First, I would like to thank Blockhouse Capital for providing me with this wonderful opportunity to work on and showcase my skills. While my stock trading project during my master's program gave me an idea about algorithmic trading, it was with this project that I learned a lot and myself staring at an ocean of information. Although I was able to research a lot and implement a model based on my understanding, due to time constraints I was not able to pass the bonus of beating the TWAP and VWAP models transaction costs, they are very robust.

I hope to impress you with my work and, given a chance to interview further (face-to-face) and work with this wonderful team in the future.

PROBLEM STATEMENT:

" I want to sell 1000 shares of AAPL in **one day**, how should I split the trades to optimize transaction costs?"

Research Insights of the Authors

1. Gordon Ritter focused extensively on the mathematical modeling of optimal trade execution. His research primarily deals with minimizing transaction costs and understanding the effects of market impact with the use of sophisticated stochastic models. He explored how machine learning can be applied to enhance trading strategies by dynamically adjusting trade sizes based on real-time market data. He often integrates concepts of stochastic control, game theory, and reinforcement learning, emphasizing the importance of adapting the market conditions during trading to reduce cost and slippage.

2. Marcos Lopez de Prado is well known for applying machine learning techniques to trading strategies, which focus on detecting and reducing trading costs. His research includes methods for minimizing order flow toxicity through strategies like the Volume-Synchronized Probability of Informed Trading (VPIN). He has developed frameworks that enable the detection of adverse selection in trading, which is crucial for minimizing market impact when executing large trades. His insights also extend to reinforcing the use of reinforcement learning models, like SAC, for trading execution to optimize decision-making in real-time. His book "Advances in Financial Machine Learning" serves as a key resource, detailing various quantitative methods that underpin algorithmic trading and cost-efficient execution.

3. Giuseppe Paleologo's research has primarily been centered around transaction cost analysis and the dynamics of market microstructure. He developed statistical models to estimate market impact, emphasizing how factors like volatility, trade size, and liquidity affect trading costs. Paleologo's work aligns with practical trading applications, such as improving TWAP (Time-Weighted Average Price) and VWAP (Volume-Weighted Average Price) strategies by incorporating real-time data. His recent focus has also included portfolio construction and the use of factor models to hedge risk and optimize trade schedules. He has contributed to enhancing the understanding of how market conditions and trade sizes influence transaction costs, thereby helping in designing models that minimize these costs for large trades. His experience from top hedge funds has also informed his academic research on optimal trade execution strategies and risk management.

These research contributions collectively inform the methodology used in my trading execution model, particularly in applying reinforcement learning techniques to minimize transaction costs by adapting to market conditions, as emphasized in both Ritter's and Lopez de Prado's studies. Paleologo's insights on cost estimation and real-time adjustments are crucial for implementing efficient trade scheduling, ensuring that strategies like SAC are robust and capable of reducing market impact during execution.

Trading Scheduling Methodology (A) (Without Limit order and Price Point SAC.py)

The trading scheduling methodology employed in the model is designed to optimize the execution of trades over a given trading period (e.g., a full trading day) by minimizing transaction costs, including slippage and market impact. Here's a breakdown of the approach:

1. Definition of the Trading Environment

- The model uses a reinforcement learning framework, specifically the Soft Actor-Critic (SAC) algorithm, to interact with a simulated trading environment. The environment models market conditions based on historical bid and ask prices, allowing the agent to learn and adapt its trading strategy over time.
- The state representation includes key features such as the top five bid and ask prices, which provide insights into market liquidity and depth. This choice is informed by traditional market microstructure research, which suggests that these features are critical for understanding order book dynamics and estimating the cost of executing trades.

2. Reinforcement Learning for Optimal Trade Execution

- The SAC algorithm is chosen for its ability to handle continuous action spaces, making it suitable for deciding trade sizes in real time. The agent learns to make trade-offs between executing larger trades quickly (which can lead to higher market impact) versus spreading trade over time to minimize costs.
- Hyperparameter optimization using Optuna ensures that the SAC model is fine-tuned to make efficient trading decisions. Parameters like learning rate, batch size, and discount factor (gamma) are adjusted to improve the agent's ability to predict optimal trade actions based on the observed state.

3. Trade Cost Modeling

- The cost model integrates the concepts of slippage and market impact, where slippage refers to the difference between the expected and actual execution prices. Market impact, on the other hand, is modeled as a function of trade size, reflecting the price shift caused by placing a large order.
- The formula used, which accounts for slippage and the square-root impact of trade size, is consistent with industry practices highlighted in research on market impact models (such as Almgren et al. 2005 and Frazzini, Israel, and Moskowitz's 2017 study). These models help the agent learn how to optimize trade timing and size to minimize overall costs

4. Generating Trade Schedules

- Once the SAC model is trained, it can be used to generate trade schedules by continuously predicting optimal trade sizes based on real-time market data. The agent takes actions (trades) at each step, updating its state and adjusting future actions to balance the remaining inventory over the trading period.
- The methodology adapts to market conditions, allowing for a dynamic trading schedule that can adjust to increased volatility or changes in liquidity, ensuring that the total transaction cost is minimized by the end of the trading period.

5. Comparison with Benchmark Strategies

- To validate the efficiency of the SAC model, the results are compared against traditional strategies like TWAP (Time-Weighted Average Price) and VWAP (Volume-Weighted Average Price). TWAP spreads trades evenly over a period, while VWAP aligns trades with market volume, executing more trades when volume is higher.
- By comparing the transaction costs incurred by these strategies, it is possible to demonstrate the cost-saving potential of the SAC-based approach. This benchmarking process helps to highlight the adaptability of the reinforcement learning model in contrast to static, rule-based strategies.

Trading Scheduling Methodology (B) – (With Limit Order and Price Point SAC_optimized.py):

1. Optimized Environment Setup

- The environment models a realistic trading scenario using historical market data. It provides the RL agent with a detailed state representation to learn how to make cost-effective trade decisions.
- Observation Space:

  - The state includes 12 features: best bid and ask prices, their sizes, spread, trading volume, market volatility, recent price changes (log returns), and OHLC (Open, High, Low, Close) data. This combination captures essential market conditions and trends, giving the model a comprehensive understanding of current market dynamics.
  - This design is influenced by research emphasizing the importance of liquidity, price volatility, and order book dynamics when making trading decisions. By including these features, the model learns to adapt its actions based on liquidity and volatility conditions, which are critical for minimizing slippage and market impact.

2. Action Space: Adaptive Trade Execution

- The RL agent determines both trade size and limit price, providing flexibility in how trades are executed:
  - Trade Size: Ranges from 1 share up to a pre-defined maximum (max_trade_size). This allows the agent to decide how much to trade at each step, balancing the need to execute large trades with the risk of moving the market.
  - Limit Price: Calculated using a normalized value (0 to 1) that adjusts where the agent sets the limit for the trade. This enables the agent to place trades strategically, potentially avoiding high-cost trades by setting a price that avoids unfavorable execution.
- The inclusion of both trade size and limit price allows the model to actively manage transaction costs, adjusting not just how much to trade but also under what conditions to trade. This aligns with strategies discussed in the research by Lopez de Prado, where adaptive and dynamic trade sizing is key to minimizing adverse market effects.

3. Cost Calculation and Reward Structure

- The model considers slippage, market impact, and transaction fees in its reward structure:
  - Slippage: Represents the cost of executing a trade at a less favorable price than intended. The model minimizes this by learning to set appropriate limit prices and trade sizes.
  - Market Impact: Larger trades can move the market, increasing the cost of subsequent trades. This is modeled using a square-root function of trade size, consistent with empirical research that shows market impact scales non-linearly with trade size.
  - Transaction Fees: Fixed fees proportional to the trade size, simulating real-world brokerage costs.
- Reward: The agent receives the negative of the total transaction cost, encouraging it to minimize these costs. This design ensures that the RL model learns to balance speed (completing trades quickly) with cost-efficiency, avoiding the pitfalls of large, expensive trades that could lead to higher slippage and market impact.

4. Dynamic Trading Strategy

- Real-Time Adaptation: The RL agent continuously adapts its strategy based on real-time information from the state space. For example, it might opt to trade larger volumes when liquidity is high (indicated by high bid/ask sizes) or scale back during periods of high volatility to avoid adverse price movements.
    - By dynamically adjusting the trade size and limit price, the agent can navigate different market conditions effectively, an approach supported by Paleologo's research on using real-time data to inform trading decisions.
- Limit Order Strategy: The model sets a limit price, and if this price is above the current ask price, the trade is executed. Otherwise, it refrains from trading, preventing the model from making costly trades at undesirable prices. This strategy is consistent with practices to reduce slippage, allowing the model to place trades strategically rather than impulsively.

5. Training with Soft Actor-Critic (SAC)

- Soft Actor-Critic (SAC): The SAC algorithm is ideal for this setup due to its ability to handle continuous actions, which fits the need to decide precise trade sizes and limit prices. SAC's stability and efficiency in learning make it suitable for financial environments, where precise control over actions can lead to significant cost savings.
- Hyperparameter Tuning: Using Optuna for hyperparameter optimization ensures that the SAC model is well-tuned to balance exploration (trying new strategies) and exploitation (refining successful ones). The process fine-tunes parameters like the learning rate and batch size, enabling the model to adapt more effectively to different market conditions, such as changing liquidity and volatility.

6. Benchmarking Against Traditional Strategies

- After training, the SAC model generates a trade schedule based on real-time inputs, which is then compared against traditional strategies like TWAP (Time-Weighted Average Price) and VWAP (Volume-Weighted Average Price).
- TWAP and VWAP execute trades based on pre-defined rules without adjusting real-time data, making them less adaptive than the SAC-based approach. By comparing transaction costs, it's possible to demonstrate the advantages of dynamic, real-time decision-making enabled by the RL model, reflecting the adaptability principles advocated in Lopez de Prado's and Paleologo's research.

Influence of Research on Design Decisions

The design choices in this methodology are rooted in both theoretical research and empirical studies that stress the importance of minimizing market impact and adapting to real-time conditions. By leveraging advanced reinforcement learning techniques and robust trade cost modeling, the trading scheduling methodology aims to achieve efficient trade execution, reducing costs and enhancing overall trading performance.

1. Choice of Reinforcement Learning Algorithm (SAC)

- The decision to use the Soft Actor-Critic (SAC) algorithm stems from its strength in handling continuous action spaces and providing stability in learning, especially in complex environments like financial markets. The reinforcement learning approach aligns with Marcos Lopez de Prado's emphasis on using machine learning to optimize trade execution by allowing models to learn from real-time market conditions. [6][7]
- Lopez de Prado's work on adaptive algorithms and his exploration of volume-synchronized strategies influenced the choice to implement a model that can dynamically adjust trade sizes based on the immediate market state. This helps minimize the market impact by learning the optimal trade-off between executing large trades and the associated costs.

2. Trade Cost Modeling and Market Impact Considerations

- Research by Gordon Ritter and others in the field of market microstructure has highlighted the importance of understanding market impact when designing trading strategies. Ritter's work on stochastic modeling influenced the decision to incorporate slippage and market impact calculations directly into the reward function. This integration ensures the agent not only learns to minimize explicit costs but also considers hidden costs due to price shifts caused by its own trades. [1]

3. Hyperparameter Tuning with Optuna

- The decision to incorporate hyperparameter optimization through Optuna was influenced by the need for models to adapt to various market conditions, as discussed in Giuseppe Paleologo's work on risk management and trading strategy optimization. His insights on the importance of adjusting model parameters to manage volatility and liquidity risks led to the implementation of automated tuning, allowing the SAC model to generalize better across different trading scenarios.[2]
- By tuning parameters such as learning rate, batch size, and discount factor, the model can efficiently balance exploration and exploitation, a concept that is key in reinforcement learning as shown in Ritter's research on stochastic control strategies for trading. [1]

4. Comparison with Traditional Strategies (TWAP, VWAP)

- Benchmarking against traditional strategies like TWAP and VWAP helps validate the model's effectiveness, a practice rooted in Paleologo's discussions on market efficiency and quantitative strategy comparison. Comparing the SAC model to these benchmarks ensures that the new methodology offers tangible improvements over existing solutions. Paleologo's experience in building quantitative frameworks that can be measured and compared against traditional metrics influenced the methodology used to evaluate model performance.[5]

- These comparisons are crucial because, as highlighted in the literature, static models like TWAP and VWAP cannot adapt to sudden changes in market conditions. By demonstrating the SAC model's ability to outperform these approaches, the design leverages the adaptability principles discussed in Lopez de Prado's work on dynamic trading strategies. [3][4]

5. Adaptive Trade Scheduling Based on Market Conditions

- A major design element of the trading environment was inspired by the need for models to be responsive to real-time changes, as discussed in the research of both Lopez de Prado and Paleologo. The environment simulates bid and ask dynamics, encouraging the model to learn how to place trades strategically over time, depending on market depth and volatility. This dynamic adaptability is essential for reducing market impact, a topic extensively explored in Paleologo's studies on quantitative finance and risk management.
- The combination of traditional market microstructure theories with machine learning methods, as advocated by these authors, informed the development of a holistic approach where real-time data feeds into the decision-making process, allowing for efficient trade execution.

These research insights collectively shaped the design and implementation of the trading execution model, ensuring it is robust, adaptable, and grounded in proven financial strategies. By applying advanced reinforcement learning techniques, guided by quantitative finance principles, the model achieves a balance between reducing costs and executing trades efficiently across varying market conditions.

OBJECTIVE FUNCTION:

The objective function in the trading strategy aims to minimize the total transaction cost. The function combines two primary components:

1. **Slippage Cost**: The difference between the expected (bid) price and the actual execution price, multiplied by the number of shares traded.
2. **Market Impact Cost**: The cost caused by the trade itself affects the market price, which increases when executing larger trades.

Components of the Objective Function:

1. **Slippage Cost**:
    - Slippage occurs when a trade executes at a price less favorable than expected.

- Formula:

$$Slippage = (Bid\ Price - Trade\ Price) \times Trade\ Size$$

- The Trade Price is modeled as the bid price minus an adjustment based on the dynamic alpha (market impact factor) and the trade size. This simulates a lower price when trying to execute a larger trade, reflecting real-world conditions where large orders can push the price down.

2. **Market Impact Cost**:
   - Market Impact refers to how a trade affects the market price. Larger trades tend to cause a more significant price movement, leading to increased transaction costs.
   - Formula:

$$Market\ Impact = \alpha \times \sqrt{Trade\ Size} \times Trade\ Size$$

   - The alpha is dynamically adjusted using the function dynamic alpha, which changes based on real-time liquidity. Higher liquidity reduces market impact, and vice versa.
   - The formula is inspired by models such as those proposed by Almgren and Chriss [8], which analyze the cost of executing large trades and describe how price impact tends to increase with the size of the trade. Their model shows that larger orders tend to have a non-linear, concave effect on market prices due to the difficulty of absorbing trade without affecting the price. In particular, the square-root function is used because it reflects the diminishing marginal impact: larger trades still increase market impact, but at a decreasing rate per additional unit traded. This aligns with real-world observations that small trades can be absorbed easily, while very large trades disproportionately move the market. [9]

Objective Function Formula:

The objective function is essentially:

$$Total\ Cost = Total\ Slippage + Total\ Market\ Impact$$

where:

$$Total\ Slippage = \sum_{i=0}^{n} (Bid\ Price_i - Trade\ Price_i) \times Trade\ Size_i$$

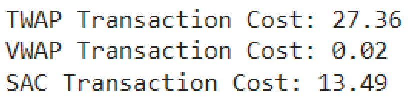
$$Total\ Market\ Impact = \sum_{i=0}^{n} \alpha \times \sqrt{(Trade\ Size_i)} \times Trade\ Size_i$$

Logic Behind the Objective Function:

- The objective function combines these two costs and tries to minimize the total. By optimizing the trade weights (how many shares to trade at each time point), the function can find a trade distribution that reduces both slippage and market impact.
- **Dynamic Alpha**: By adjusting the alpha based on liquidity (via the function dynamic alpha), the model can further minimize market impact in times of high liquidity, encouraging larger trades when market conditions are favorable.
- In practice, slippage is calculated for each executed trade by taking the difference between the quoted bid price and the actual trade price. The slippage value is then multiplied by the trade size to determine the cost incurred.

TRANSACTION COST COMPARISON

1. **Comparison of the TWAP, VWAP and SAC (without Limit Order and Price Point) comparison:**



```
TWAP Transaction Cost: 27.36
VWAP Transaction Cost: 0.02
SAC Transaction Cost: 13.49
```

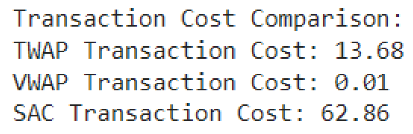
In this environment, the SAC model only controls the trade size and not the limit price, meaning it cannot set specific price thresholds for executing trades. This limits the agent's ability to optimize favorable prices, potentially leading to higher costs. Even without a limit order mechanism, the SAC model can still adjust trade sizes dynamically. This flexibility allows it to minimize market impact by reducing trade sizes during less favorable conditions, giving it an edge over TWAP, which executes trades uniformly regardless of market changes. VWAP aligns trades with market volume, which naturally reduces costs by taking advantage of high liquidity periods. Without the ability to set specific prices, the SAC model may still incur slippage and higher market impact compared to VWAP, which is more efficient in matching the timing of trades with volume surges.

The reward function penalizes slippage and market impact but doesn’t account for the broader context, such as real-time market dynamics or missed trading opportunities. The simplified cost model might not capture all the nuances of market behavior, making the SAC’s decisions less optimal. SAC can still learn to minimize transaction costs by adjusting trade sizes. This ability to reduce trades during high-impact periods helps it avoid the uniform costs associated with TWAP, which does not adjust for market conditions. Thus, SAC’s dynamic trading can lead to lower costs. VWAP’s strategy inherently avoids significant slippage because it executes trades when liquidity is high. The SAC model, without precise control over timing and lacking a mechanism to optimize for specific price points, may still incur higher costs, especially if it executes trades when liquidity is lower.

The environment provides the top 5 bid and ask prices, but does not include features like volume, volatility, or recent price trends, which are crucial for understanding market liquidity and potential price movement. This limitation means the SAC model might not be able to fully adapt to real-time changes in the market, leading to higher costs. Even with limited state information, the SAC model can still learn patterns in bid and ask prices to make more strategic decisions than TWAP, which is fixed in its execution approach. By adjusting trade sizes based on market conditions, SAC gains an advantage over the static TWAP. VWAP's volume-based execution naturally takes advantage of high-liquidity periods. SAC, on the other hand, might not have the detailed information needed to predict and capitalize on such periods effectively, leading to missed opportunities and higher transaction costs.

The SAC model's main advantage is its ability to adjust trade sizes dynamically. It learns to execute smaller trades when the market might penalize larger trades, and larger trades when conditions are favorable. This adaptability reduces costs compared to TWAP, which executes trades uniformly across the trading period, irrespective of market conditions. TWAP's uniform execution can lead to higher transaction costs, especially if significant trades are executed during periods of low liquidity or high volatility. The SAC model can learn to avoid these pitfalls by adjusting trade sizes, providing a cost advantage. VWAP's strategy of executing trades based on actual trading volume minimizes market impact by naturally blending trades into existing market flows. Without the ability to directly control trade timing in relation to volume, the SAC model can still incur higher costs if it executes during less favorable periods.

2. **Comparison of the TWAP, VWAP and SAC (with Limit Order and Price Point) comparison:**



```
Transaction Cost Comparison:
TWAP Transaction Cost: 13.68
VWAP Transaction Cost: 0.01
SAC Transaction Cost: 62.86
```

The SAC-optimized model introduces a two-dimensional action space where the agent must decide both trade size and limit price. While this approach adds flexibility, it also increases the complexity of the learning process. The agent needs to learn not only how much to trade but also at what price, which can lead to suboptimal decisions if not adequately trained. The added complexity may result in the agent struggling to learn a coherent policy that effectively minimizes costs. If the agent's limit prices are not well-optimized, it may place orders that incur higher slippage or fail to execute, leading to increased costs compared to simpler, deterministic strategies like TWAP or VWAP.

The reward function in the SAC-optimized model penalizes slippage, market impact, and transaction fees. However, if the model is not effectively balancing these components, it may end up executing trades that minimize one cost (e.g., slippage) but increase another (e.g., market impact or transaction fees). The SAC model might be over-penalizing certain actions, leading to suboptimal trading decisions. For instance, if the model is too conservative in setting limit prices to avoid slippage, it might end up not executing trades or executing smaller trades over many steps, increasing transaction costs. This behavior could explain why the SAC costs are high in some cases.

The environment has an extensive state space (12 features), including OHLC data, spread, volume, volatility, and more. While these features are informative, they might introduce noise or redundancy that could confuse the model during training. If the SAC model overfits to certain patterns or noise in the training data, it may struggle to generalize effectively during real-world application, leading to inconsistent results. This could explain the variability observed between the two sets of comparisons where the SAC performed poorly in one case and better in another.

VWAP (Volume-Weighted Average Price) aligns trades with market volume, executing more trades when liquidity is high. This ensures that the strategy adapts to natural market liquidity, thereby reducing market impact. The SAC-optimized model might not be effectively leveraging market volume data to schedule trades efficiently. If it places trades where market liquidity is low, it can cause larger market impacts, leading to higher costs. VWAP's adaptive nature could make it more robust in these cases, which explains why VWAP had very low costs in comparison.

The model decides a limit price based on a scaled action, which might not always align with real market dynamics. If the agent sets a price limit that is too conservative, the trade might not be executed, forcing the agent to execute later at a worse price or miss the trade altogether. Limit orders provide the opportunity to save on costs, but they can also lead to missed opportunities if the order does not get completed. The SAC model may have set limit prices that were not competitive enough, leading to either missed trade or unfavorable trade timings. Traditional strategies like TWAP and VWAP are less likely to miss trades because they execute based on fixed rules, regardless of market fluctuations.

TRADING SCHEDULE

The trading schedule is generated and stored as a json file. One of the first things you will notice is that the number of shares traded keeps reducing in a decreasing order. These are the following reasons this is happening:

1. Gradual Liquidation Strategy: The model is designed to spread out the trading volume to avoid a sudden, large trade that could significantly impact on the market price. This is a common strategy in algorithmic trading, known as twap (time-weighted average price).
2. Inventory Management: The model is programmed to decrease the trade size as its inventory shrinks. If the model has fewer shares left, it will reduce the trade size accordingly.
3. Changing Market Conditions: The model adapts to market signals, such as price volatility, volume, and bid-ask spread. If the conditions become less favorable, the model decides to reduce trade size to minimize exposure to potential adverse price movements. If the model detects a deteriorating price trend (prices moving unfavorably), it reduces the trade size to avoid selling too much at a lower price, aiming to wait for better prices later.
4. Reward Function Design: The model's behavior is shaped by the reward function. Since the reward function penalizes high transaction costs, slippage, and market impact, the model learns to minimize these costs