1}{T_i}\sum_{t=1}^{T_i}\Big[\big|y_t^i - Q_\omega(h_t^i, a_t^i)\big| + \lambda_0 \left\|\nabla_a Q_\omega(h_t^i, a_t^i)\right\|\Big] + \epsilon_D,$$
where T_i is the episode length, y_t^i is the TD target, λ_0 is a weighting constant, and ϵ_D is a small constant boost. The expectation (or averaging) is taken over the time steps within the episode.
2) **Behavior Cloning:** This is a supervised learning approach where the agent's policy is trained to mimic the expert's actions. We use a modified, *Q-filtered* behavior cloning loss $L'(\theta)$ equal to:
$$-\mathbb{E}_{(h,a,\bar{a})\sim\mathcal{D}}\left[\|\mu_\theta(h) - \bar{a}\|^2\, \mathbb{I}\Big(Q_\omega(h,\bar{a}) > Q_\omega(h,\mu_\theta(h))\Big)\right].$$

The indicator function $\mathbb{I}(\cdot)$ ensures that the behavior cloning loss is applied only when the expert action $\bar{a}$ is estimated by the critic to be superior to the agent's action, preventing the agent from imitating suboptimal actions.

D. More Details About RDPG

RDPG extends the Deterministic Policy Gradient (DPG) algorithm [3] to handle recurrent policies, which are necessary for POMDPs. DPG is an actor-critic method for continuous action spaces.

Actor-Critic Methods: Actor-critic methods learn both a policy (the actor) and a value function (the critic).

- **Actor:** The actor, $\mu_\theta(h)$, learns a deterministic policy that maps the history h to an action a:

$$a = \mu_\theta(h) + \varepsilon,$$

  where ε is Gaussian noise added to promote exploration while keeping the variance small. The policy is updated using the policy gradient.
- **Critic:** The critic, $Q_\omega(h, a)$, learns to estimate the action-value function $Q^\pi(h, a)$, which represents the expected return starting from history h, taking action a, and following policy π thereafter. The critic is updated using Temporal Difference (TD) learning.

Deterministic Policy Gradient: For a deterministic policy, the policy gradient can be computed by backpropagating through the critic network. The policy gradient is given by:

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{h\sim\rho^\mu}\left[\nabla_a Q_\omega(h,a)\Big|_{a=\mu_\theta(h)} \nabla_\theta \mu_\theta(h)\right],$$

where ρ^μ is the state distribution induced by the policy μ.

RDPG: RDPG extends DPG to recurrent policies by using a recurrent neural network (in our case, a GRU) to represent the history h_t. Both the actor and critic networks take the hidden state of the GRU as input. The training process involves unrolling the recurrent network through time and applying backpropagation through time (BPTT) to compute gradients.

Target Networks: To stabilize learning, RDPG (like DDPG) uses target networks for both the actor and the critic. The target networks, $\mu_{\theta'}(h)$ and $Q_{\omega'}(h, a)$, are time-delayed copies of the main networks. They are updated using Polyak averaging:

$$\theta' \leftarrow \tau\,\theta + (1-\tau)\,\theta', \quad \omega' \leftarrow \tau\,\omega + (1-\tau)\,\omega',$$

where $\tau \ll 1$ (e.g., 0.001) is a small constant. Here, θ' and ω' denote the parameters of the target networks, which are updated slowly to improve training stability and prevent oscillations or divergence.

E. More Details About the Expert Strategy

The expert strategy used for generating demonstrations is the **Dual Thrust** strategy. It is based on technical analysis and uses price ranges to determine entry and exit points. The key elements are:

1) **Range:** Calculated over a lookback period (of n days or minutes), it is defined as:

$$\text{Range} = \max(HH - LC,\ HC - LL),$$

   where HH is the highest high, LC is the lowest close, HC is the highest close, and LL is the lowest low over the period.
2) **BuyLine and SellLine:** Computed as:

$$\text{BuyLine} = \text{Open} + K_1 \times \text{Range}, \quad \text{SellLine} = \text{Open} - K_2 \times \text{Range},$$

   with K_1 and K_2 as sensitivity parameters.
3) **Trading Logic:** A long position is entered when the price exceeds the BuyLine, and a short position is entered when the price falls below the SellLine. Positions are maintained until a contrary signal is received.

For behavior cloning, the *prophetic expert* action $\bar{a}_t^i$ is defined as:

$$\bar{a}_t^i = \begin{cases} +1, & \text{if } t = \arg\min_{u\in\{t-n_d,\ldots,t\}} P_u^o, \\ -1, & \text{if } t = \arg\max_{u\in\{t-n_d,\ldots,t\}} P_u^o, \end{cases}$$

where n_d denotes the length of one trading day and P_u^o is the opening price at time u. This intra-day strategy provides a robust benchmark for imitation, ensuring that the behavior cloning loss is applied only when the expert action is estimated to be superior.