A-FORCE PROJECT

AI DRIVEN

WHITE PAPER

AI-DRIVEN NEXT-GENERATION CRYPTOCURRENCY MARKET MAKER PLATFORM

A-FORCE.SITE

Chapter 1: High-Frequency Algorithm Optimization

1. Dynamic Market Data Capture

Real-Time Data Acquisition

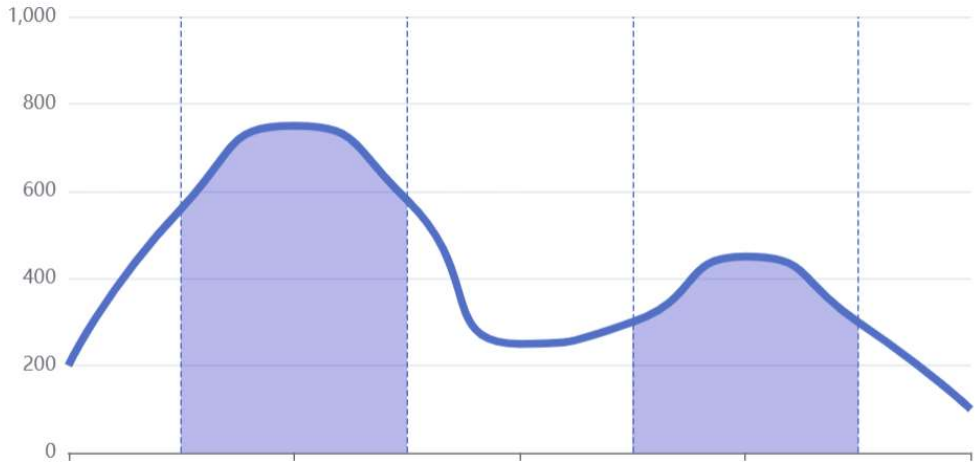
A-Force utilizes advanced low-latency data acquisition technology to obtain real-time market data from multiple cryptocurrency exchanges (such as **Binance, Coinbase**, etc.) with millisecond-level intervals. This includes, but is not limited to, exchange market depth (order books), historical transaction prices, trading volumes, and other related information. By continuously receiving and updating this data, the platform ensures that users have access to the latest market dynamics, making sure that trading decisions are always based on the most accurate and up-to-date information. To **enhance** the complexity and describe the real-time data collection process, A-Force introduces concepts such as stochastic processes, weighted averages, and latency optimization. The following formula integrates the latency, noise, and weighting factors from multiple data sources to ensure that the platform can efficiently capture market data in real-time from multiple exchanges.

$$T_{\text{latency}} = \frac{\sum_{i=1}^{N} \left(\Delta t_i + \gamma_i \cdot \exp\left(-\frac{(T_{\text{sync}}^{(i)} - T_{\text{ref}})^2}{\sigma_{\text{sync}}^2} \right) \right)}{N} + \sum_{j=1}^{M} W_j \cdot \left(\int_0^{T_{\max}} \frac{dP_j(t)}{dt} \cdot \left(1 - \frac{t}{T_{\max}} \right) dt \right)$$

Formula Explanation:

- TlatencyT_{\text{latency}}: The average data acquisition latency (in milliseconds). This represents the total delay that A-Force experiences when obtaining real-time data from multiple cryptocurrency exchanges.
- NN: The number of exchanges (e.g., multiple cryptocurrency exchanges like Binance, Coinbase).
- Δti\Delta t_i: The data acquisition latency for the ii-th exchange (in milliseconds). Each exchange may have a different delay based on factors like network conditions and trading volume.
- γi\gamma_i: The latency correction factor for the ii-th exchange, simulating the effects of latency fluctuations.
- Tsync(i)T_{\text{sync}}^{(i)}: The synchronization timestamp for the ii-th exchange, representing the time at which the data is captured.
- TrefT_{\text{ref}}: The reference timestamp, typically taken as the most recent timestamp to serve as the baseline time.
- σsync2\sigma_{\text{sync}}^2: The standard deviation used to adjust the synchronization timestamp, controlling the influence of timestamp synchronization errors.
- WjW_j: The weight of the jj-th data source. Different exchanges or data sources are assigned weights based on their influence, data quality, and other factors.
- ∫0TmaxdPj(t)dt⋅(1−tTmax)dt\int_0^{T_{\text{max}}} \frac{dP_j(t)}{dt} \cdot \left(1 -

\frac{t}{T_{\text{max}}} \right) dt: This integral simulates the rate of change in market prices. It calculates the impact of price changes on data acquisition latency, taking into account a time decay factor (1−tTmax)\left(1 - \frac{t}{T_{\text{max}}}\right) to simulate the effect of price volatility on the data.



Formula Analysis:

- **Latency Calculation:**
  For each exchange, the data acquisition latency Δti\Delta t_i is calculated, and the total latency TlatencyT_{\text{latency}} is obtained by performing a weighted average of the latencies across all exchanges.
- **Time Synchronization Optimization:**
  To improve synchronization accuracy, the formula introduces a correction term for timestamps. This takes into account the time errors that occur due to the synchronization of data from different exchanges. A Gaussian decay function is applied to further adjust the synchronization errors.
- **Market Price Change Impact:**
  When calculating the total latency, the formula also considers the impact of market price fluctuations on data updates. Using the integral formula, the rate of price change for each data source is weighted to reflect how market volatility at different exchanges influences data acquisition and latency.

Multi-Source Data Fusion

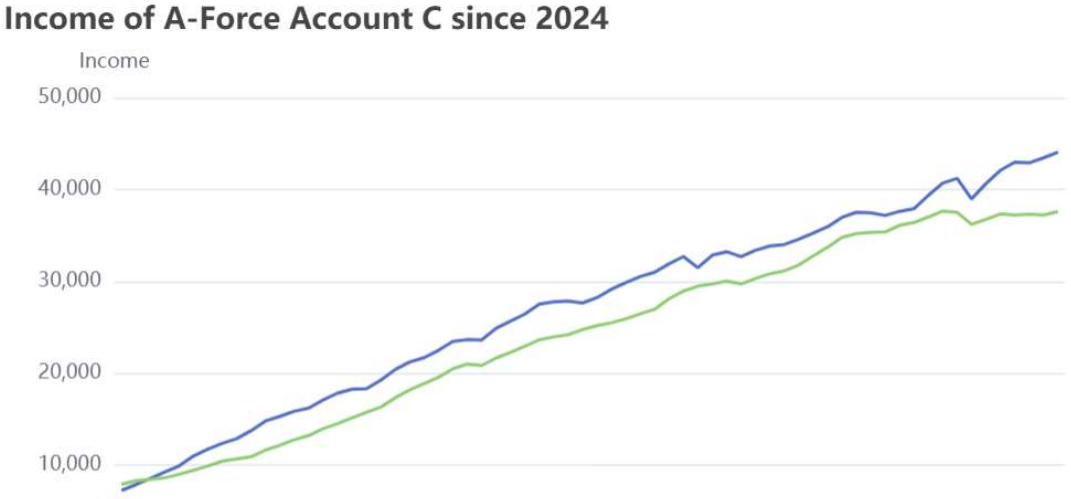
To improve data quality and the completeness of information, A-Force integrates data from different exchanges, eliminating differences between exchanges through deduplication, timestamp synchronization, and other methods, ensuring that the captured data is more precise. To enhance the complexity of multi-source data fusion, the A-Force platform can introduce concepts such as **weighted averages, deduplication mechanisms, and timestamp synchronization corrections**. The following formula demonstrates how data is integrated from multiple exchanges while removing duplicates, considering timestamp synchronization and data source weights.

$$D_{\text{fused}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(\frac{R_i \cdot (1-\alpha_i) + \alpha_i \cdot S_i}{1+\beta_i \cdot \delta_i} \right) \cdot \exp\left(-\frac{\left(T_{\text{sync}}^{(i)} - T_{\text{ref}}\right)^2}{\sigma_{\text{sync}}^2} \right)}{\sum_{i=1}^{N} W_i}$$

Formula Explanation:

- **DfusedD_{\text{fused}}:** The final fused data quality score, representing the quality of the dataset after integration, deduplication, and timestamp synchronization.
- **NN:** The number of exchanges involved in data fusion.
- **WiW_i:** The weight of the ii-th exchange. This weight is based on factors such as trading volume, data quality, or other considerations to evaluate the importance of each exchange and its influence in the fusion process.
- **RiR_i:** The data validity score for the ii-th exchange (e.g., accuracy, completeness of the data).
- **αi\alpha_i:** The deduplication factor that controls the precision of the deduplication process. This factor simulates the effect of the deduplication mechanism, with values ranging from 0⩽αi⩽10 \leq \alpha_i \leq 1, balancing the decision on whether to prioritize retaining duplicate data from a particular exchange.
- **SiS_i:** The market depth or historical transaction volume of the ii-th exchange, measuring the richness and reliability of the exchange's data.
- **βi\beta_i:** The timestamp synchronization correction factor for the ii-th exchange, used to adjust the influence of synchronization errors on the data fusion process.
- **δi\delta_i:** The redundancy ratio for the ii-th exchange, which follows a Beta distribution:
  δi∼Beta(α,β)\delta_i \sim \text{Beta}(\alpha, \beta).
  This term is used to simulate and reduce redundancy between data sources, avoiding the impact of duplicate data on the fusion quality.
- **Tsync(i)T_{\text{sync}}^{(i)}:** The timestamp for the ii-th exchange, indicating when the data was captured.
- **TrefT_{\text{ref}}:** The reference timestamp for data synchronization, typically the latest timestamp used as the synchronization baseline.
- **σsync2\sigma_{\text{sync}}^2:** The standard deviation of timestamp

synchronization errors, controlling the impact of timestamp synchronization accuracy.



Formula Analysis:

- **Weighted Average and Data Validity:**
  First, a weighted average is calculated for the fused data from all exchanges, where the data from each exchange is influenced by its weight WiW_i and validity RiR_i.
- **Deduplication Mechanism:**
  The deduplication factor αi\alpha_i controls the deduplication strategy. Specifically, if the impact of duplicate data from an exchange is small, the deduplication factor αi\alpha_i will be small, indicating a preference to retain data from that exchange. Otherwise, the deduplication factor avoids redundant data affecting the overall result.
- **Timestamp Synchronization Correction:**
  The timestamp synchronization error is computed as Tsync(i)−TrefT_{\text{sync}}^{(i)} - T_{\text{ref}}, and an exponential decay function exp⁡(−(Tsync(i)−Tref)2 σ sync2)\exp\left(-\frac{(T_{\text{sync}}^{(i)} - T_{\text{ref}})^2}{\sigma_{\text{sync}}^2}\right) is used to correct timestamp errors across different data sources, ensuring the final fused data maintains consistent time.
- **Reducing Redundant Data:**
  The redundancy ratio δi\delta_i follows a Beta distribution to effectively reduce the unnecessary impact of duplicate data on the fusion quality.

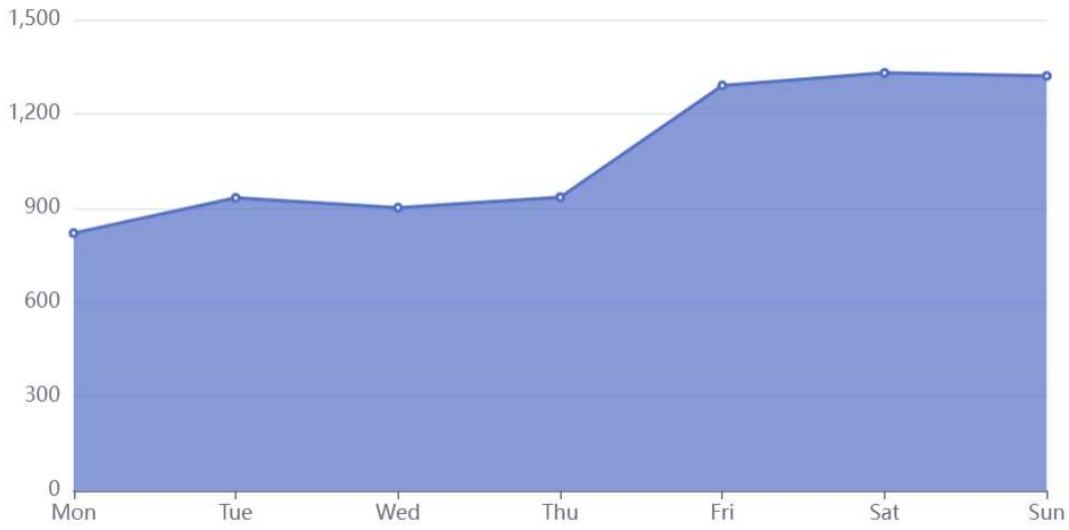
Data Synchronization and Processing

The A-Force platform supports simultaneous synchronization and processing of multiple data sources, ensuring that each data point is provided to the trading strategy engine within the same time window, preventing trading errors due to delays or data discrepancies. To better describe the mechanisms involved in data synchronization and processing on the A-Force platform, factors such as **time synchronization across multiple data sources, delay correction, synchronization error adjustment**, and **weighting** are introduced. The following formula takes into account the delays, synchronization time errors, and other factors between data sources, ensuring that synchronized data is delivered to the trading strategy engine on time.

$$T_{\text{sync}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(|T_{\text{timestamp}}^{(i)} - T_{\text{ref}}| + \alpha_i \cdot \left(\beta_i \cdot \left(\frac{T_{\text{latency}}^{(i)}}{1+\gamma_i} \right) \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \theta_j \cdot \int_0^{T_{\max}} \frac{dP_j(t)}{dt} \cdot \left(1 - \frac{t}{T_{\max}} \right) d$$

Formula Explanation:

- **TsyncT_{\text{sync}}:** Data synchronization error (in milliseconds), representing the total error when synchronizing data across multiple data sources on the A-Force platform.
- **NN:** The number of data sources involved in synchronization (i.e., multiple exchanges).
- **WiW_i:** The weight of the ii-th data source, indicating the importance of that data source (e.g., based on market depth, trading volume, etc.).
- **Ttimestamp(i)T_{\text{timestamp}}^{(i)}:** The timestamp for the ii-th data source, indicating when the data point was collected.
- **TrefT_{\text{ref}}:** The reference timestamp for data synchronization, typically the latest timestamp from a data source used as the baseline.
- **αi\alpha_i:** Calibration factor that reflects the calibration accuracy of the ii-th data source, with a range of 0⩽αi⩽10 \leq \alpha_i \leq 1, used to adjust synchronization precision.
- **βi\beta_i:** Delay correction factor for the ii-th data source, reflecting the degree of delay correction required.
- **Tlatency(i)T_{\text{latency}}^{(i)}:** The data capture latency for the ii-th data source (in milliseconds).
- **γi\gamma_i:** Random fluctuation factor, simulating random delays due to network fluctuations, trading volumes, etc.
- **MM:** The number of market data sources involved in synchronization.
- **θj\theta_j:** The synchronization priority weight for the jj-th market data source, representing its importance in the synchronization process.
- **∫0TmaxdPj(t)dt·(1−tTmax)dt\int_0^{T_{\text{max}}} \frac{dP_j(t)}{dt} \cdot \left(1 - \frac{t}{T_{\text{max}}} \right) dt:** This integral simulates the impact of market price changes on data synchronization. It calculates the rate of price change and combines it with a time decay factor (1−tTmax)\left(1 - \frac{t}{T_{\text{max}}} \right), simulating how market volatility affects synchronization delays.



Formula Analysis:

- **Weighted Average Synchronization Error:**
  For each data source ii, the timestamp error ∣Ttimestamp(i)−Tref∣ |T_{\text{timestamp}}^{(i)} - T_{\text{ref}}| is first calculated. Then, the weight WiW_i and delay correction factor are combined to ensure that the influence of data sources with varying importance is appropriately adjusted.
- **Delay Correction:**
  The delay correction factors βi\beta_i and γi\gamma_i consider network fluctuations and exchange load when synchronizing data. Delay impacts are corrected using weighted averages and calibration factors to ensure data synchronization and accuracy.
- **Impact of Market Price Changes:**
  The rate of change in market prices from the data sources is combined with the time decay factor to simulate the effect of market fluctuations on synchronization delays. This helps mitigate the negative effects of market volatility on data synchronization.

2 Intelligent Strategy Generation

Statistical Arbitrage and Machine Learning Integration: A-Force utilizes statistical arbitrage algorithms, combined with historical market data, trend analysis, and arbitrage models, to quickly identify price discrepancies and trading opportunities in the market. In addition, the platform uses machine learning algorithms (such as deep learning and reinforcement learning) to train and optimize these strategies. By continuously adjusting strategy parameters through model training, the platform adapts to dynamic market changes, providing higher accuracy and flexibility. To better describe how the A-Force platform integrates statistical arbitrage algorithms with machine learning models, it incorporates elements like historical data backtesting, price deviation detection, trend analysis, deep learning optimization, and reinforcement learning adjustments. The following formula takes into account the synergy between statistical arbitrage strategies and machine learning to discover market price deviations and optimize strategies.

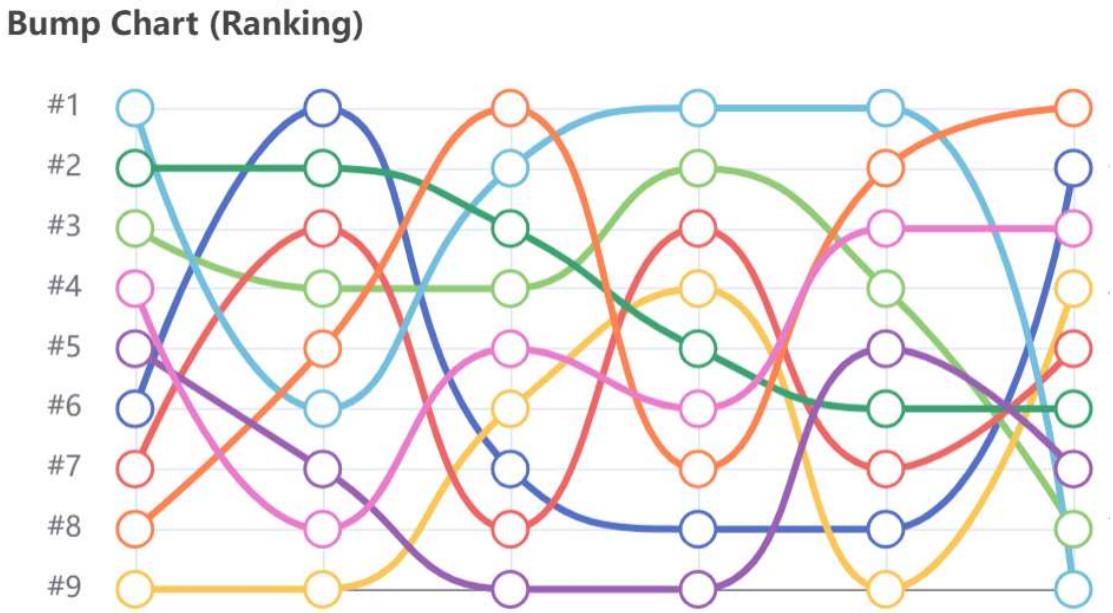
$$S_{\text{final}} = \frac{\sum_{i=1}^{N} \left(W_i \cdot \left(\text{sign}\left(P_{\text{diff}}^{(i)} \right) \cdot \alpha_i \cdot \exp\left(-\frac{(T_{\text{train}}^{(i)} - T_{\text{start}})^2}{\sigma_{\text{train}}^2} \right) \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j))$$

Formula Explanation:

- **Sfinal**: The final strategy output, representing the decision signal after being processed by statistical arbitrage algorithms and machine learning optimization.
- **NN**: The number of data sources involved in the arbitrage strategy (e.g., multiple market comparisons or historical data from different time windows).
- **Wi**: The weight of the i-th data source, representing the influence of that data source on the final strategy outcome. The weight is adjusted based on the reliability of the data source, market depth, and other factors.
- **Pdiff(i)**: The price difference for the i-th market comparison (e.g., arbitrage opportunities between markets), reflecting the degree of price deviation between markets.
- **sign(Pdiff(i))**: The sign function of price deviation, used to determine the direction of the arbitrage opportunity. If the price difference is positive, it indicates an arbitrage opportunity; otherwise, it is negative.
- **αi**: The arbitrage intensity adjustment factor for the i-th data source, used to adjust the strength of the arbitrage signal based on market volatility.
- **Ttrain(i)**: The training time window for the i-th data source, indicating the period used to train the strategy.
- **Tstart**: The strategy start time, used to calculate the difference between the training phase and the current time.
- **σtrain²**: The standard deviation of the training time window, representing the impact of the training phase on strategy optimization.
- **MM**: The number of states or actions in the reinforcement learning model (e.g., behavior or decisions under different strategy states).
- **θj**: The weight of the j-th state or action, representing the influence of each action or

state in the reinforcement learning model.

- **Rj**: The reward value for the j-th state or action, representing the return received after executing an action in that state within the machine learning model.
- **γj**: The discount factor in reinforcement learning, controlling the impact of future rewards on current decisions.
- **tj**: The time step corresponding to the j-th state or action, indicating the moment when the decision is executed.



Formula Analysis:

- **Weighted Average and Arbitrage Opportunity**: By using a weighted average, the price deviation **Pdiff(i)** and the corresponding signal strength are combined to determine the existence of an arbitrage opportunity, and each market signal is adjusted. The weight factor **Wi** can be adjusted based on factors like liquidity, market participation, and other variables to ensure more important data sources have a greater impact on the strategy.
- **Price Deviation Detection and Direction Judgment**: The function **sign(Pdiff(i))** determines the direction of the price deviation, and combined with the adjustment factor **αi**, it adjusts the execution strength of the arbitrage strategy.
- **Training Optimization and Dynamic Adjustment**: The difference between the training time window **Ttrain(i)** and the start time **Tstart** is used, along with a Gaussian decay function, to optimize the strategy's adaptability during different training phases. The strategy dynamically adapts to market changes based on past training.
- **Reinforcement Learning Reward Mechanism**: Through the reinforcement learning model, each state and action's reward value **Rj** is used, and with the discount factor **γj**, the strategy is dynamically adjusted. This ensures the flexibility and executability of the arbitrage strategy under various market conditions.

Adaptive Trading Strategy

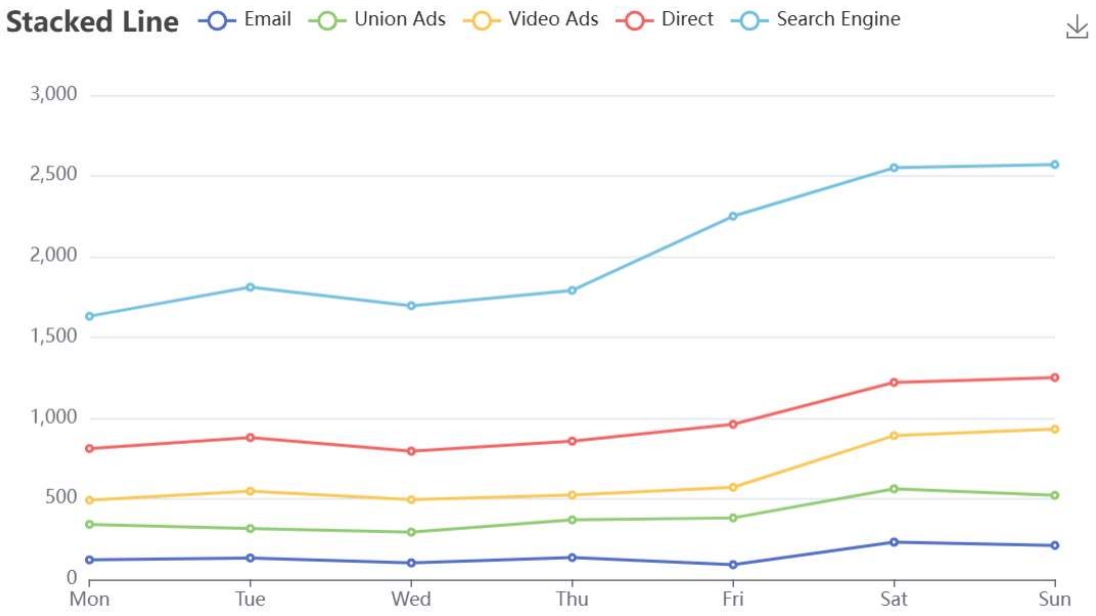
A-Force's trading strategies are not static; they are dynamically adjusted. When market volatility increases or unexpected events occur, the intelligent strategy engine automatically analyzes data and adjusts the trading strategies. These adjustments are made through real-time learning and optimization, enabling quick responses to market changes and improving the strategy's adaptability. To describe in more detail how A-Force dynamically adjusts trading strategies, the platform combines concepts like market volatility analysis, real-time learning, response to unexpected events, and adaptive optimization. The following formula incorporates adaptive weight adjustment, volatility calculation, dynamic parameter updates, and an unexpected event response mechanism to simulate how the platform rapidly adjusts trading strategies in response to market changes.

$$S_{\text{adjusted}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(\alpha_i \cdot P_{\text{volatility}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{update}}^{(i)} - T_{\text{ref}})^2}{\sigma_{\text{update}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j))$$

Formula Explanation:

- **Sadjusted**: The adjusted trading strategy output, representing the result of the dynamic adjustment of the strategy based on market changes (volatility, unexpected events, etc.).
- **NN**: The number of market data sources involved in the adaptive adjustment.
- **Wi**: The weight of the i-th data source, reflecting the influence of that data source on the final strategy output. This weight is dynamically adjusted based on factors like the data source's reliability and market depth.
- **αi**: The adjustment factor for the i-th data source, representing its adaptability in response to market volatility or unexpected events.
- **Pvolatility(i)**: The volatility indicator for the i-th market, reflecting the current volatility of that market, used to adjust the flexibility of the strategy.
- **Tupdate(i)**: The strategy update time for the i-th data source, used to control the time interval between strategy updates.
- **Tref**: The reference timestamp, indicating the last update time of the strategy.
- **σupdate²**: The standard deviation during the update process, representing the sensitivity of the strategy adjustment.
- **MM**: The number of states or actions in the reinforcement learning model, reflecting the learning process during strategy adjustment.
- **θj**: The weight of the j-th state or action, representing the influence of each action or state during the adaptive learning process.
- **Rj**: The reward value for the j-th state or action, representing the return or reward gained from executing a particular action in that state.
- **γj**: The discount factor in reinforcement learning, controlling the influence of future rewards on current decisions in the strategy adjustment.
- **tj**: The time step corresponding to the j-th action, indicating the time state when the action is executed.

Formula Analysis:



- **Market Volatility Adjustment**: Through weighted averages, the market volatility **Pvolatility(i)** and the adjustment factor **αi** are combined to dynamically adjust the strength of the trading strategy. When volatility increases, the corresponding adjustment factor increases to enhance the strategy's adaptability.
- **Adaptive Adjustment Factor**: Each data source's adaptive adjustment factor **αi** is dynamically optimized based on market changes. Using an exponential decay model, this ensures rapid strategy updates, especially in volatile markets.
- **Time Sensitivity**: By calculating the difference between the strategy update time **Tupdate(i)** and the reference timestamp **Tref**, along with a Gaussian decay function **exp(−((Tupdate(i)−Tref)²)/ σ update²)**, the strategy's update sensitivity is controlled. This ensures the strategy adapts quickly in fast-changing markets.
- **Reinforcement Learning Reward Mechanism**: Through the reinforcement learning model, the reward value **Rj** of each state and action is considered, combined with the discount factor **γj**, to enhance adaptive strategy adjustments. The discount factor controls the weight of future rewards, ensuring the strategy remains flexible in a dynamic environment.
- **Unexpected Event Response**: When an unexpected event occurs, the model quickly calculates its impact on market volatility and adjusts the strategy by using the adjustment factor **αi** and the reward mechanism **Rj** to swiftly respond to the event and correct the strategy.

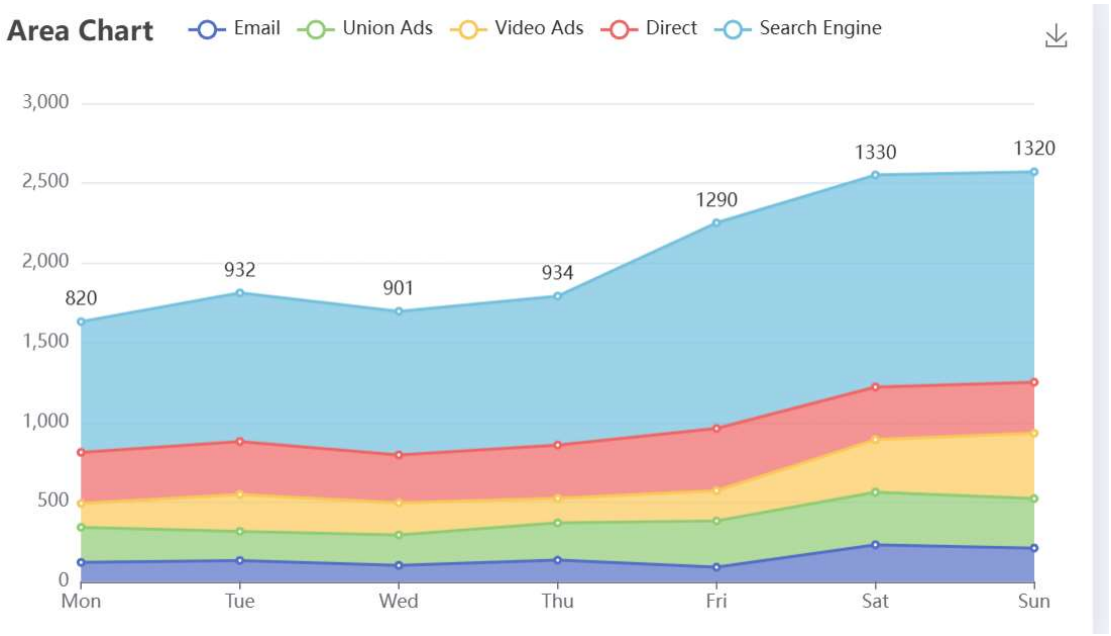
Real-time Strategy Optimization

The platform continuously analyzes the market in real time, using a backtesting system to validate and optimize trading strategies, ensuring the strategies can be consistently executed under different market conditions and can predict and mitigate potential risks in the market. To describe how the A-Force platform performs **real-time strategy optimization**, combining concepts like market analysis, backtesting systems, risk prediction and avoidance, the following formula covers several factors in real-time strategy optimization, including dynamic adjustments to market states, backtest validation, risk management, and optimization iterations.

$$S_{\text{optimized}} = \frac{\sum_{i=1}^{N} \left(W_i \cdot \left(P_{\text{profit}}^{(i)} \cdot \alpha_i \cdot \exp\left(-\frac{(T_{\text{backtest}}^{(i)} - T_{\text{start}})^2}{\sigma_{\text{backtest}}^2} \right) \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j))$$

Formula Explanation:

- **Soptimized**: The final optimized trading strategy, representing the strategy output after backtesting and risk-avoidance optimizations.
- **NN**: The number of market data sources involved in the strategy optimization.
- **Wi**: The weight of the i-th market data source, representing the influence of that data source on the optimized strategy output. The weight is dynamically adjusted based on market liquidity, accuracy, and other factors.
- **Pprofit(i)**: The profit indicator for the i-th market, reflecting the market's profit performance during past strategy executions.
- **αi**: The adjustment factor for the i-th data source, representing the dynamic adjustments based on the market performance.
- **Tbacktest(i)**: The backtest time window for the i-th data source, indicating the time period used to validate the strategy's effectiveness.
- **Tstart**: The strategy start time, used to calculate the difference between the backtesting process and the current moment.
- **σbacktest²**: The standard deviation during the backtest process, representing the stability of backtest results and the space for optimizing the strategy.
- **MM**: The number of states or actions in the reinforcement learning model, reflecting the adaptability of the strategy during risk avoidance and optimization.
- **θj**: The weight of the j-th state or action, representing its importance during risk prediction and backtest optimization.
- **Rj**: The reward value for the j-th state or action, representing the potential return from taking a specific action in that state.
- **γj**: The discount factor in reinforcement learning, used to adjust the impact of future rewards on current decisions.
- **tj**: The time step corresponding to the j-th state or action, indicating the time state when the decision is made.

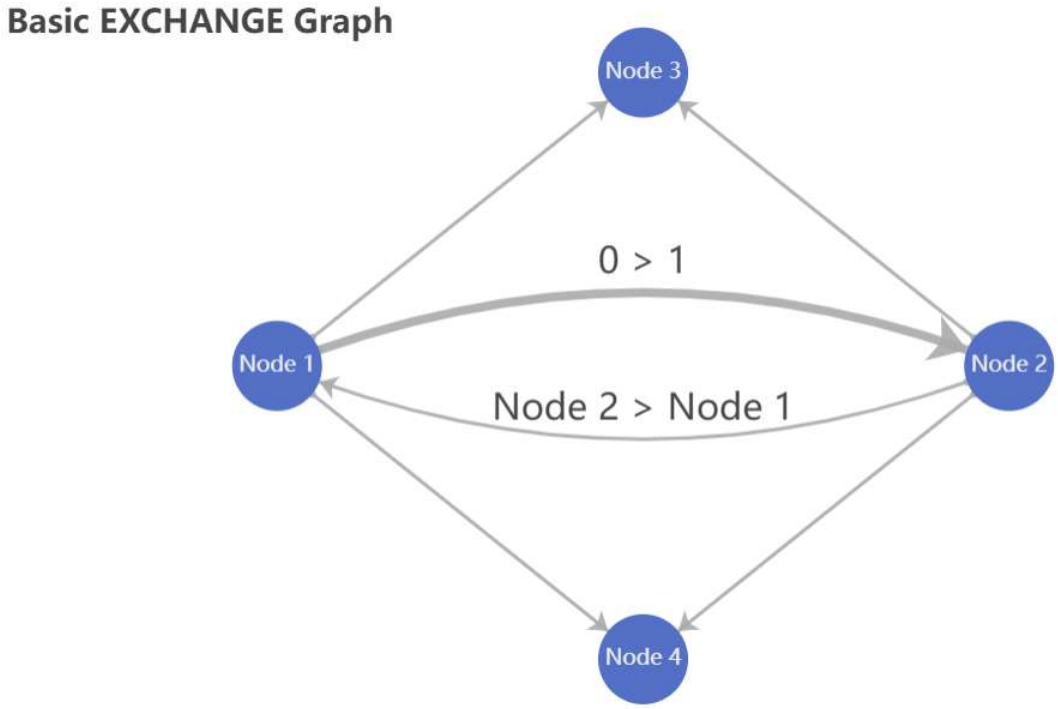


Formula Analysis:

- **Profit Indicator and Optimization Adjustment**: By using weighted averages, the profit performance **Pprofit(i)** of each market data source and the adjustment factor **αi** are combined to dynamically adjust the strength of strategy optimization. When a market performs well, the weight of the corresponding data source is increased, enhancing the execution of the optimized strategy.
- **Backtest Validation and Strategy Updates**: By calculating the difference between the backtest time window **Tbacktest(i)** and the strategy start time **Tstart**, along with a Gaussian decay function **exp(−((Tbacktest(i)−Tstart)²)/σbacktest²)**, the strategy is gradually validated and adjusted during backtesting, ensuring that the strategy adapts to different market environments.
- **Risk Prediction and Reward Mechanism**: During the backtesting process, combined with risk management strategies, the reinforcement learning model uses the reward value **Rj** and discount factor **γj** to predict and avoid risks. Through reinforcement learning, the strategy's risk management capabilities are continuously adjusted to mitigate potential risks.
- **Adaptive Strategy Adjustment**: By adjusting the state and action weights **θj** in reinforcement learning, the strategy is continuously adapted to handle different market environments while maximizing long-term returns.

3. Risk Control and Fund Management

Dynamic Risk Management: A-Force is equipped with a dynamic risk control module that monitors market volatility, trading volume, liquidity, and other factors in real time. It calculates risk exposure to manage the risk exposure of funds. This system dynamically adjusts the fund allocation based on market changes, ensuring that during periods of high market volatility, the positions are reduced to prevent losses due to sharp market fluctuations.



Complex Dynamic Risk Management Formula

To describe A-Force's **dynamic risk management**, the platform combines factors such as market volatility monitoring, risk exposure calculation, fund allocation adjustment, and market liquidity analysis. The following formula demonstrates how A-Force dynamically adjusts fund allocation based on real-time monitoring of market risks and reduces position risks in markets with high volatility.

$$R_{\text{managed}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(\frac{V_{\text{volume}}^{(i)} \cdot P_{\text{volatility}}^{(i)}}{1+\beta_i \cdot \sigma_{\text{risk}}^{(i)}} \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \theta_j \cdot \left(\frac{R_j \cdot \exp\left(-\gamma_j \cdot t_j\right)}{1+\kappa_j \cdot L_j} \right)$$

Formula Explanation:

- **Rmanaged**: The managed risk value, representing the risk exposure after dynamic fund adjustments under risk management.
- **NN**: The number of market data sources involved in risk management.
- **Wi**: The weight of the i-th market data source, reflecting its impact on the risk management strategy. This weight is dynamically adjusted based on market liquidity, trading volume, and other factors.
- **Vvolume(i)**: The trading volume for the i-th market, reflecting the current level of trading activity in that market.
- **Pvolatility(i)**: The volatility indicator for the i-th market, representing the current price fluctuation in that market, used to quantify market risk.
- **βi**: The adjustment factor, based on market liquidity and other factors, controlling the impact of market volatility on fund allocation.
- **σrisk(i)**: The risk exposure standard deviation for the i-th market, representing the potential risk fluctuations in that market.
- **MM**: The number of states or actions in the reinforcement learning model, reflecting the adaptability of the strategy in risk control.
- **θj**: The weight of the j-th state or action, representing the influence of each state in risk control.
- **Rj**: The reward value for the j-th state or action, representing the potential return from taking a specific action in that state.
- **γj**: The discount factor in reinforcement learning, used to adjust the impact of future rewards on current decisions.
- **tj**: The time step corresponding to the j-th state or action, indicating the time state when the action is taken.
- **κj**: The fund adjustment factor, controlling the flexibility of fund allocation adjustments.
- **Lj**: The liquidity indicator for the j-th market, representing the liquidity of that market under different conditions, influencing the fund allocation decision.

Formula Analysis:

- **Market Volatility and Trading Volume Adjustment**: By using weighted averages, the market volatility **Pvolatility(i)** and trading volume **Vvolume(i)** are combined to calculate market risk exposure. When market volatility is high, the corresponding fund allocation will be dynamically adjusted to reduce positions in high-risk markets.
- **Risk Exposure Standard Deviation Calculation**: By calculating the risk exposure standard deviation **σrisk(i)**, market risk fluctuations are quantified, which helps control the fund's risk exposure. Higher volatility increases risk exposure, thus affecting the degree of adjustment in fund allocation.
- **Fund Allocation Flexibility**: Through the reinforcement learning model, combined with states **θj** and actions **Rj**, the fund allocation strategy is adjusted. Under the influence of market liquidity **Lj** and the fund adjustment factor **κj**, positions are flexibly adjusted to reduce risk in highly volatile markets.
- **Dynamic Risk Adjustment**: Based on the risk exposure **σrisk(i)** for each market, and combined with fund allocation factors, the system automatically adjusts the fund distribution across markets. As market risk increases, the system will dynamically reduce the fund allocation to high-risk markets, thus lowering overall risk exposure.

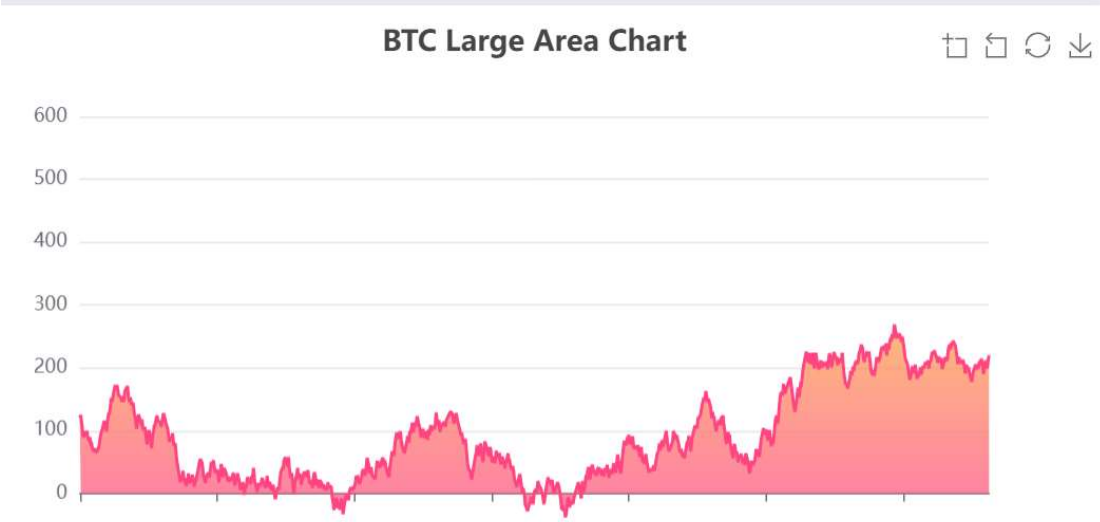
Intelligent Fund Allocation

The platform has a built-in fund allocation algorithm that automatically optimizes fund distribution based on real-time market conditions and the risk assessment of each trading strategy. For example, in high-risk markets, the system will allocate less capital, while in low-volatility markets, it will increase the fund allocation. To precisely describe A-Force's **intelligent fund allocation**, the platform introduces key concepts such as market risk assessment, fund allocation optimization, and volatility adjustment. The following formula demonstrates how the platform dynamically optimizes fund allocation based on real-time market conditions and the risk evaluation of each trading strategy, ensuring optimal fund distribution.

$$F_{\text{allocated}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(\frac{V_{\text{market}}^{(i)} \cdot (1-\alpha_i)}{1+\beta_i \cdot \sigma_{\text{risk}}^{(i)}} \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j))$$

Formula Explanation:

- **Fallocated**: The final allocated funds, representing the amount of funds allocated based on market conditions and risk assessments through the intelligent fund allocation algorithm.
- **NN**: The number of market data sources involved in fund allocation.
- **Wi**: The weight of the i-th market data source, reflecting its influence on fund allocation. This weight is dynamically adjusted based on market volatility and risk.
- **Vmarket(i)**: The liquidity or trading volume of the i-th market, reflecting the market's activity level and influencing the degree of fund allocation.
- **αi**: The adjustment factor, representing the reduction of fund allocation in high-risk markets. This factor simulates the impact of market volatility on fund investment and typically increases as market volatility increases.
- **βi**: The risk factor, controlling the degree to which market risk affects fund allocation, reflecting the potential risks of the market.
- **σrisk(i)**: The risk assessment value for the i-th market, calculated based on factors such as market volatility and potential risks.
- **MM**: The number of states or actions in the reinforcement learning model, representing strategy adjustments and optimization during the fund allocation process.
- **θj**: The weight of the j-th state or action, representing the influence of each state during fund allocation adjustments.
- **Rj**: The reward value for the j-th state or action, representing the potential return from taking a specific action in that state.
- **γj**: The discount factor in reinforcement learning, used to adjust the impact of future rewards on current decisions.
- **tj**: The time step corresponding to the j-th state or action, indicating the time state when the decision is made.



Formula Analysis:

- **Market Risk Assessment and Fund Adjustment**: Fund allocation is adjusted based on the market's liquidity (**Vmarket(i)**) and volatility (**σrisk(i)**). When market liquidity is higher, the fund allocation increases, while in high-risk markets, the system reduces fund allocation by adjusting the factor **αi**.
- **Volatility Control**: Through the volatility adjustment factor **βi**, the system dynamically adjusts fund allocation based on market risk levels. In low-volatility markets, fund allocation increases, while in high-volatility markets, the system reduces fund allocation to minimize risk exposure.
- **Reinforcement Learning Dynamic Optimization**: Through the reinforcement learning reward mechanism **Rj**, the platform optimizes fund allocation strategies based on the reward value of each state and action. The discount factor **γj** ensures the system flexibly adjusts based on future potential returns, optimizing long-term fund allocation benefits.
- **Adaptive Fund Adjustment**: With dynamic weights **Wi** and adjustment factors **αi**, fund allocation is optimized in real-time as market conditions and volatility change, enabling the platform to adapt to various market conditions and volatility levels.

Stop Loss and Take Profit Strategies

A-Force's fund management system allows dynamic stop-loss and take-profit points to ensure that users can lock in profits or limit losses as market conditions change. These stop-loss and take-profit points are not static but are dynamically adjusted based on real-time market data to provide maximum protection for users' assets. To describe A-Force's **dynamic stop-loss and take-profit strategies**, the platform incorporates key factors such as real-time market data, volatility calculation, and dynamic adjustment of stop-loss and take-profit points. The following formulas demonstrate how stop-loss and take-profit points are dynamically adjusted based on market changes and real-time data to protect assets and lock in profits.

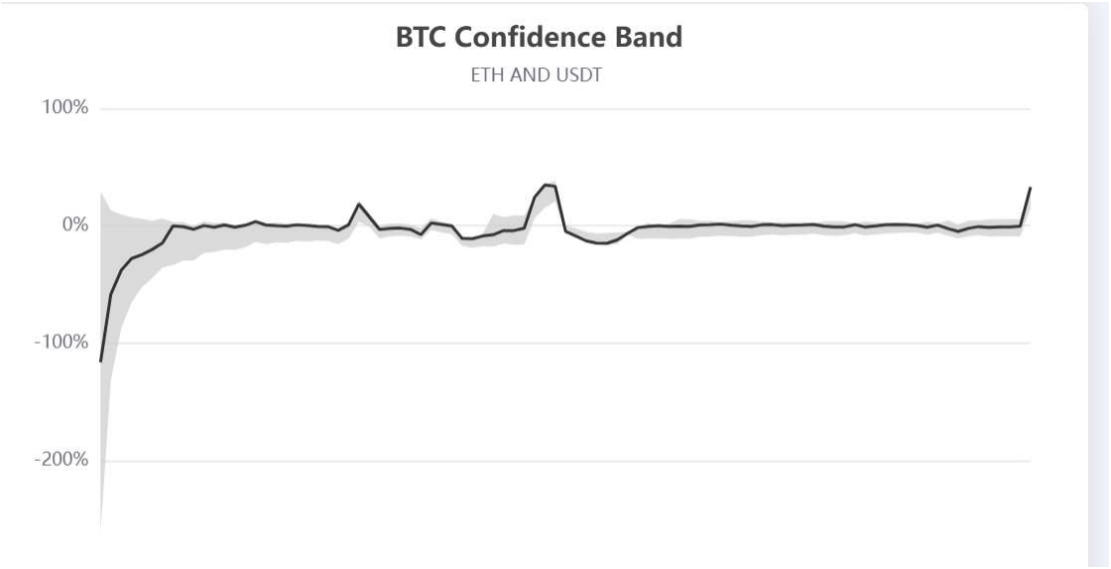
Formula Explanation:

$$S_{\text{stop-loss}} = P_{\text{entry}} \cdot \left(1 - \alpha_{\text{loss}} \cdot \exp\left(-\frac{(T^{(i)}_{\text{volatility}} - T_{\text{start}})^2}{\sigma^2_{\text{volatility}}}\right)\right)$$

$$S_{\text{take-profit}} = P_{\text{entry}} \cdot \left(1 + \alpha_{\text{profit}} \cdot \exp\left(-\frac{(T^{(i)}_{\text{volatility}} - T_{\text{start}})^2}{\sigma^2_{\text{volatility}}}\right)\right)$$

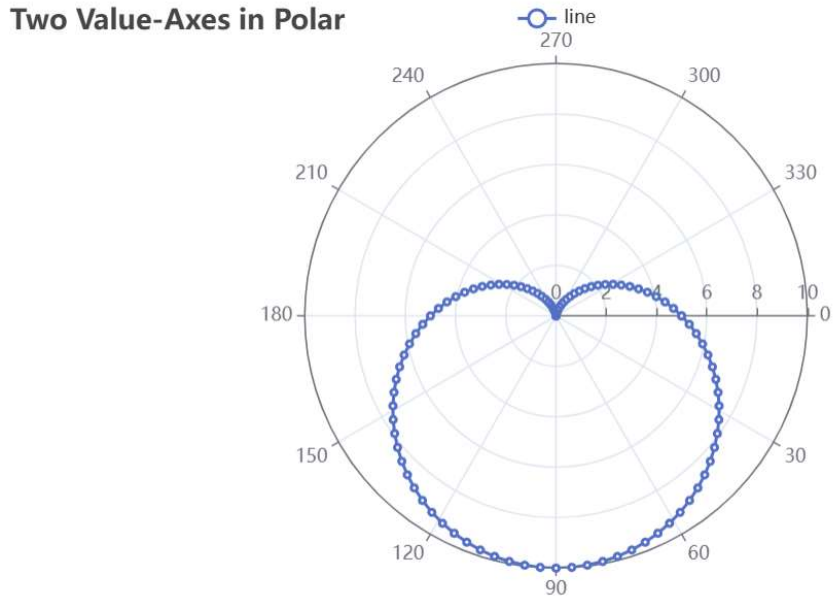
Stop Loss Formula - Sstop-loss:

- **Sstop-loss**: The dynamic stop-loss point, indicating the price level at which the platform automatically executes a stop-loss operation to limit losses when the market price falls to this level.
- **Pentry**: The entry price, i.e., the market price at the time of executing the trade.
- **αloss**: The stop-loss intensity factor, controlling the adjustment magnitude of the stop-loss point. This factor is dynamically adjusted based on market volatility, usually increasing during high market volatility to ensure the stop-loss point adapts to market fluctuations.
- **Tvolatility(i)**: The real-time volatility data for the i-th market, reflecting the market's price fluctuation.
- **Tstart**: The strategy start time, used to calculate the volatility change from entry time to the current time.
- **σvolatility²**: The standard deviation of volatility, representing the amplitude of market price fluctuations, which affects the dynamic adjustment of the stop-loss point.



Take Profit Formula - Stake-profit:

- **Stake-profit**: The dynamic take-profit point, indicating the price level at which the platform automatically executes a take-profit operation to lock in profits when the market price rises to this level.
- **Pentry**: The entry price of the user.
- **αprofit**: The take-profit intensity factor, controlling the adjustment magnitude of the take-profit point. This factor is dynamically adjusted based on market volatility, typically increasing when market prices rise to ensure the flexibility and responsiveness of the take-profit point.
- **Tvolatility(i)**: The real-time volatility data for the i-th market, reflecting the market's price fluctuation.
- **Tstart**: The strategy start time, used to calculate the volatility change from entry time to the current time.
- **σvolatility²**: The standard deviation of volatility, representing the amplitude of market price fluctuations, which affects the dynamic adjustment of the take-profit point.



Formula Analysis:

- **Dynamic Adjustment of Stop Loss and Take Profit Points**: Both stop-loss **Sstop-loss** and take-profit **Stake-profit** points are dynamically adjusted based on market volatility. When volatility increases, the stop-loss point is appropriately lowered, and the take-profit point is raised, ensuring the strategy can adapt flexibly to market fluctuations.
- **Impact of Volatility**: By monitoring market volatility, the dynamic stop-loss and take-profit points can automatically adjust during periods of significant price fluctuations. For instance, in highly volatile markets, the system can lock in profits or reduce losses more quickly, enhancing asset safety.
- **Efficient Fund Management**: This system can automatically calculate and adjust stop-loss and take-profit points, ensuring users' assets are protected during periods of high market volatility, preventing unnecessary losses due to sudden market changes.
- **Enhanced Flexibility**: By using the intensity factors **αloss** and **αprofit**, the adjustment magnitude of the stop-loss and take-profit points is dynamically modified in real-time based on market conditions. During high volatility, these factors increase, making the stop-loss and take-profit points more flexible and responsive to market changes.

4. Multithreaded Concurrent Computing

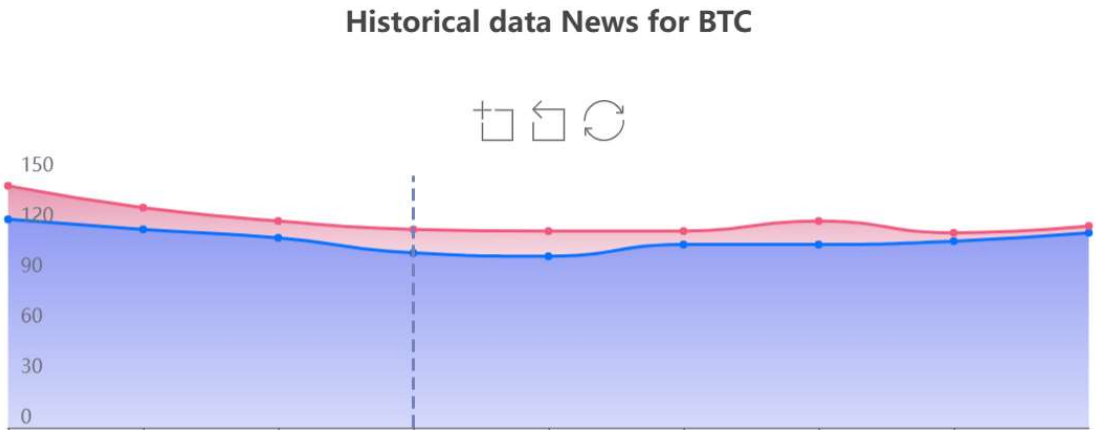
Parallel Strategy Processing

A-Force uses a multithreaded architecture that allows multiple independent trading strategy modules to run simultaneously. These strategy modules can execute in their respective threads, processing more trading tasks concurrently. Each strategy can independently make decisions based on different market conditions, thus increasing the overall system throughput and execution efficiency. To describe how A-Force implements **parallel strategy processing** using a multithreaded architecture, we introduce concepts like thread scheduling, strategy execution optimization, resource allocation, and execution efficiency improvement. The following formula demonstrates how parallel execution of multiple strategy modules enhances system throughput and execution efficiency.

$$T_{\text{total}} = \sum_{i=1}^{N} \left(\frac{W_i \cdot \left(T_{\text{task}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{load}}^{(i)} - T_{\text{opt}})^2}{\sigma_{\text{load}}^2} \right) \right)}{1 + \gamma_i \cdot \delta_i} \right)$$

Formula Explanation:

- **Ttotal**: The total strategy processing time, representing the total time required to process multiple strategies in parallel.
- **NN**: The number of parallel strategy modules being executed.
- **Wi**: The weight of the i-th strategy module, representing its resource consumption and importance in the overall processing.
- **Ttask(i)**: The execution time of the i-th strategy module, representing the time required to execute that strategy.
- **Tload(i)**: The system load for the i-th strategy module, reflecting the computational resources consumed by the strategy module during execution.
- **Topt**: The system's optimal load time, indicating the ideal state where system load is minimized.
- **σload²**: The load standard deviation, controlling the volatility of system load changes, which is used to adjust the efficiency of task execution.
- **γi**: The resource scheduling factor, reflecting the flexibility in resource allocation for the strategy module.
- **δi**: The execution efficiency factor for the i-th strategy module, indicating the efficiency adjustment in parallel execution. The higher the value, the higher the execution efficiency of the strategy.



Formula Analysis:

- **Total Execution Time of Parallel Strategies**: The total time for parallel execution is calculated by considering each strategy module's execution time **Ttask(i)** and its weight **Wi**, while dynamically adjusting according to the load optimization factor. Each parallel strategy module will adjust its execution efficiency based on the load condition and the system's optimal load time **Topt**, ensuring efficient execution across strategies.
- **Load Adjustment and Optimization**: The difference between the system load **Tload(i)** and the optimal load time **Topt** is adjusted using a Gaussian decay factor **exp(−((Tload(i)−Topt)²)/σload²)**, optimizing each strategy's execution efficiency. This adjustment helps reduce unnecessary computational burden when the system load is too high, thus optimizing the overall execution time.
- **Resource Allocation and Scheduling Factor**: Through the resource scheduling factor **γi** and the execution efficiency factor **δi**, the system can dynamically allocate computing resources based on each strategy's needs and efficiency, ensuring efficient execution of the strategies.
- **Throughput and Efficiency Improvement**: By executing strategies in parallel through multithreading, the system can process multiple strategy tasks simultaneously, reducing execution time and increasing overall trading strategy throughput and execution efficiency. This parallel mechanism enables the A-Force platform to handle more trading tasks and make independent decisions based on different market conditions.

Efficient Resource Allocation

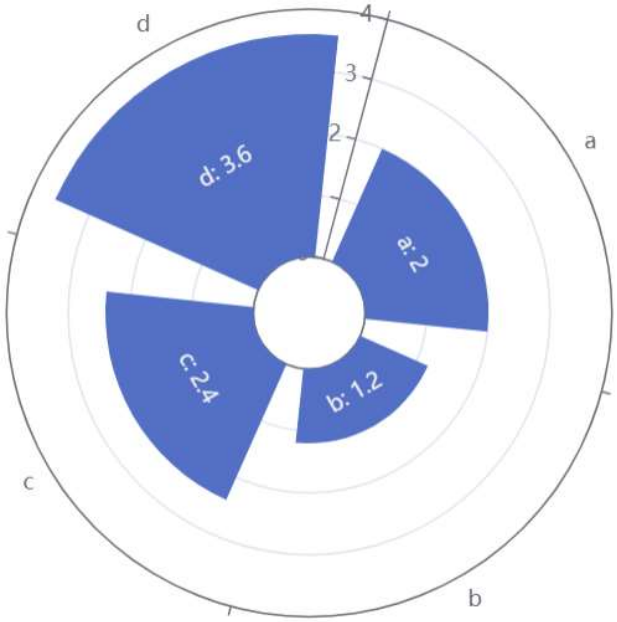
In multithreaded computing, the A-Force system dynamically allocates resources, separating compute-intensive tasks (such as real-time data analysis, model training) from I/O-intensive tasks (such as data transfer, order submission). This ensures that tasks are completed efficiently without interference. To describe how the A-Force system achieves **efficient resource allocation** in multithreaded computing, we introduce factors such as the allocation of compute-intensive and I/O-intensive tasks, task priority, and resource scheduling. The following formula shows how resources are dynamically allocated based on task types to ensure efficient execution without mutual interference.

$$T_{\text{total}} = \sum_{i=1}^{N} \left(W_i \cdot \left(\frac{T_{\text{comp}}^{(i)} \cdot \alpha_{\text{comp}}}{1 + \gamma_i \cdot T_{\text{I/O}}^{(i)}} \right) + \sum_{j=1}^{M} \theta_j \cdot \left(R_j \cdot \exp\left(-\frac{(T_{\text{task}}^{(j)} - T_{\text{opt}})^2}{\sigma_{\text{task}}^2} \right) \right) \right)$$

Formula Explanation:

- **Ttotal**: The total resource allocation time, representing the total time required after allocating resources based on task type and resource demand in a multithreaded environment.
- **NN**: The number of compute-intensive tasks (e.g., real-time data analysis, model training).
- **Wi**: The weight of the i-th compute-intensive task, representing its relative importance in the overall resource allocation.
- **Tcomp(i)**: The execution time of the i-th compute-intensive task, representing the time required to execute the task.
- **αcomp**: The resource demand coefficient for compute-intensive tasks, reflecting the extent of resource consumption for that task.
- **γi**: The resource scheduling factor, indicating the allocation priority between compute-intensive tasks and I/O-intensive tasks.
- **TI/O(i)**: The I/O time for the i-th task, representing the time consumed by I/O operations.
- **MM**: The number of I/O-intensive tasks (e.g., data transfer, order submission).
- **θj**: The weight of the j-th I/O-intensive task, representing its demand for resource allocation.
- **Rj**: The execution time of the j-th I/O-intensive task, representing the time required to execute that task.
- **Ttask(j)**: The workload time of the j-th I/O-intensive task, representing the task's load.
- **Topt**: The optimal load time, representing the ideal state in which the system can execute tasks efficiently.
- **σtask²**: The standard deviation of the task load, representing potential fluctuations in I/O-intensive tasks.

TOP FOR EXCHANGE FOR BTC



Formula Analysis:

- **Resource Allocation for Compute-intensive Tasks**: For each compute-intensive task, the system allocates resources based on its execution time **Tcomp(i)** and resource demand coefficient **αcomp**. The execution time of compute-intensive tasks is affected by I/O tasks (**γi·TI/O(i)**), and the system allocates computational resources based on task priority to prevent resource contention between compute and I/O tasks.
- **Handling I/O-intensive Tasks**: For each I/O-intensive task, the system adjusts resource allocation based on task weight **θj**. The execution time **Ttask(j)** and workload time **Topt** influence the resource scheduling strategy, ensuring that I/O tasks do not interfere with compute-intensive tasks during execution.
- **Resource Scheduling Optimization**: Using the resource scheduling factor **γi** and task load fluctuation **σtask²**, the system dynamically optimizes resource allocation based on task demands and load states. Task priority **θj** and workload are dynamically adjusted during scheduling, ensuring efficient parallel execution of both compute and I/O tasks.
- **Load Balancing and Task Distribution**: The system optimizes load distribution using the Gaussian decay factor **exp(−((Ttask(j)−Topt)²)/σtask²)**. Task loads are executed when the system reaches its optimal load state, maximizing resource utilization and minimizing task interference. This ensures that both compute-intensive and I/O-intensive tasks run efficiently in parallel, improving overall system performance.

Load Balancing and Fault Tolerance Mechanism

A-Force's multithreaded architecture supports load balancing, automatically adjusting system resources based on the running status and computational needs of each thread. If a thread encounters a failure or timeout, the system quickly switches to a backup thread through its fault tolerance mechanism, ensuring that a single point of failure does not bring down the entire system. To describe the **load balancing and fault tolerance mechanism** in A-Force's multithreaded architecture, we introduce key factors such as thread scheduling, resource allocation, fault detection, and switching. The following formula demonstrates how load balancing is dynamically carried out in a multithreaded environment, and how the system quickly switches to a backup thread in case of a thread failure, ensuring system stability and high availability.

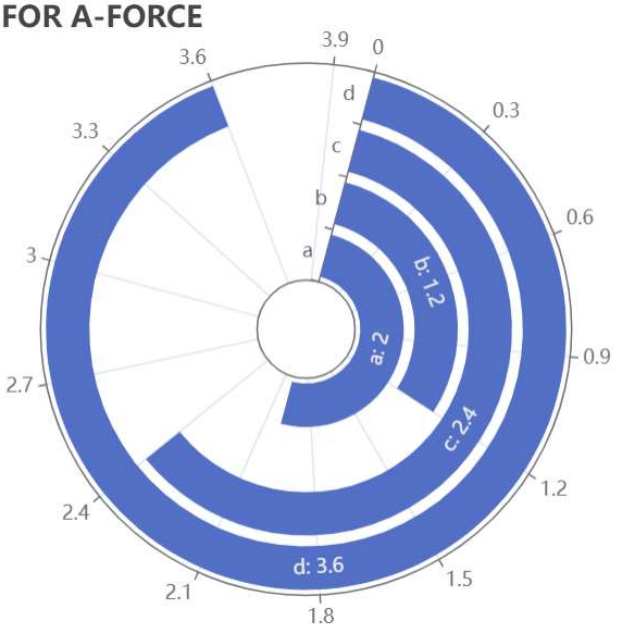
$$T_{\text{adjusted}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{task}}^{(i)} \cdot \left(1 + \alpha_i \cdot \exp\left(-\frac{(T_{\text{load}}^{(i)} - T_{\text{opt}})^2}{\sigma_{\text{load}}^2} \right) \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot (R_j \cdot \exp\left(-\gamma_j \cdot t_j\right)) \cdot (1 - \delta_j) \right)$$

Formula Explanation:

- **Tadjusted**: The adjusted task processing time, representing the total task execution time after resource adjustments under load balancing and fault tolerance mechanisms.
- **NN**: The number of threads involved in load balancing and fault tolerance.
- **Wi**: The weight of the i-th thread, representing its computational demand and resource consumption in the overall system.
- **Ttask(i)**: The task execution time for the i-th thread, indicating the time required for that thread to complete its task.
- **αi**: The thread scheduling factor, reflecting the adjustment strength of the thread's load. This factor is usually increased when the load is heavy to balance the load.
- **Tload(i)**: The current load of the i-th thread, reflecting the computational resources consumed by the thread.
- **Topt**: The optimal load time, indicating the ideal state where system load is minimized to ensure thread load balancing.
- **σload²**: The load standard deviation, controlling the magnitude of load fluctuations to ensure that load changes are not too large.
- **MM**: The number of backup threads, which are used as alternatives to handle thread failures.
- **θj**: The weight of the j-th backup thread, indicating its importance in the fault tolerance mechanism.
- **Rj**: The resource consumption of the j-th backup thread, reflecting the computational resources required for the backup thread to execute its task.
- **γj**: The discount factor in the fault tolerance mechanism, used to control the timeliness and responsiveness of switching to backup threads.
- **tjt**: The switching delay for the j-th backup thread, indicating the time needed for the backup thread to start running after a failure occurs.
- **δj**: The fault occurrence factor, indicating the impact of a thread failure on the backup

thread switch. If a thread fails, **δj = 1**; otherwise, **δj = 0**.

6 TOP EXCHANGE FOR A-FORCE



Formula Analysis:

- **Load Balancing Adjustment**: The load **Tload(i)** of each thread is compared with the optimal load **Topt**, and adjustments are made through a Gaussian decay function. Higher loads will increase the thread scheduling factor **αi**, optimizing the load balance between threads and preventing overloading of any single thread.
- **Resource Allocation and Thread Priority**: The system dynamically allocates computational resources based on each thread's weight **Wi** and task execution time **Ttask(i)**. Threads with higher priority will receive more resources, enhancing task execution efficiency.
- **Backup Thread Switching**: When a thread fails, the system quickly switches to a backup thread via the fault tolerance mechanism. The resource consumption **Rj** and switching delay **tjt** of the backup thread affect the efficiency of the fault tolerance process. By adjusting the discount factor **γj** and fault occurrence factor **δj**, the system can quickly deploy backup threads to minimize the impact of the failure.
- **Fault Detection and Recovery**: When a thread detects a failure, the fault tolerance mechanism activates the backup thread. The weight **θj** of the backup thread is adjusted to ensure the system can quickly recover. If a thread cannot execute normally, the backup thread automatically takes over the task, ensuring the system continues to run without being affected by the failure.

This system ensures **high availability** by allowing the A-Force platform to dynamically balance load and switch to backup threads in case of failures, maintaining continuous operation even in the event of thread malfunctions.

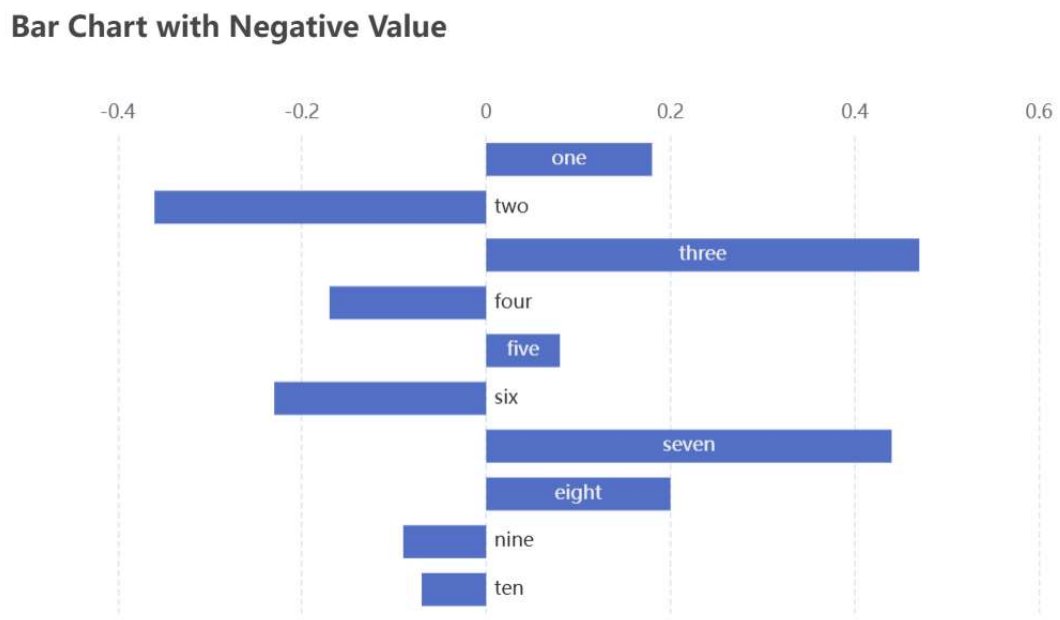
Fast Response Mechanism

In high-frequency trading, even milliseconds of delay can lead to trade failures. A-Force uses optimized multithreading and asynchronous task handling mechanisms to complete trade decisions and executions with ultra-low latency, ensuring efficient and stable trading performance even in high-pressure environments. To describe A-Force's **fast response mechanism** in high-frequency trading, we incorporate key factors such as multithreading optimization, asynchronous task handling, and latency reduction. The following formula demonstrates how optimizing resource allocation and task handling maximizes trading execution efficiency, reduces latency, and ensures the system's stability and responsiveness under pressure in market environments.

$$T_{\text{response}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{task}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{load}}^{(i)} - T_{\text{opt}})^2}{\sigma_{\text{load}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} (\theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j)) \cdot (1 - \delta_j))$$

Formula Explanation:

- **Tresponse**: The final trade response time, representing the time it takes to complete trade decision-making and execution under multithreading optimization and asynchronous task handling.
- **NN**: The number of threads involved in task execution.
- **Wi**: The weight of the i-th task, indicating its importance during execution.
- **Ttask(i)**: The execution time of the i-th task, representing the time required to complete that task.
- **Tload(i)**: The current load of the i-th thread, indicating the computational resources consumed by that thread.
- **Topt**: The optimal load time, representing the ideal execution time when the system is in its best load state.
- **σload²**: The load standard deviation, controlling the magnitude of load fluctuations, ensuring load changes are not excessive, thus optimizing response time.
- **MM**: The number of asynchronous tasks, indicating the number of tasks that need to be processed via asynchronous mechanisms (e.g., market data retrieval, trade submissions).
- **θj**: The weight of the j-th asynchronous task, indicating its impact on the response process.
- **Rj**: The execution time of the j-th asynchronous task, representing the time required to execute that task.
- **γj**: The delay factor in asynchronous tasks, used to adjust the impact of task response delays.
- **tjt**: The processing delay for the j-th asynchronous task, indicating the time required to complete the task.
- **δj**: The fault occurrence factor for asynchronous tasks. If a task fails, **δj = 1** (indicating a need for re-scheduling), otherwise **δj = 0**.



Formula Analysis:

- **Task Execution and Load Adjustment**: Each thread's task execution time **Ttask(i)** and current load **Tload(i)** are dynamically adjusted to minimize response time under high market pressure. Load optimization is achieved using a Gaussian decay factor **exp⁡(−((Tload(i)−Topt)²)/σload²)**, ensuring that each task adjusts based on the system load, preventing task backlog and optimizing task execution efficiency.
- **Asynchronous Tasks and Latency Optimization**: The execution time **Rj** and delay factor **γj** for asynchronous tasks adjust the response time. Asynchronous tasks, often involving I/O operations (e.g., data retrieval, order submission), can be optimized by adjusting the execution sequence and delay factor, enabling the system to quickly respond to market changes while reducing blocking and waiting times.
- **Fault Recovery and Task Rescheduling**: When an asynchronous task fails, the fault occurrence factor **δj** triggers the fault tolerance mechanism, allowing the system to dynamically adjust task priority or reschedule backup threads to resume task execution, ensuring that the system can quickly recover and respond to market changes under any circumstance.
- **Efficient Resource Allocation**: The combination of multithreading and asynchronous handling ensures efficient resource utilization. By adjusting task priorities **Wi** and the weight of asynchronous tasks **θj**, resources are allocated based on each task's urgency and execution time. High-priority tasks (such as trade decision-making and market volatility analysis) are allocated more computational resources, ensuring they are completed swiftly and efficiently.

Chapter 2: Real-time Data Processing Engine

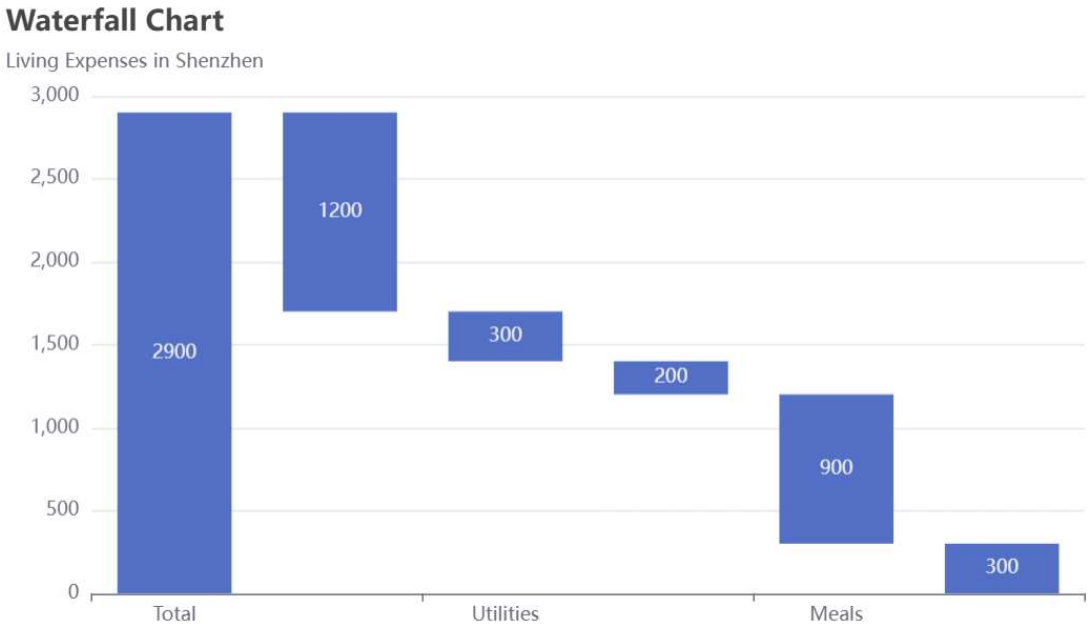
1 Ultra-Low Latency Data Collection

WebSocket and **FIX Protocol**: A-Force utilizes **WebSocket** and **FIX** (Financial Information eXchange) protocol for efficient data transmission. **WebSocket** provides a persistent, bidirectional communication link, enabling real-time data exchange between the server and the client. The **FIX** protocol, as the industry standard for financial markets, ensures efficient and reliable data transmission. To describe how the A-Force platform efficiently leverages **WebSocket** and **FIX** protocol for data transmission, the following formula demonstrates the collaboration between the two in data exchange. By considering **real-time bidirectional communication**, **data transmission efficiency**, and **protocol optimization**, we showcase how to maximize the efficiency and reliability of data exchange.

$$T_{\text{data}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(\frac{T_{\text{packet}}^{(i)} \cdot \alpha_i}{1+\gamma_i \cdot \delta_i} \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \cdot \left(1 - \delta_j \right) \right)$$

Formula Explanation: • **Tdata**: Total data transmission time, representing the data transfer time during the collaboration of **WebSocket** and **FIX** protocol.
• **NN**: Number of data packets (each **WebSocket** frame or **FIX** message).
• **Wi**: Weight of the i-th data packet, representing the importance of that packet in the entire data transfer process. It is usually allocated based on factors such as packet size and transmission priority.
• **Tpacket(i)**: Transmission time of the i-th packet, indicating the time required to transmit that packet.
• **αi**: Data packet transmission efficiency factor, indicating the optimization level of each data packet. Efficient data packet transmission reduces the transmission time **Tpacket(i)**.
• **γi**: Transmission delay factor, representing the impact of network delay or protocol parsing time.
• **δi**: Fault occurrence factor, representing the probability of errors or packet loss during transmission, requiring retransmission.
• **MM**: Number or type of **FIX** protocol messages.
• **θj**: Weight of the j-th **FIX** protocol message, reflecting its importance in data exchange.
• **Rj**: Transmission time of the j-th **FIX** protocol message, representing the time required to send and receive the message.
• **tj**: Network delay of the j-th message, indicating the delay time of the data packet during transmission.

• **δj**: Error handling factor in the **FIX** protocol, representing retransmission delay in case of data loss or parsing errors.



Formula Analysis: • **Data Packet Transmission Optimization**: The transmission time **Tpacket(i)** is calculated using a weighted average, and dynamic adjustments are made according to the optimization factor **αi**. Due to its low-latency feature, **WebSocket** allows for fast data transmission in low-latency environments, while the **FIX** protocol ensures efficient and reliable data transfer.

• **Delay and Network Fluctuation Adjustment**: The delay of each data packet is influenced by the **γi** and **δi** factors, representing delays caused by network fluctuations or transmission faults. These factors are adjusted through optimization algorithms that fine-tune network connections and protocol handling, improving the stability of data transfer.

• **FIX Protocol Reliability Assurance**: In the **FIX** protocol, the message transmission time **Rj** and network delay **tj** are carefully managed to ensure efficient and secure transmission of trading information. Additionally, retransmission mechanisms and error handling factors **δj** ensure the integrity and accuracy of data transmission, preventing packet loss or misdelivery.

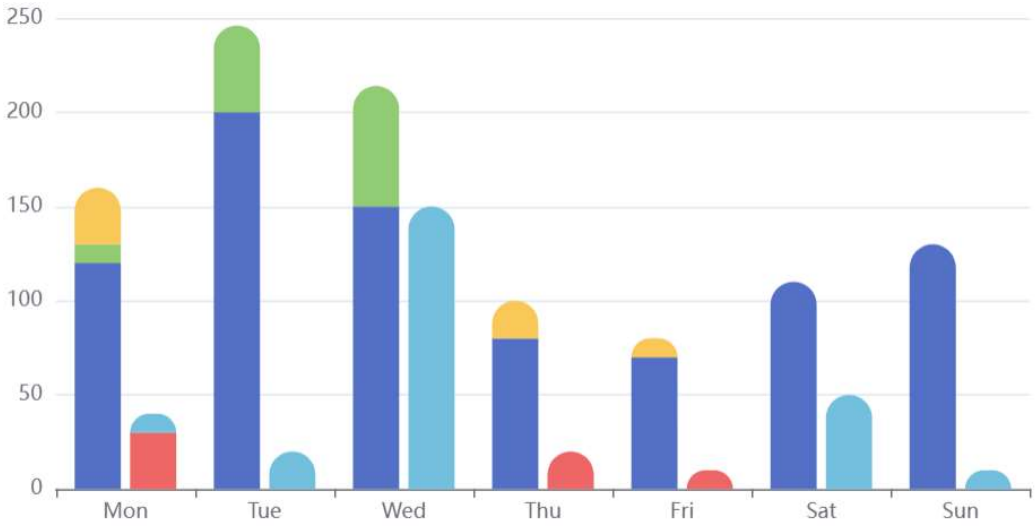
• **Resource Scheduling and Optimization**: By dynamically adjusting the weights **Wi** of each data packet and **FIX** message **θj**, the system can optimize data transmission in real time based on market conditions. For example, trade orders and market updates typically require priority transmission, while high-frequency quote data can be allocated different bandwidth and priority levels based on network conditions.

Millisecond-Level Latency:

Through these advanced communication protocols, A-Force achieves millisecond-level latency, enabling market data to be transmitted from exchanges to the platform in a very short time. This ensures that users can obtain real-time market updates and make timely trading decisions. To describe how the A-Force platform utilizes advanced communication protocols to achieve **millisecond-level latency**, the following formula demonstrates how **WebSocket** and **FIX protocol** optimization, along with **network latency management** and **improved transmission efficiency**, ensure that the platform can receive market data and make decisions in the shortest possible time.

$$T_{\text{delay}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{packet}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{latency}}^{(i)} - T_{\min})^2}{\sigma_{\text{latency}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation: • **Tdelay**: Total delay time, representing the time required for data transmission from the exchange to the platform (unit: milliseconds).
• **NN**: Number of data packets, representing each market data packet received from the exchange.
• **Wi**: Weight of the i-th data packet, representing the importance of that packet in the overall delay calculation.
• **Tpacket(i)**: Transmission time of the i-th packet, indicating the time required to transmit each data packet.
• **Tlatency(i)**: Latency time of the i-th packet, reflecting the delay in data transmission from the exchange to the platform.
• **Tmin**: Ideal minimum latency time, representing the minimum latency in the platform's optimal state.
• **σlatency²**: Standard deviation of latency, controlling the amplitude of latency fluctuations to ensure delay stability.
• **MM**: Number or type of **FIX** protocol messages, representing the number of messages the platform receives.
• **θj**: Weight of the j-th **FIX** protocol message, representing the impact of the message on the delay calculation.
• **Rj**: Transmission time of the j-th **FIX** protocol message, indicating the time required to send and receive the message.
• **γj**: Delay factor in the **FIX** protocol, representing the delay caused by **FIX** protocol parsing or transmission.
• **tj**: Network delay of the j-th **FIX** message, indicating the latency during its network transmission.
Formula Analysis: • **Data Packet Delay Optimization**: The transmission time **Tpacket(i)** and latency time **Tlatency(i)** of each data packet are dynamically adjusted using a Gaussian decay factor **exp(−((Tlatency(i)−Tmin)²/σlatency²))**. Through this adjustment, the system reduces the impact of latency fluctuations, ensuring stable low latency in high-frequency trading.



• **FIX Protocol Delay Management**: Through the transmission time **Rj** and network delay **tj** of **FIX** protocol messages, the platform can reduce delays caused by protocol parsing and data transmission within the **FIX** protocol's efficient data transfer framework. The delay factor **γj** dynamically adjusts the impact of network delay, ensuring the rapid transmission of packets and messages.

• **Load and Delay Control**: The system dynamically adjusts the weights **Wi** of each data packet and **FIX** message **θj** to ensure that market data is quickly transmitted and promptly reaches the decision-making system. When market data fluctuates, the system can respond quickly and optimize network latency and data transmission time.

High Throughput and Reliability

A-Force's data collection system can simultaneously handle real-time market data from multiple data sources (multiple exchanges and markets), maintaining low latency and high throughput in high-frequency trading environments, thus avoiding trading losses caused by latency. To describe how the A-Force platform achieves **high throughput and reliability** in high-frequency trading environments, we introduce factors such as **multi-data source processing**, **latency control**, **throughput optimization**, and **data reliability**. The following formula demonstrates how the system efficiently processes data in the real-time market data stream from multiple exchanges and markets, ensuring trading reliability and low latency.

$$T_{\text{throughput}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{data}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{latency}}^{(i)} - T_{\min})^2}{\sigma_{\text{latency}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation: • **Tthroughput**: Total throughput, representing the system's ability to process real-time market data in a high-frequency trading environment.

• **NN**: Number of data sources, representing the real-time data sources from multiple exchanges and markets.

• **Wi**: Weight of the i-th data source, representing the contribution of that data source to the system's throughput.

• **Tdata(i)**: Data transmission amount of the i-th data source, representing the time or volume of data processed from that data source.

• **Tlatency(i)**: Latency of the i-th data source, representing the data transmission delay from the data source to the platform.

• **Tmin**: Minimum latency of the system, representing the lowest latency of the platform in an ideal scenario.

• **σlatency²**: Standard deviation of latency, controlling the amplitude of latency fluctuations to optimize throughput stability.

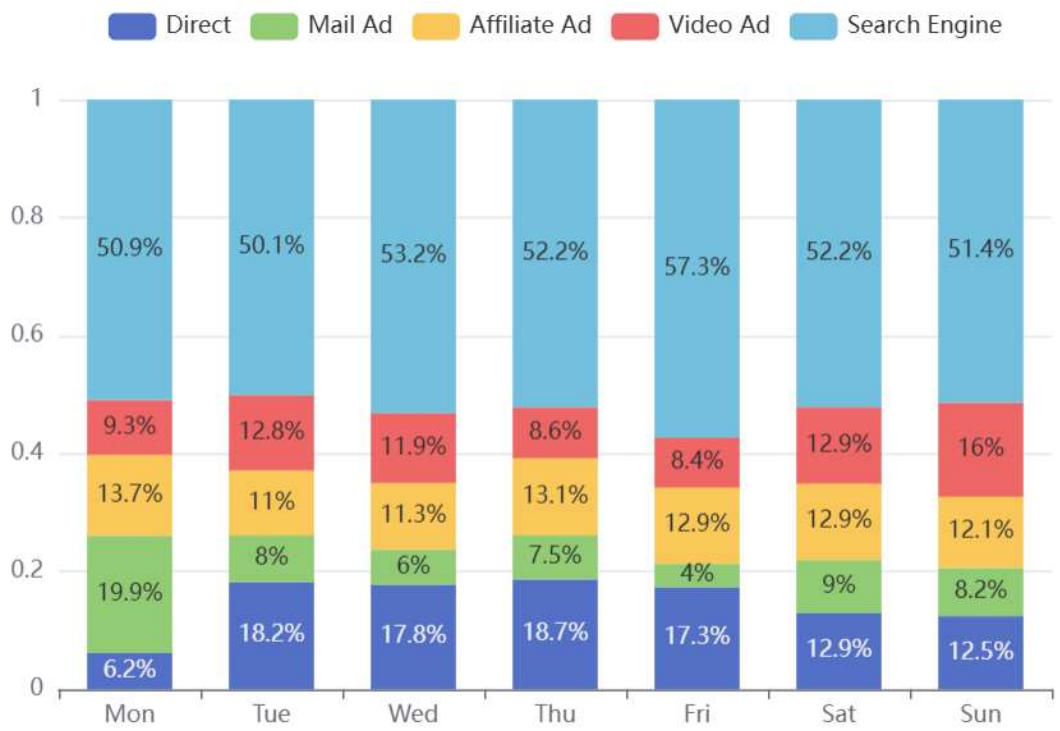
• **MM**: Number or type of **FIX** protocol messages or market data.

• **θj**: Weight of the j-th **FIX** protocol message or market data, representing the impact of that data on throughput.

• **Rj**: Transmission time of the j-th packet, representing the time required to transmit the packet.

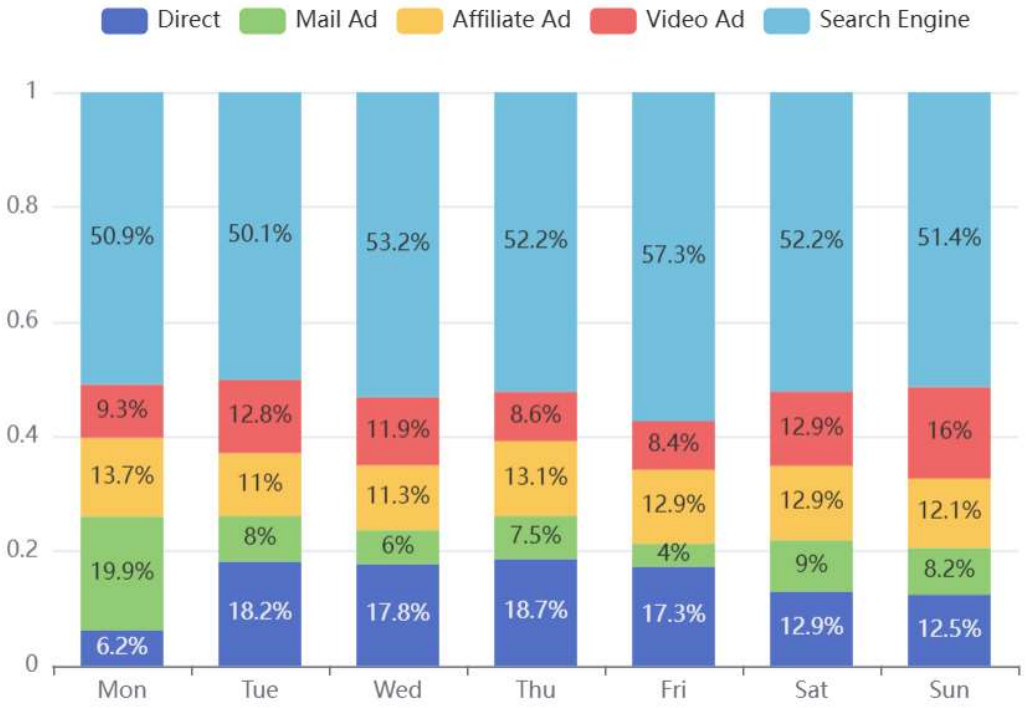
• **γj**: Delay factor in **FIX** protocol or market data, reflecting the impact of transmission delay or protocol parsing on throughput.

• **tj**: Network delay of the j-th packet, indicating the transmission delay of the data packet in the network.



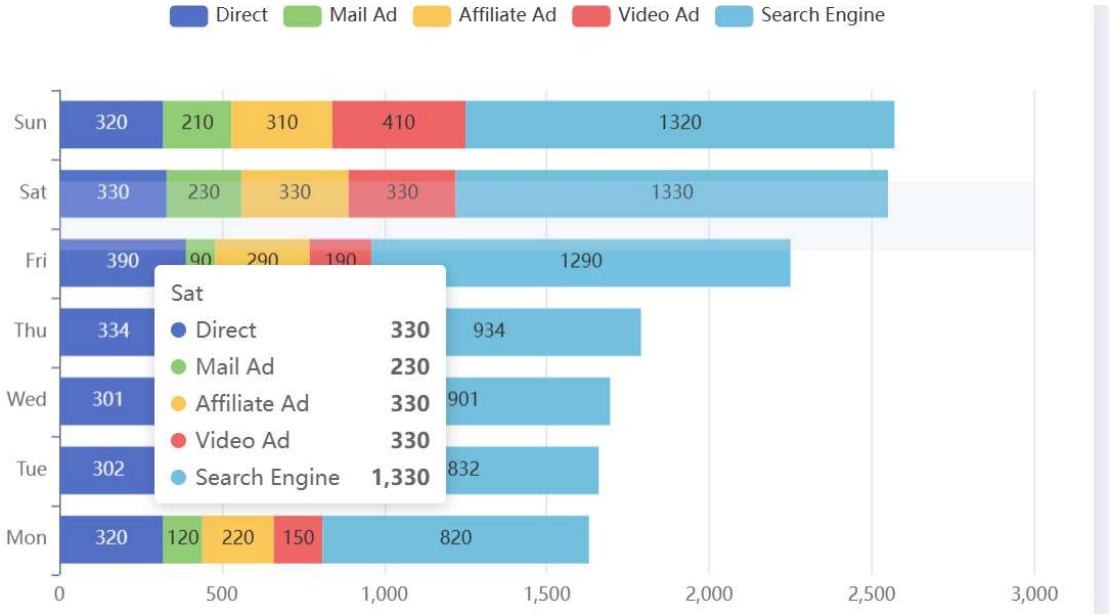
Formula Analysis: • **Multi-Data Source Throughput Calculation**:

By calculating the weighted average, combining each data source's transmission time **Tdata(i)** and latency time **Tlatency(i)**, the system computes the total throughput in a multi-data source environment. Lower latency **Tlatency(i)** increases throughput, while higher latency is adjusted through a decay factor to ensure the system can stably process large amounts of data.

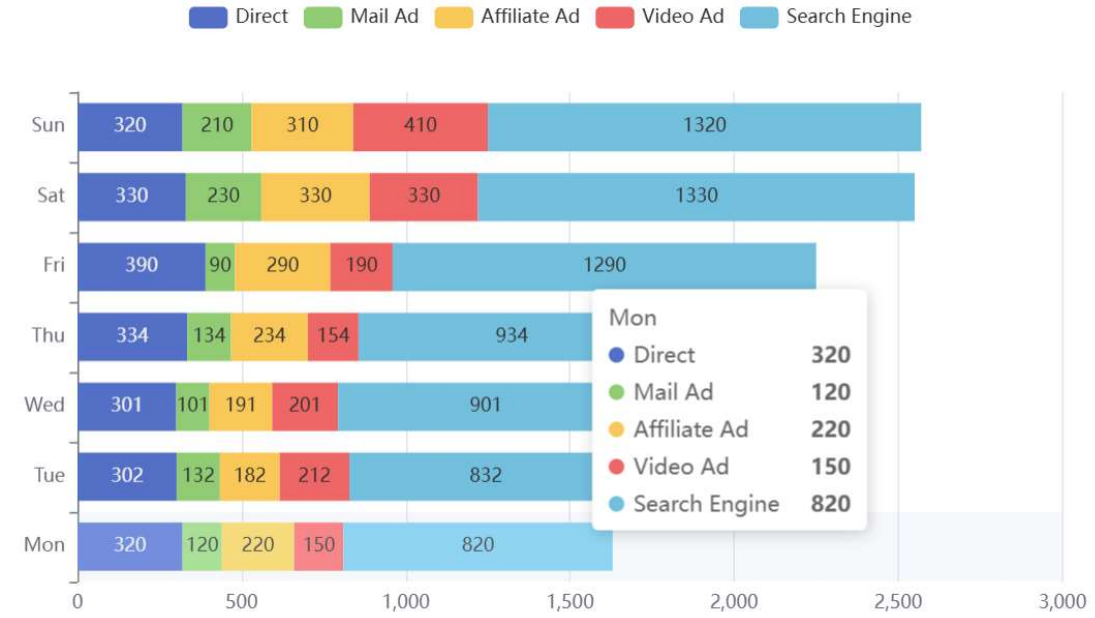


• **Latency and Throughput Optimization**: By controlling latency fluctuations, the system can maintain high throughput while avoiding trading losses caused by data transmission delays.

The latency standard deviation **σlatency²** controls the amplitude of latency fluctuations, reducing throughput fluctuations due to network issues or data source problems.



• **FIX Protocol and Market Data Processing**: By processing the transmission time **Rj** and network delay **tj** of **FIX** protocol messages and market data, the system ensures efficient processing of each data packet and market message. **FIX** protocol optimization ensures the reliability of data transmission, while the delay factor **γj** allows the platform to control the transmission efficiency of the protocol and data flow.



• **Balancing Throughput and Reliability**: Throughput optimization ensures the platform can handle a large volume of market data, while latency control optimization guarantees that trading strategies can promptly respond to market changes, preventing erroneous trades or losses caused by delays.

2. Real-time Data Cleaning and Filtering

Noise Data Removal

After receiving raw market data, A-Force immediately applies automated data cleaning algorithms to remove invalid and noisy data (such as erroneous trades, network disconnections, etc.). These cleaning operations help ensure the purity and accuracy of the platform's data. To describe how the A-Force platform uses **automated data cleaning algorithms** to remove invalid and noisy data, the following formula incorporates key factors such as **noise identification**, **data cleaning**, and **quality control**. With these algorithms, the system effectively removes abnormal data such as erroneous trades and network interruptions, ensuring the purity and accuracy of the platform's data.

$$D_{\text{clean}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(D_{\text{raw}}^{(i)} \cdot \exp\left(- \frac{(T_{\text{noise}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{noise}}^2} \right) \right)}{\sum_{i=1}^{N} W_i}$$

Formula Explanation:

• **Dclean**: The cleaned dataset, representing the purified dataset after noise data removal.

• **NN**: The number of data sources (e.g., multiple exchanges, market data sources, etc.).

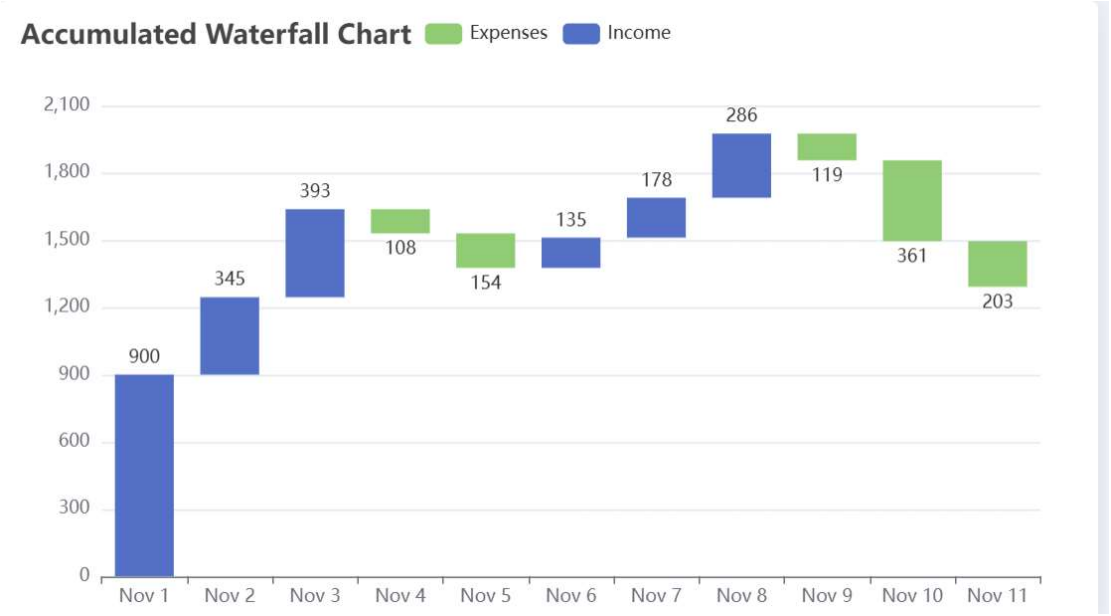
• **Wi**: The weight of the i-th data source, representing the importance of that data source in the data cleaning process. This weight can be dynamically adjusted based on the reliability and data quality of the source.

• **Draw(i)**: The raw data from the i-th data source, representing the unprocessed dataset.

• **Tnoise(i)**: The amount of noise data from the i-th data source, representing the proportion or frequency of noise data detected from that source.

• **Tthreshold**: The threshold for noise data, representing the system-defined noise level. When noise exceeds this threshold, the data is considered invalid and is removed.

• **σnoise²**: The standard deviation of noise, controlling the volatility of noise data, reflecting the extent of noise data's impact in the system.



Formula Analysis:

• **Noise Data Identification and Removal**: The raw data **Draw(i)** from each data source is filtered based on the amount of noise data **Tnoise(i)** and the noise threshold **Tthreshold**. When the amount of noise exceeds the set threshold, the related data is considered invalid and is either discarded or replaced.

• **Noise Fluctuation Control**: The impact of noise data is adjusted using a Gaussian decay factor $\exp(-((T_{noise}(i)-T_{threshold})^2/\sigma_{noise}^2))$. When noise data is far from the threshold, it is more strictly removed, ensuring that only data meeting the standards enters the processing pipeline.

• **Data Quality Control**: Through the weight **Wi** of each data source and noise control, the system can dynamically adjust the sensitivity of noise filtering based on the importance of the data source. This ensures that data from higher-weight sources are more likely to retain high-quality data.

Outlier Detection

Through machine learning models and rule engines, A-Force can automatically identify and filter out abnormal fluctuations or data errors, such as extreme price movements or false data reports. This prevents the strategy from being affected by unrealistic data. To describe how the A-Force platform uses **machine learning models** and **rule engines** to automatically detect and filter abnormal fluctuations or data errors, the following formula incorporates factors such as **outlier detection**, **data cleaning**, **rule engine optimization**, and **model training**. Through these methods, the system effectively identifies extreme price fluctuations or false data and prevents these unrealistic data from impacting trading strategies.

$$D_{\text{filtered}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(D_{\text{raw}}^{(i)} \cdot \left(1 - \alpha_i \cdot \exp\left(- \frac{(T_{\text{outlier}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{outlier}}^2} \right) \right) \right)}{\sum_{i=1}^{N} W_i}$$

Formula Explanation:

• **Dfiltered**: The dataset after outlier detection and filtering, representing valid and cleaned market data.

• **NN**: The number of data sources (e.g., real-time data sources from multiple exchanges or markets).

• **Wi**: The weight of the i-th data source, representing the importance of that data source in the data cleaning process. This weight can be dynamically adjusted based on factors such as data source reliability, trading volume, and data quality.

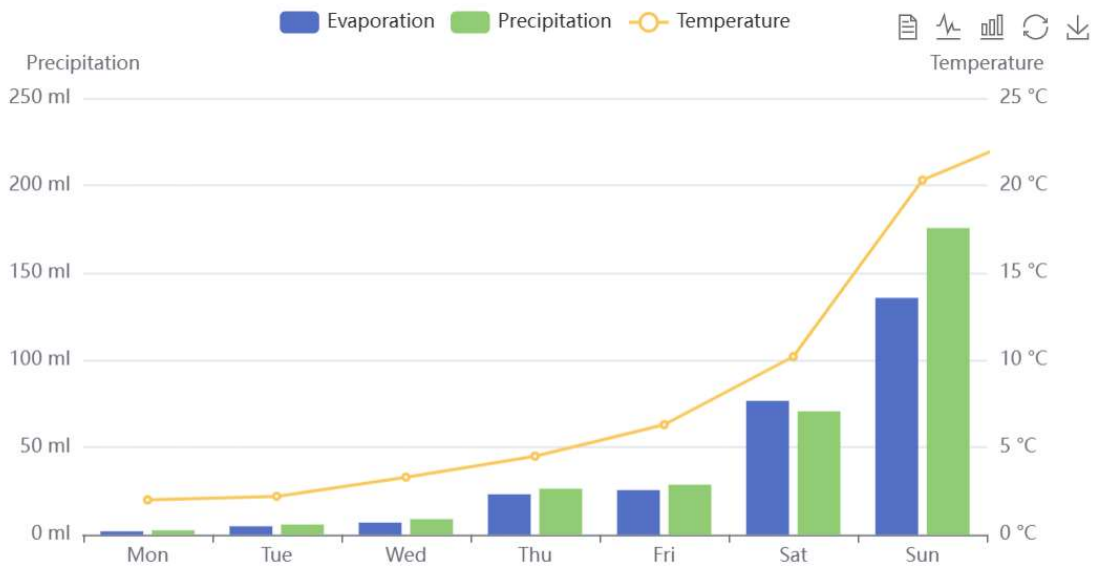
• **Draw(i)**: The raw data from the i-th data source, representing the unprocessed dataset.

• **Toutlier(i)**: The outliers from the i-th data source, representing extreme fluctuations or erroneous data in that source.

• **Tthreshold**: The threshold for outliers, representing the system-defined boundary for abnormal data. When data fluctuations exceed this threshold, the data is considered an outlier and is filtered out.

• **αi**: Data cleaning intensity factor, representing the sensitivity of outlier data filtering. This factor adjusts the precision of outlier detection.

• **σoutlier²**: Standard deviation of outliers, representing the magnitude of abnormal fluctuations, controlling the volatility impact of outliers.



Formula Analysis:

• **Outlier Detection and Filtering**: The system calculates the raw data **Draw(i)** from each data source and the outliers **Toutlier(i)**, then compares them with the defined threshold **Tthreshold**. Data values exceeding the threshold are considered outliers and are either discarded or corrected, thus improving the accuracy and reliability of the data.

• **Abnormal Fluctuation Impact Control**: Using a Gaussian decay factor **$\exp(-((T_{outlier}(i)-T_{threshold})^2/\sigma_{outlier}^2))$**, the system dynamically adjusts the weight of outliers based on how far they deviate from the threshold. When data is further from the threshold, the system will filter out these abnormal fluctuations more strictly.

• **Sensitivity Adjustment**: The cleaning intensity factor **α_i** controls the sensitivity of outlier detection for each data source. For market data with large fluctuations, the system increases the sensitivity of outlier detection to prevent extreme fluctuations from affecting the trading strategy.

Data Quality Assurance

The cleaning algorithm assigns a quality score to each piece of data based on data quality indicators such as completeness, timeliness, and accuracy, then selects the most reliable data for further analysis. To describe how the A-Force platform ensures data quality through **data cleaning algorithms**, the following formula incorporates key factors such as **data integrity**, **timeliness**, **data scoring**, and **selection**. By scoring the quality of each piece of data, the system is able to select the most reliable data for further analysis, ensuring high-quality data is input into the strategy engine.

$$D_{\text{validated}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(D_{\text{raw}}^{(i)} \cdot \left(1 + \alpha_i \cdot \exp\left(- \frac{(T_{\text{quality}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{quality}}^2} \right) \right) \right)}{\sum_{i=1}^{N} W_i}$$

Formula Explanation:

• **Dvalidated**: The final dataset after data quality assurance, representing the valid data selected after quality scoring and filtering.

• **NN**: The number of data sources, representing multiple exchanges, market data sources, or historical data sources.

• **Wi**: The weight of the i-th data source, representing the importance of that data source in data quality assurance.

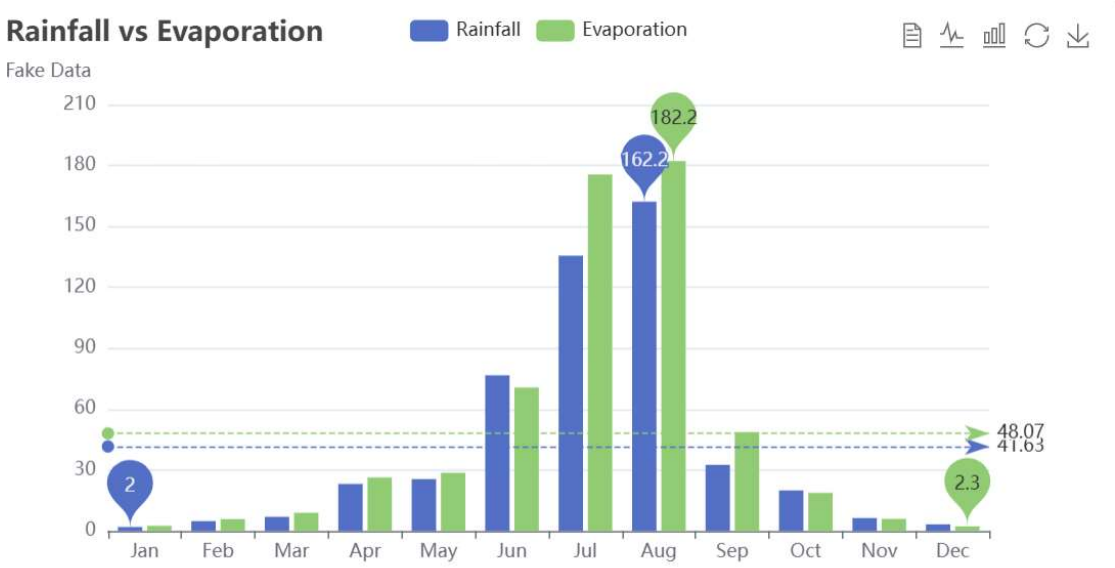
• **Draw(i)**: The raw data from the i-th data source, representing uncleaned raw market data.

• **Tquality(i)**: The quality score of the i-th data source, representing the data quality of that source, including integrity, timeliness, accuracy, etc.

• **Tthreshold**: The threshold for data quality scores, representing the system-defined minimum acceptable data quality score. Data below this threshold will be excluded.

• **αi**: Data quality cleaning factor, controlling the intensity of adjustments to the data quality score. This factor is dynamically adjusted based on the data source's quality characteristics.

• **σquality²**: Standard deviation of data quality scores, representing the volatility of data quality, ensuring that variations in scores from different data sources influence the final dataset.



Formula Analysis:

• **Data Quality Scoring and Selection**: The system evaluates the validity of each data source based on its quality score **Tquality(i)**. The score is calculated using multiple dimensions, such as data completeness, timeliness, and accuracy. If the data quality score is above the set threshold **Tthreshold**, the data is considered valid and retained; otherwise, it is excluded.

• **Gaussian Decay Factor**: Using a Gaussian decay factor **$\exp(-((Tquality(i)-Tthreshold)^2/\sigma quality^2))$**, the system adjusts sensitivity based on how far the data quality score deviates from the threshold. When the score is far from the threshold, the system more strictly removes substandard data, ensuring that the final selected data has high quality.

• **Quality Adjustment Factor**: The data quality cleaning factor **αi** is used to adjust the sensitivity of data quality assurance. When the data source quality is high, the cleaning factor decreases, allowing more high-quality data to be retained; when the data quality is low, the cleaning factor increases, further optimizing the data selection process.

3. In-memory Computing and Distributed Processing

Memory-First Architecture:

A-Force's data processing engine uses a memory-first architecture, where real-time data is loaded into memory for processing, avoiding the I/O latency associated with traditional hard disk storage. This greatly increases the speed of data processing, allowing the system to respond to market changes in milliseconds. To describe how the A-Force platform enhances data processing speed through the **memory-first architecture**, the following formula shows how loading real-time data into memory for processing avoids traditional hard disk I/O delays and improves the overall system's response speed.

$$T_{\text{mem}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{data}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{I/O}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{I/O}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation:

• **Tmem**: Memory processing time, representing the time taken by the system to process data through the memory-first architecture (unit: milliseconds).

• **NN**: The number of data sources (e.g., real-time data sources from multiple exchanges or markets).

• **Wi**: The weight of the i-th data source, representing the importance of that data source in memory processing.

• **Tdata(i)**: Real-time data processing time of the i-th data source, representing the time from loading the data into memory to processing it.

• **TI/O(i)**: I/O latency of the i-th data source, representing the data read time from traditional hard disk storage.

• **Tthreshold**: I/O latency threshold, representing the system's optimized target latency. When I/O latency exceeds this threshold, the system will reduce the use of hard disk storage and prioritize memory for data processing.

• **σI/O²**: Standard deviation of I/O latency, representing the variability in I/O operations, ensuring that latency fluctuations do not impact data processing efficiency.

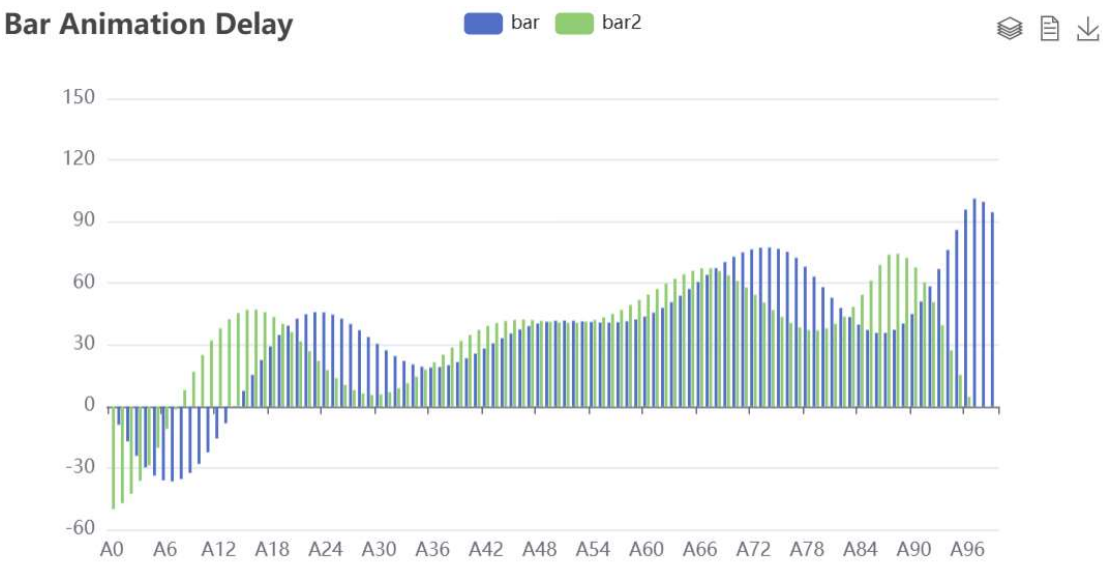
• **MM**: The number of asynchronous data processing tasks, representing the number of tasks to be processed in parallel (e.g., data preprocessing, trade signal generation, etc.).

• **θj**: The weight of the j-th asynchronous task, representing the impact of that task on the memory-first architecture.

• **Rj**: The processing time of the j-th task, representing the time required for the task to execute after the data is loaded into memory.

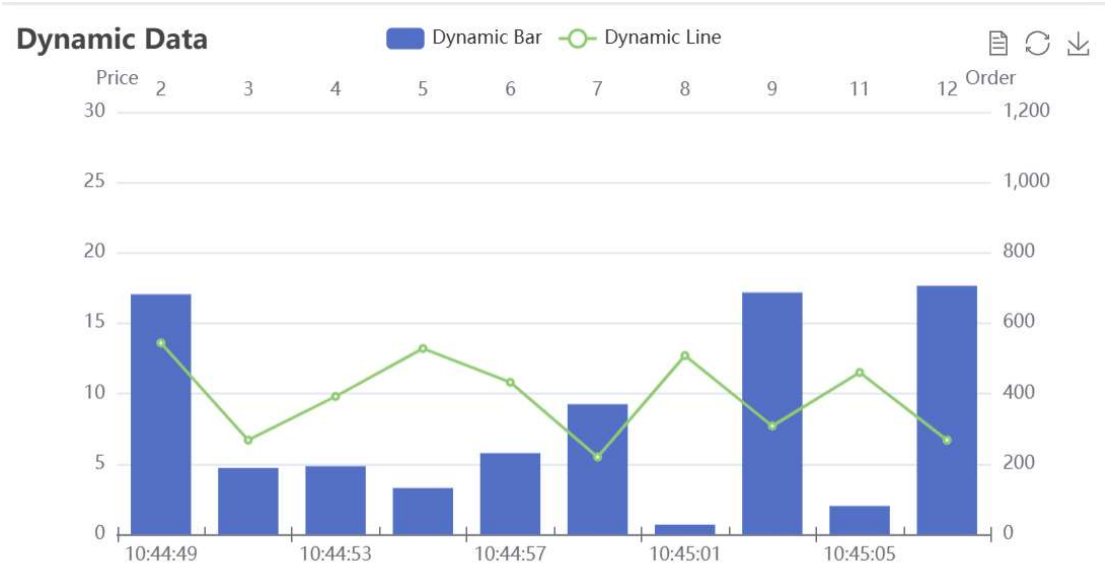
• **γj**: Delay factor of the asynchronous task, representing the increased processing time due to task delays.

• **tj**: The processing delay of the j-th task, representing the time it takes for the data to execute after being read from memory.

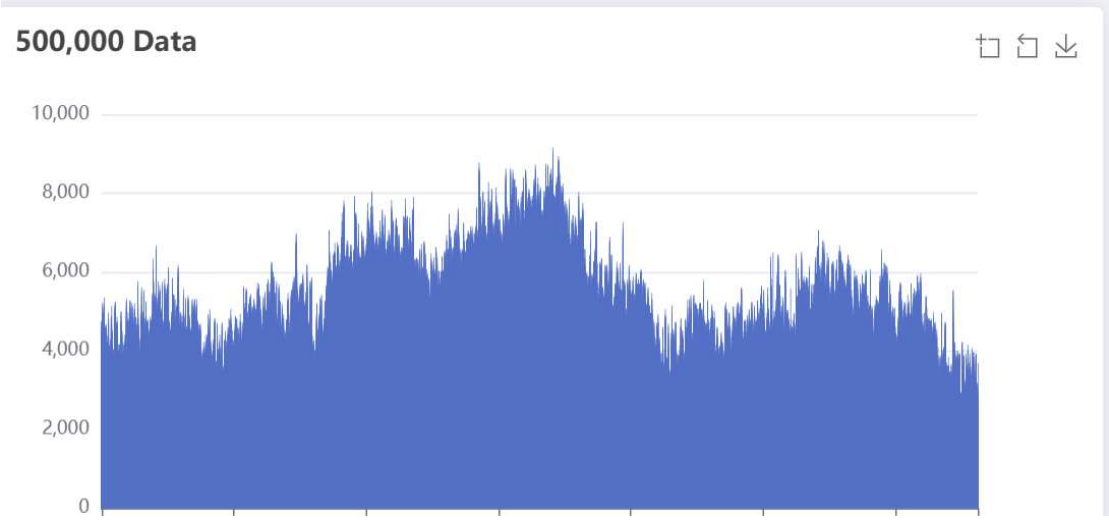


Formula Analysis:

• **Memory Data Processing Optimization**: The system processes data at high speed by loading real-time data into memory **Tdata(i)** and avoiding I/O latency **TI/O(i)**. The introduction of a memory-first architecture prevents I/O blocking from hard disk storage, significantly improving data processing speed.



• **I/O Latency Control and Optimization**: Using a Gaussian decay factor $\exp(-((TI/O(i)-Tthreshold)^2/\sigma I/O^2))$, the system dynamically adjusts the impact of hard disk I/O operations, ensuring that the memory-first architecture works effectively. Under high load conditions, the system automatically switches to memory mode for data processing, avoiding performance degradation caused by I/O latency.



• **Asynchronous Tasks and Parallel Processing**: The memory-first architecture not only improves the processing speed of individual tasks by reducing I/O latency but also enhances overall throughput by enabling parallel processing of multiple tasks (such as data preprocessing, trade signal generation, etc.). The processing time **Rj** and delay factor **γj** adjust the priority and resource allocation of tasks, enabling multiple tasks to be processed concurrently in memory.

Distributed Computing Framework

A-Force uses distributed computing frameworks like Apache Kafka and Apache Flink to handle large-scale data streams. Kafka is used for high-throughput data stream transmission, while Flink processes real-time data analytics and complex event stream processing. Through distributed processing, the platform is able to handle a large amount of concurrent trading data globally, and performs parallel computing across multiple nodes, improving the system's fault tolerance and scalability. To describe how the A-Force platform uses **distributed computing frameworks** (such as Apache Kafka and Apache Flink) to handle large-scale data streams, the following formula demonstrates how these technologies are used to perform parallel computing across multiple nodes, enhancing system throughput, fault tolerance, and scalability. This formula incorporates factors like **data flow transmission**, **real-time data processing**, **complex event stream processing**, and **system fault tolerance**.

$$T_{\text{distributed}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{flow}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{latency}}^{(i)} - T_{\min})^2}{\sigma_{\text{latency}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} (\theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j))) + \sum_{k=1}^{K} \left(\delta_k \cdot \left(C_k \cdot \exp\left(-\frac{(T_{\text{failure}}^{(k)} - T_{\text{threshold}})^2}{\sigma_{\text{failure}}^2} \right) \right) \right)$$

Formula Explanation:

• **Tdistributed**: The total processing time after distributed computing, representing the time taken by the platform to process the data stream in a distributed computing framework.

• **NN**: The number of data sources, representing the input from multiple exchanges, markets, or other data streams.

• **Wi**: The weight of the i-th data source, representing the contribution of that data source in the entire distributed computing process.

• **Tflow(i)**: The data flow transmission time of the i-th data source, representing the time taken for data to be transmitted from the source to the processing node.

• **Tlatency(i)**: The transmission latency of the i-th data source, representing the delay between the input and output of the data flow.

• **Tmin**: The system's minimum latency, representing the lowest latency under ideal conditions.

• **σlatency²**: Standard deviation of latency, controlling latency fluctuation to ensure efficient processing of the data stream.

• **MM**: The number of complex event stream processing tasks, representing the complex events that need to be processed by Apache Flink.

• **θj**: The weight of the j-th event stream processing task, representing the priority and resource usage of the event stream processing.

• **Rj**: The execution time of the j-th event stream processing task, representing the time required to process that event stream.

• **γj**: The delay factor of abnormal events, representing the impact of event processing delays on system performance.

• **tj**: The processing delay of the j-th event stream, representing the delay during complex event processing.

• **KK**: The number of computing nodes, representing the number of nodes participating in the distributed computation.

• **δk**: The fault tolerance factor of the k-th computing node, representing the importance of

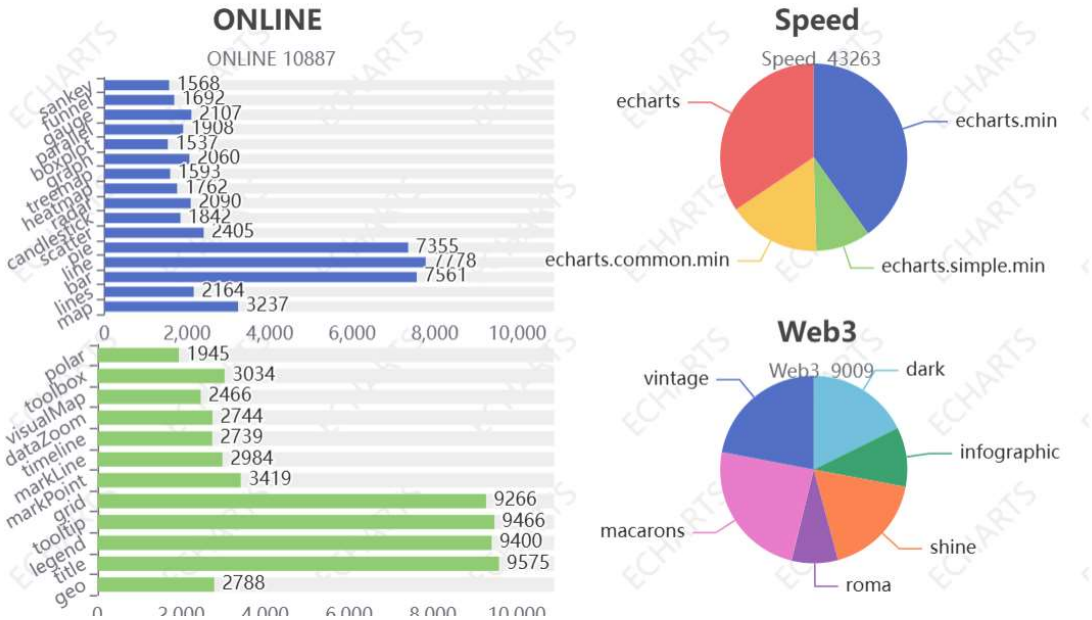
that node in the fault tolerance mechanism.

• **Ck**: The computing power of the k-th node, representing the resource contribution of that node in the distributed computation.

• **Tfailure(k)**: The failure time of the k-th node, representing the time taken for that node to fail.

• **Tthreshold**: The failure threshold, representing the maximum tolerable latency defined by the system. When the node's latency exceeds this threshold, the system will automatically switch to a backup node.

• **σfailure²**: Standard deviation of failure latency, controlling the impact range of node failures, ensuring the system can quickly recover.



Formula Analysis:

• **Data Flow Transmission and Latency Optimization**: The system dynamically optimizes the flow transmission time **Tflow(i)** and latency **Tlatency(i)** for each data source. Using the Gaussian decay factor **exp(−((Tlatency(i)−Tmin)²/σlatency²))**, the system adjusts according to real-time transmission delays and fluctuations to ensure efficient data transfer across the distributed network with low latency.

• **Complex Event Stream Processing**: Complex event stream processing via Apache Flink is scheduled based on task priorities **θj** and execution times **Rj**, ensuring that high-priority event streams are processed quickly. The processing delays are controlled by the factors **γj** and **tj**, guaranteeing efficient responsiveness.

• **Fault Tolerance Mechanism and Node Switching**: When a computing node fails, the system automatically switches to a backup node through the fault tolerance mechanism. The failure time **Tfailure(k)** and the fault tolerance factor **δk** ensure that node failures are quickly identified and addressed. By adjusting the fault tolerance factor **δk** and the computing power **Ck**, the system can efficiently allocate node resources, ensuring efficient distributed computation.

Horizontal Scalability

This architecture supports horizontal scaling, allowing the system to add more computing nodes based on the load to handle an increase in market data volume or a surge in transaction volume. To describe how the A-Force platform utilizes **horizontal scalability** to handle a surge in market data or transaction volume, the following formula combines factors like **load balancing**, **node expansion**, **resource allocation**, and **system scalability**. Through horizontal scaling, the platform can dynamically add computing nodes based on demand, ensuring the system operates efficiently and can handle increased workloads.

$$T_{\text{scaled}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{task}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{load}}^{(i)} - T_{\text{opt}})^2}{\sigma_{\text{load}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j)) \cdot \left(1 + \frac{L_j}{N_{\text{nodes}}} \right) \right)$$

Formula Explanation:

• **Tscaled**: Scaled task processing time, representing the total time taken by the system to process tasks after horizontal scaling.

• **NN**: The number of computing nodes in the current system, representing the number of nodes used by the platform.

• **Wi**: The weight of the i-th task, representing the impact of that task on the total processing time.

• **Ttask(i)**: The execution time of the i-th task, representing the time required to process that task.

• **Tload(i)**: The current load of the i-th node, representing the amount of computational resources consumed by that node.

• **Topt**: The optimal load time for the system, representing the ideal state when the load is minimized.

• **σload²**: Standard deviation of the load, representing the fluctuation in load, ensuring load balancing in the system.

• **MM**: The number of tasks that need to be processed by distributed computing or asynchronous processing.

• **θj**: The weight of the j-th task, representing the contribution of that task to the processing system.

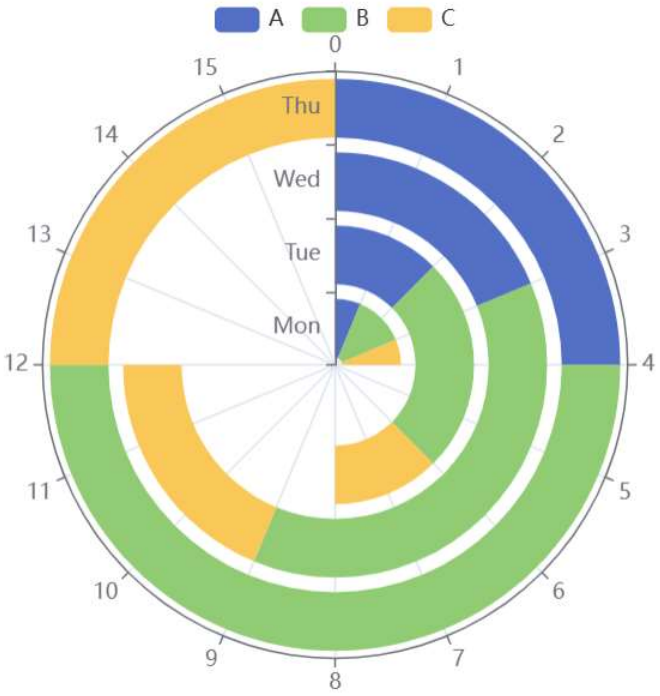
• **Rj**: The processing time of the j-th task, representing the time required from the start of execution to completion of that task.

• **γj**: Delay factor for abnormal events, representing the impact of task delays on system performance.

• **tj**: The processing delay of the j-th task, representing the delay in the data flow or event flow during processing.

• **Lj**: The load of the j-th task, representing the system's resource demand for that task.

• **Nnodes**: The number of nodes currently used by the system, representing the available computing nodes in the platform.

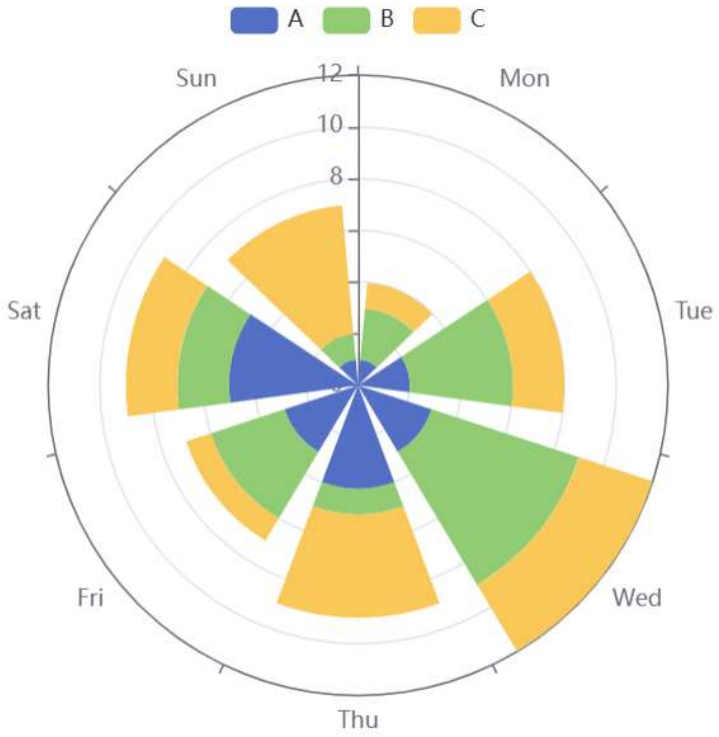


Formula Analysis:

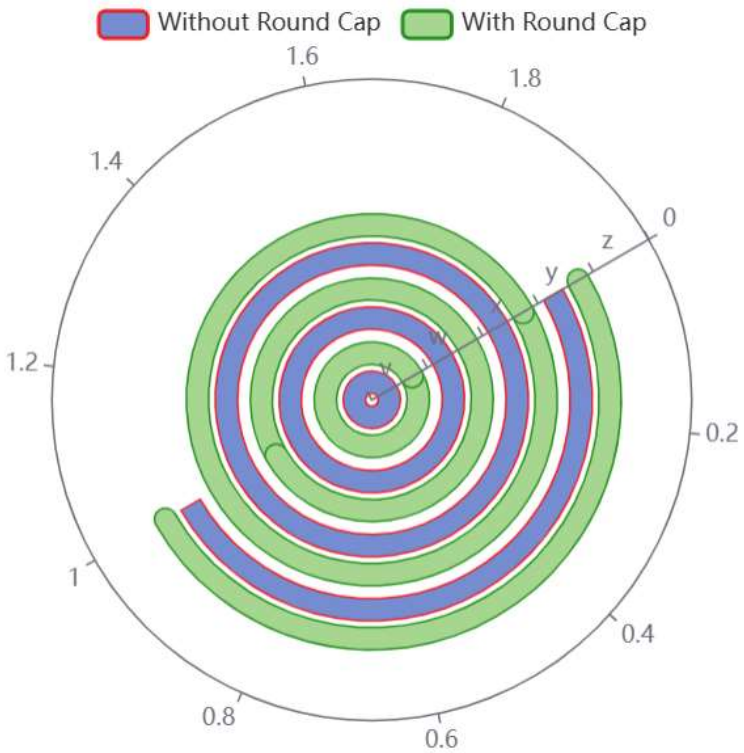
• **Horizontal Scaling and Load Balancing**: Through horizontal scaling, the system can dynamically add more computing nodes **Nnodes**, allocating more resources when the load increases. The increase in the number of nodes **Nnodes** impacts the resource allocation for each task, reducing the load **Tload(i)** on individual nodes and optimizing task processing time through load balancing.

• **Task Parallel Processing and Throughput Optimization**: As the number of computing nodes increases, tasks **Ttask(i)** are distributed across multiple nodes for parallel processing, reducing the overall task processing time. The standard deviation of the load σ_{load}^2 ensures smooth operation of the system during load changes, preventing bottlenecks due to excessive node pressure.

• **Impact of Node Expansion on Latency**: As more computing nodes are added, the system's computational capacity increases, reducing the latency impact of individual nodes. The term



Lj / Nnodes represents the dynamic adjustment of task load based on the number of nodes, ensuring that adding nodes effectively increases the system's processing capacity and throughput.



• **System Adaptability**: The system can dynamically scale horizontally based on changes in

market demand. When market data or transaction volume surges, the platform can add more computing nodes in real-time, maintaining efficient data processing speed and stable performance.

4. Dynamic Event Detection and Triggering

Event-Driven Mechanism

A-Force's internal event detection system can monitor market data streams in real time and generate corresponding trade instructions when market anomalies such as price fluctuations, market abnormalities, or breaking news events occur. For instance, when the price of a cryptocurrency suddenly rises above a preset threshold, the system automatically triggers a buy or sell order. To describe how the A-Force platform uses an **event-driven mechanism** to monitor market data streams and generate trade instructions in real-time, the following formula combines factors like **event detection**, **threshold setting**, **trade instruction generation**, and **market volatility response**. Through these mechanisms, the system can react immediately and execute corresponding trade instructions when price fluctuations, market anomalies, or breaking news events are detected.

$$T_{\text{event-driven}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{event}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{volatility}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{volatility}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation:

• **Tevent-driven**: The total trade response time based on the event-driven mechanism, representing the time from when the market event occurs to when the trade instruction is generated and executed.

• **NN**: The number of market data sources involved in event detection (e.g., different cryptocurrency trading pairs, market data sources, etc.).

• **Wi**: The weight of the i-th market data source, representing the importance of that source in event triggering and trade decision-making.

• **Tevent(i)**: The response time for the i-th event, representing the time from the market event (e.g., price fluctuation, breaking news) to the triggering of a trade instruction.

• **Tvolatility(i)**: The volatility of the i-th data source, representing the intensity of market price fluctuations, typically measured by the magnitude of price movement.

• **Tthreshold**: The event trigger threshold, representing the system-defined threshold for market fluctuations. When market fluctuations exceed this threshold, the system automatically generates a trade instruction. For example, when the price fluctuation surpasses a predefined percentage, a buy or sell action is triggered.

• **σvolatility²**: Standard deviation of volatility, representing the variation in market volatility, ensuring that market events with significant volatility are effectively handled.

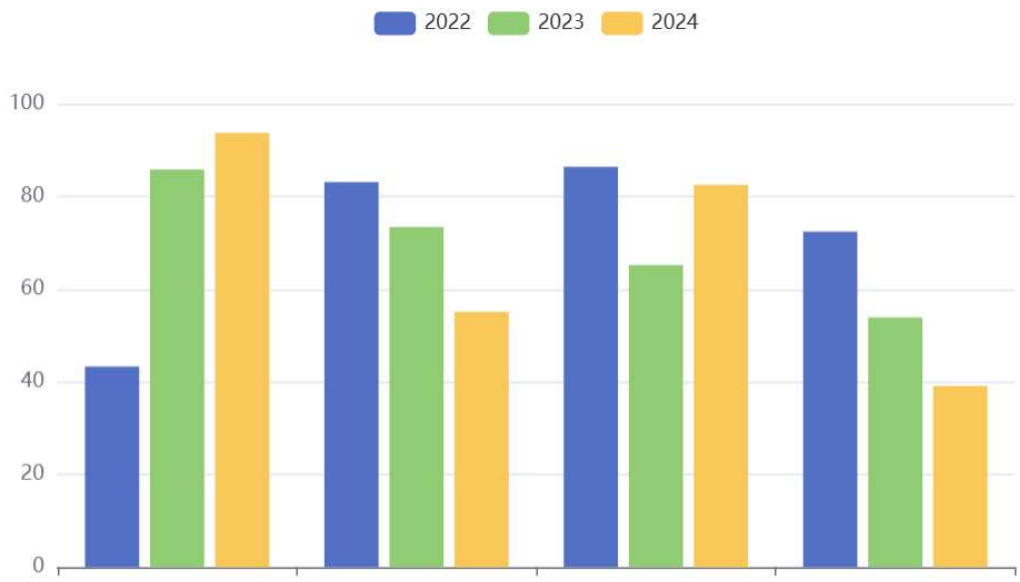
• **MM**: The number of trade instructions to be executed through the event-driven mechanism.

• **θj**: The weight of the j-th trade instruction, representing the priority of the instruction or the resource allocation for executing the trade.

• **Rj**: The execution time of the j-th trade instruction, representing the time from the generation of the instruction to the completion of the trade.

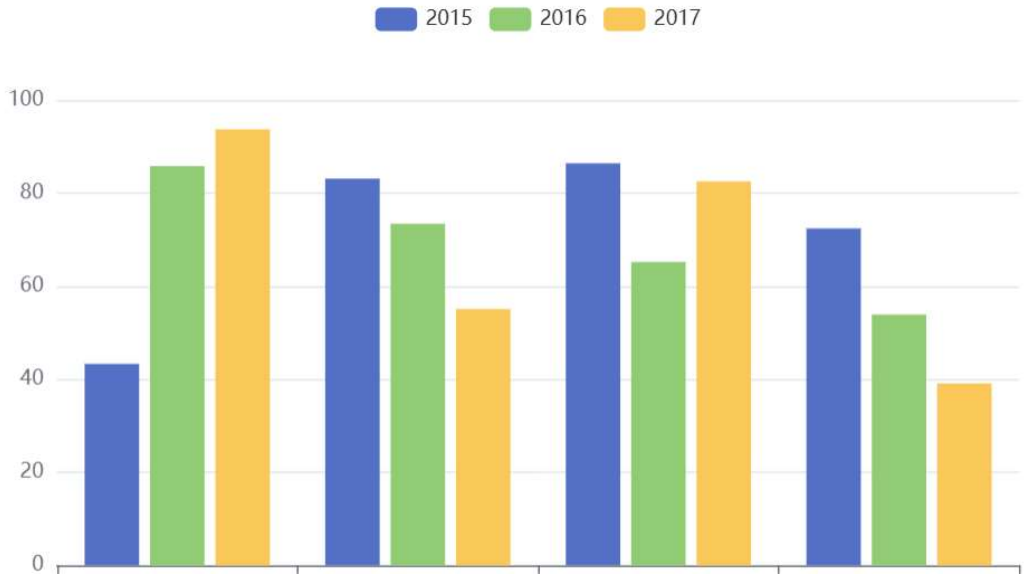
• **γj**: The trade delay factor, representing the impact of system processing delays or network delays on the trade response time.

• **tj**: The processing delay of the j-th trade instruction, representing the delay during the trade execution process.



Formula Analysis:

• **Event Detection and Response Time**: The system detects events in real-time based on price fluctuations and sudden events in the market data stream. For example, when the price of a cryptocurrency exceeds a set threshold, the system immediately responds and generates a trade instruction. The response time **Tevent(i)** is the time from the event occurrence to the generation of the trade instruction, influenced by volatility **Tvolatility(i)** and the threshold **Tthreshold**.



• **Volatility and Threshold Control**: Using a Gaussian decay factor **$\exp(-((T_{volatility}(i)-T_{threshold})^2/\sigma_{volatility}^2))$**, the system adjusts the threshold based on the actual market volatility, ensuring that when market fluctuations exceed the set threshold,

trade instructions are triggered promptly.

• **Trade Instruction Generation and Execution**: When a market event is triggered, the platform immediately generates a trade instruction **Rj**, and the execution time is calculated based on the trade delay factor **γj** and the processing delay **tj**. The system optimizes the trade execution process to ensure that instructions are executed in the shortest possible time, minimizing potential risks caused by delays.

Intelligent Decision Engine

Combining machine learning and data mining techniques, A-Force's event-driven system can identify market trends and potential investment opportunities, optimizing trading strategies. To describe how the A-Force platform uses the **intelligent decision engine** to optimize trading strategies, the following formula integrates key factors such as **machine learning**, **data mining**, **market trend identification**, and **investment opportunity forecasting**. With these techniques, the system can recognize market trends and automatically adjust strategies to maximize trading effectiveness and profits.

$$S_{\text{decision}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{trend}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{signal}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{signal}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation:

• **Sdecision**: The final trading decision output, representing the optimized trading strategy based on market trends and investment opportunity forecasts.

• **NN**: The number of data sources involved in the decision-making process (e.g., data from different markets, exchanges, technical indicators, etc.).

• **Wi**: The weight of the i-th data source, representing the importance of that source in optimizing the trading strategy.

• **Ttrend(i)**: The market trend signal of the i-th data source, representing the trend strength derived from market data and technical analysis.

• **Tsignal(i)**: The signal strength of the i-th data source, representing the trend or trading signal extracted from that source.

• **Tthreshold**: The signal trigger threshold, representing the level at which market signals are used to generate trading strategies. For instance, when market signals exceed this threshold, the system generates a trade strategy based on the market trend.

• **σsignal²**: Standard deviation of signal strength, representing the fluctuation in market signals, ensuring that signals are used for decision-making under stable conditions.

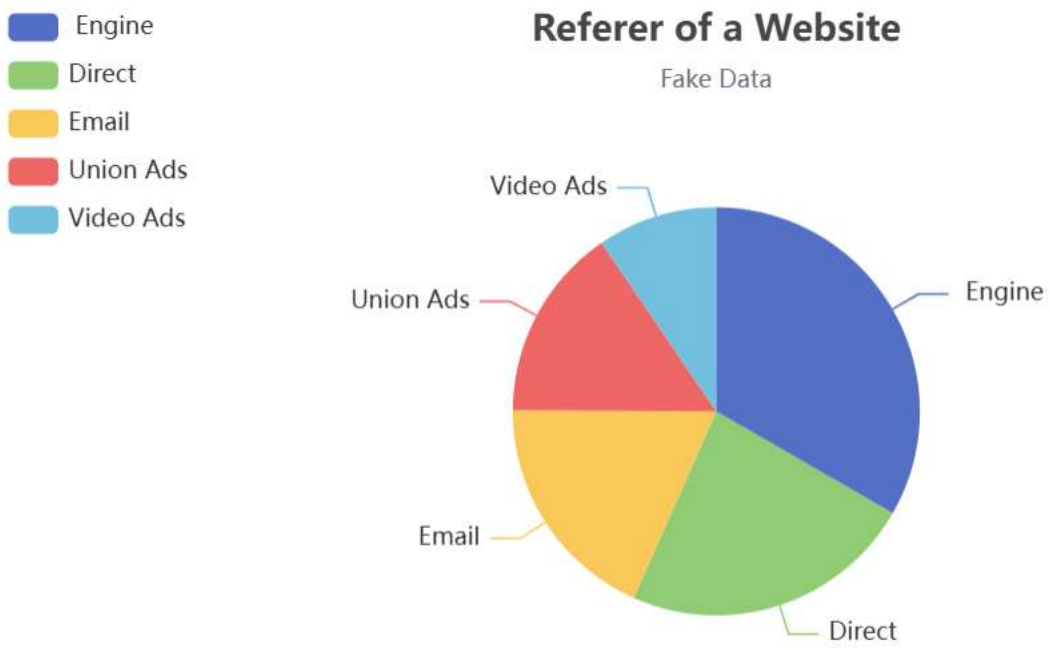
• **MM**: The number of investment opportunity identification tasks based on machine learning and data mining techniques.

• **θj**: The weight of the j-th investment opportunity, representing the influence of that opportunity on trading decisions.

• **Rj**: The expected return of the j-th investment opportunity, representing the potential return from the predicted investment opportunity.

• **γj**: The delay factor for the investment opportunity, representing the delay caused by market changes or data updates.

• **tj**: The response time for the j-th investment opportunity, representing the time taken from recognizing a market trend to discovering an investment opportunity.



Formula Analysis:

• **Market Trends and Signal Strength**: The system first analyzes the market trend **Ttrend(i)**, then determines whether to generate a trading signal based on the signal strength **Tsignal(i)**. By comparing with the set threshold **Tthreshold**, when the signal strength exceeds the threshold, the system generates the corresponding trading strategy. This method allows the system to automatically capture market trends and generate effective decisions based on the signals.

• **Signal Fluctuation and Stability Control**: A Gaussian decay factor **$\exp(-((T_{signal}(i)-T_{threshold})^2/\sigma_{signal}^2))$** is used to optimize the signal's volatility, ensuring the stability and accuracy of market signals. Larger fluctuations are adjusted by the standard deviation **σ_{signal}^2**, ensuring that trading decisions are not disturbed by short-term volatility.

• **Investment Opportunity Identification and Optimization**: Machine learning and data mining techniques help the system analyze market trends and identify potential investment opportunities **Rj**. Each investment opportunity **Rj** is weighted based on its expected return and time delay. The system evaluates each opportunity's priority using **θj** and **γj**, ensuring the platform quickly responds to the most promising investment opportunities.

• **Trading Decision Optimization**: By dynamically adjusting the weight of each data source **Wi** and each investment opportunity **θj**, the system can automatically optimize trading decisions. Through machine learning models and data mining techniques, A-Force learns from historical data and optimizes trading strategies in real-time to adapt to the ever-changing market environment.

Real-time Reaction

Whenever a new market signal or abnormal fluctuation occurs, the system immediately generates and submits trading instructions, ensuring that the most timely trading opportunities are captured. To describe how the A-Force platform **immediately generates and submits trading instructions** when new signals or abnormal fluctuations are detected, the following formula incorporates factors such as **signal detection**, **volatility identification**, **instruction generation**, and **execution**. Through these mechanisms, the platform can respond in real-time to market changes and ensure that trading opportunities are captured in the shortest time possible.

$$T_{\text{reaction}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{signal}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{volatility}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{volatility}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} (\theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j)))$$

Formula Explanation:

• **Treaction**: The total real-time reaction time, representing the time taken from identifying the market signal or fluctuation to generating and submitting the trade instruction.

• **NN**: The number of signal sources, representing data obtained from multiple markets, exchanges, or signal sources.

• **Wi**: The weight of the i-th signal source, representing the importance of that signal in the system's decision-making process.

• **Tsignal(i)**: The detection time of the i-th signal, representing the time it takes from the signal occurrence to the system identifying the signal.

• **Tvolatility(i)**: The volatility of the i-th market signal, representing the intensity of the market fluctuation.

• **Tthreshold**: The volatility threshold, representing the level of fluctuation that triggers a trade instruction. When market fluctuations exceed this threshold, the system generates the corresponding trade instruction.

• **σvolatility²**: The standard deviation of volatility, controlling the fluctuations in market volatility, ensuring that high-volatility market events are quickly identified and responded to.

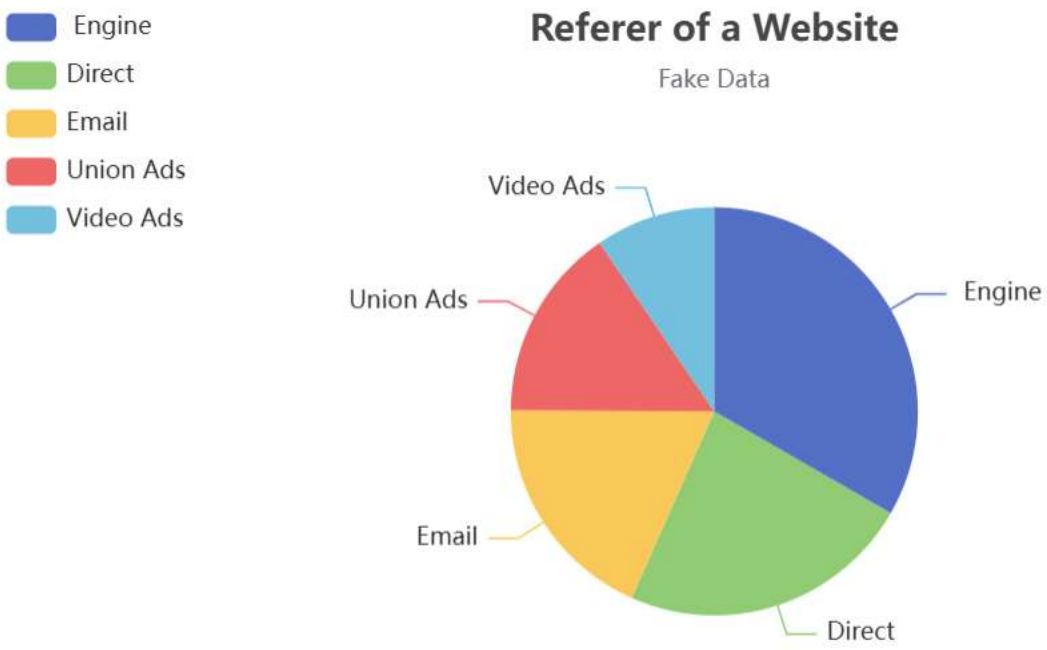
• **MM**: The number of trade instructions to be generated based on market signals and fluctuations.

• **θj**: The weight of the j-th trade instruction, representing the priority and influence of that instruction in the overall strategy.

• **Rj**: The execution time of the j-th trade instruction, representing the time required to generate and submit the trade instruction.

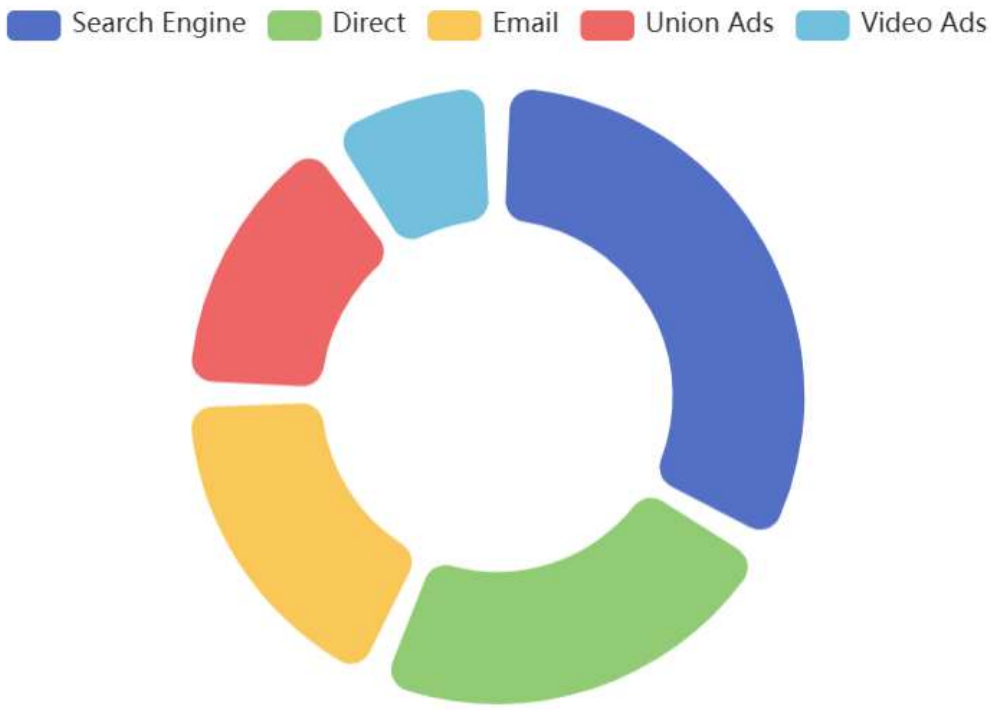
• **γj**: The delay factor, representing the impact of execution delay on the overall reaction time.

• **tj**: The processing delay of the j-th trade instruction, representing the time taken from generating the instruction to submitting it.



Formula Analysis:

• **Signal Detection and Reaction Time**: The system detects each signal source **Tsignal(i)** and signal volatility **Tvolatility(i)**, and compares them with the set volatility threshold **Tthreshold** to trigger the generation of a trade instruction. When market fluctuations exceed the set threshold, the system immediately generates the corresponding trade instruction to respond to market changes.



• **Volatility and Threshold Control**: Using a Gaussian decay factor **exp(−((Tvolatility(i)−Tthreshold) ² / σ volatility ²))**, the system dynamically adjusts the

sensitivity of the response based on market volatility. Signals with higher volatility are prioritized, ensuring the platform quickly captures potential trading opportunities.

• **Trade Instruction Generation and Execution**: Whenever a market signal or fluctuation is detected, the system generates a trade instruction **Rj** and executes it. The execution time **Rj** and delay factor **γj** ensure that the instruction is submitted to the exchange in the shortest possible time after detecting a favorable signal, avoiding potential trading losses due to delays.

5. Cache and Historical Backtracking

Real-time Data Caching

A-Force uses caching technologies such as Redis or Memcached to ensure fast access and processing of real-time data across multiple strategies or data analysis processes. The use of caching significantly enhances system response times, particularly in trading environments that require high-frequency updates and queries. To describe how the A-Force platform uses **caching technologies** (such as Redis or Memcached) to ensure fast access and processing of real-time data, the following formula incorporates key factors like **cache management**, **data storage**, **access optimization**, and **system response speed**. By using caching technology, the platform efficiently accesses real-time data across multiple strategies and data analysis processes, thus improving system response times, especially in high-frequency trading environments.

$$T_{\text{cache}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{data}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{access}}^{(i)} - T_{\text{opt}})^2}{\sigma_{\text{access}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation:

• **Tcache**: The optimized data access time through caching, representing the total time taken to process data using high-speed cache.

• **NN**: The number of data sources, representing real-time data obtained from multiple markets, exchanges, or data sources.

• **Wi**: The weight of the i-th data source, representing the importance of that data source in cache optimization.

• **Tdata(i)**: The data transmission time for the i-th data source, representing the time taken to transfer data from the source to the cache.

• **Taccess(i)**: The cache access time for the i-th data source, representing the time taken to read data from the cache.

• **Topt**: The optimal cache access time, representing the ideal data access time under optimal conditions.

• **σaccess²**: The standard deviation of cache access time, controlling the fluctuation in access times to ensure stable cache performance.

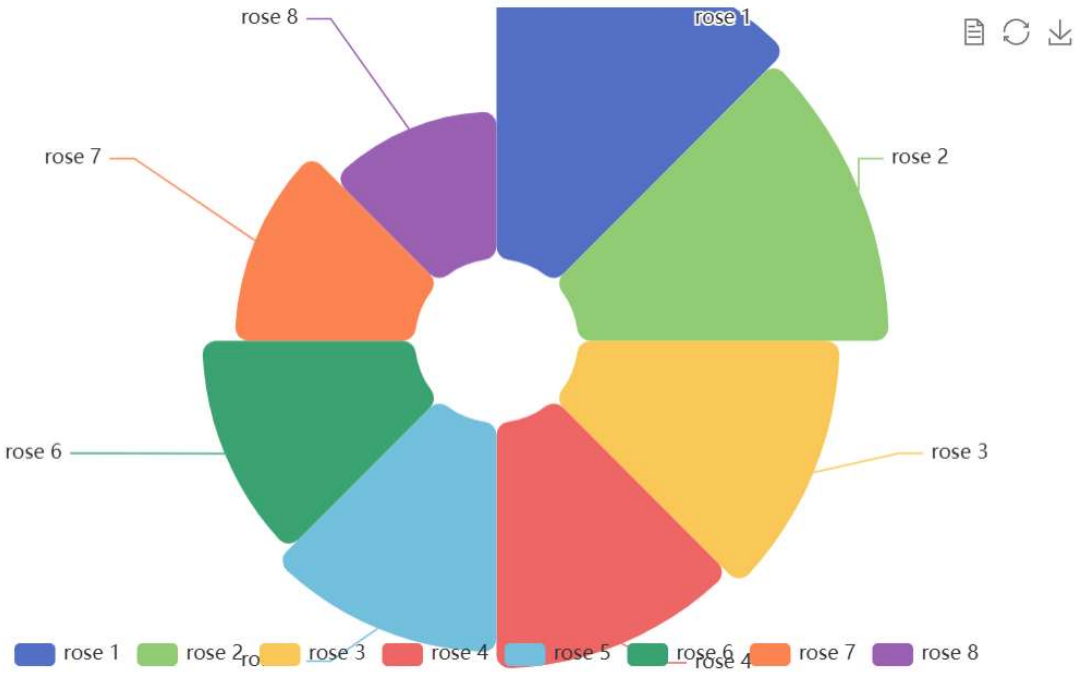
• **MM**: The number of trade instructions or data analysis tasks to be processed via cache.

• **θj**: The weight of the j-th task, representing the importance of that task in data processing.

• **Rj**: The execution time of the j-th task, representing the time required to read data from the cache and complete the task.

• **γj**: The abnormal delay factor, representing the impact of delays caused by system load or cache failure on task processing.

• **tj**: The processing delay of the j-th task, representing the time taken to process data from cache after reading.



Formula Analysis:

• **Cache Access Optimization**: The system uses caching technologies (like Redis or Memcached) to pre-load frequently accessed market data or trading signals into memory. The data transmission time **Tdata(i)** and cache access time **Taccess(i)** are optimized to ensure that data is read from the cache in milliseconds, avoiding delays caused by traditional storage.

• **Access Time Control**: A Gaussian decay factor $\exp(-((T_{access}(i)-T_{opt})^2/\sigma_{access}^2))$ is used to optimize cache access time, ensuring that access times approach the system's optimal value **Topt**. When cache access times deviate significantly from the optimal value, the system adjusts its caching strategy to ensure fast data access.

• **High-frequency Task Parallel Processing**: With caching technology, the platform can process large numbers of concurrent trade instructions and data analysis tasks **MM**, while ensuring low latency and high throughput for each task. The task execution time **Rj** and delay factor **γj** are used to evaluate the execution efficiency of tasks in the cache, and resources are dynamically adjusted as needed to optimize performance.

Historical Data Backtesting

The platform not only processes real-time data but also supports trading strategies by backtesting historical data (short-term historical analysis). By storing data from the past few minutes, hours, or days, A-Force can analyze historical price trends, identify patterns and trends, and use this analysis to inform trading decisions. To describe how the A-Force platform uses **historical data backtesting** to support trading strategies, the following formula integrates key factors such as **historical data storage**, **trend identification**, **pattern analysis**, and **strategy optimization**. By analyzing past data, the system can recognize market patterns and provide decision support for real-time trading.

$$T_{\text{backtest}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{data}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{history}}^{(i)} - T_{\text{opt}})^2}{\sigma_{\text{history}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation:

• **Tbacktest**: The total execution time after backtesting, representing the time from acquiring historical data to completing the analysis and generating decisions.

• **NN**: The number of historical data sources, representing the sources of the historical data analyzed by the platform (e.g., multiple markets, exchanges).

• **Wi**: The weight of the i-th historical data source, representing the importance of that source in the backtesting analysis.

• **Tdata(i)**: The historical data processing time for the i-th data source, representing the time required to retrieve and analyze data from the historical data storage.

• **Thistory(i)**: The historical data span of the i-th data source, representing the time range of the data used for backtesting (e.g., past few minutes, hours, or days).

• **Topt**: The optimal backtesting time for the system, representing the ideal time taken to analyze historical data.

• **σhistory²**: The standard deviation of historical data processing, representing the volatility of backtesting analysis, ensuring the stability of historical data analysis.

• **MM**: The number of tasks requiring backtesting analysis to generate trading strategies.

• **θj**: The weight of the j-th trading strategy task, representing the priority or importance of that strategy in the backtesting analysis.

• **Rj**: The execution time of the j-th task, representing the time taken to generate and execute the strategy from backtested data.

• **γj**: The abnormal delay factor, representing the delay in strategy execution due to data delays or system performance degradation.

• **tj**: The processing delay of the j-th strategy task, representing the delay from strategy generation to actual execution.

Formula Analysis:

• **Historical Data Processing and Backtesting Analysis**: The system first retrieves data from multiple historical data sources **Tdata(i)** and analyzes past market trends **Thistory(i)**, such as data from the past few minutes, hours, or days. By analyzing this historical data, the platform can identify market patterns and trends, providing the basis for trading strategies.

• **Volatility Control and Analysis Stability**: Using a Gaussian decay factor **$\exp(-((\text{Thistory}(i)-\text{Topt})^2/\sigma\text{history}^2))$**, the system ensures the volatility of historical data analysis is effectively controlled. When the historical data span approaches the optimal value **Topt**, the analysis results become more stable, and the system can provide accurate support for trading decisions based on the trends in historical data.

• **Trading Strategy Optimization**: Based on the analysis of historical data, the platform generates various trading strategies **Rj**. The priority of each strategy is controlled by the weight **θj**, while the execution time **Rj** and delay factor **γj** ensure that strategies can quickly respond to market dynamics and maximize potential returns.

Fast Data Recovery

In the event of system failures or network interruptions, A-Force quickly recovers the latest market data using caching and backup mechanisms, ensuring the continuity of trading strategies and high system availability. To describe how the A-Force platform implements **fast data recovery** through **caching and backup mechanisms**, the following formula combines key factors such as **fault detection**, **data backup**, **cache recovery**, and **system availability**. With these mechanisms, the system can rapidly recover the latest market data during a failure or network disruption, ensuring the continuous operation of trading strategies and high system availability.

$$T_{\text{recovery}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{cache}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{failure}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{failure}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation:

• **Trecovery**: Data recovery time, representing the time required to restore market data using caching and backup mechanisms after a system failure or network interruption.

• **NN**: The number of data sources, representing the sources used for caching and backup in the system.

• **Wi**: The weight of the i-th data source, representing the importance of that source in the recovery process.

• **Tcache(i)**: Cache recovery time for the i-th data source, representing the time taken to recover data from the cache.

• **Tfailure(i)**: Failure recovery time for the i-th data source, representing the time taken from the failure event to the completion of data recovery.

• **Tthreshold**: The failure recovery threshold, representing the maximum recovery time the system can tolerate. If recovery time exceeds this threshold, the system will automatically switch to a backup data source.

• **σfailure²**: Standard deviation of failure recovery time, representing the fluctuation in recovery times, ensuring consistency during fast data recovery.

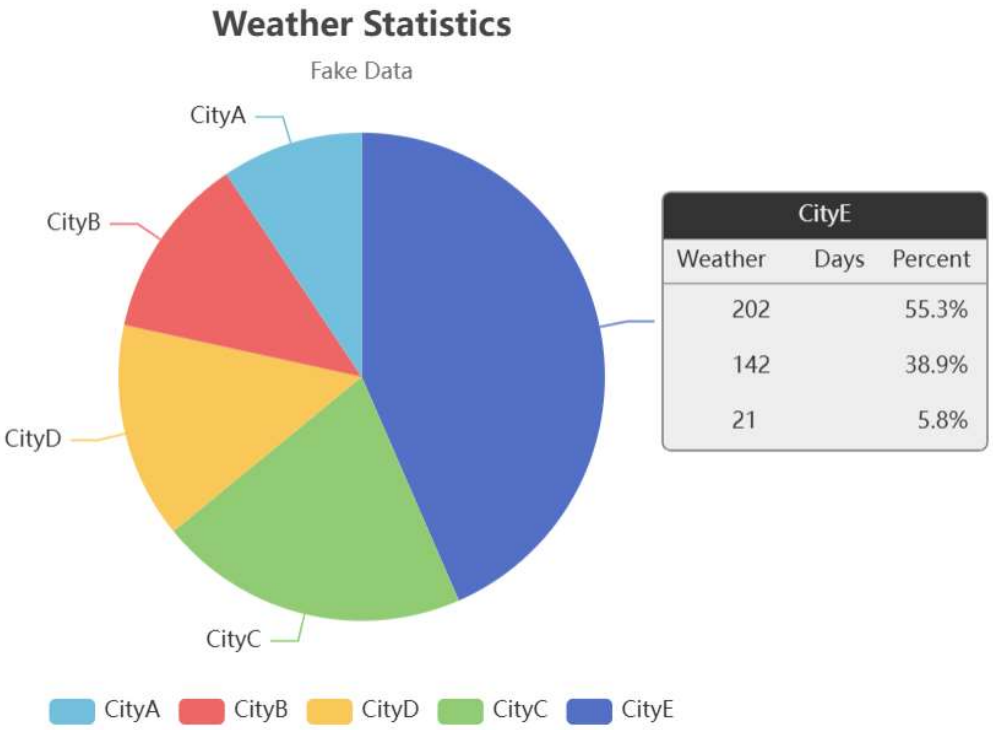
• **MM**: The number of backup data sources, representing the sources used for data backup in the system.

• **θj**: The weight of the j-th backup data source, representing the importance of that source in data recovery.

• **Rj**: Recovery time for the j-th backup data source, representing the time required to restore data from the backup.

• **γj**: Delay factor, representing the impact of delays during the data recovery process on the system's recovery speed.

• **tj**: Recovery delay for the j-th backup data source, representing the time delay during data recovery.



Formula Analysis:

• **Cache and Backup Recovery Optimization**: The system first utilizes the most recent market data stored in the cache **Tcache(i)** to quickly recover data. For failure recovery, the platform evaluates the recovery speed of each data source using recovery time **Tfailure(i)** and adjusts recovery sensitivity based on the threshold **Tthreshold**. The system adjusts the weight **Wi** based on the importance of each data source, ensuring that the most important sources are recovered first when a failure occurs.

• **Delay Control During Data Recovery**: Using a Gaussian decay factor **$\exp(-((T_{failure}(i)-T_{threshold})^2/\sigma_{failure}^2))$**, the system controls delays during the recovery process. Larger failure recovery times are attenuated by the standard deviation **$\sigma_{failure}^2$**, ensuring the data recovery process is minimally affected by delays.

• **Utilizing Backup Data Sources**: When the cache cannot recover the data, the platform uses the backup data source **Rj** to restore market data. The delay in the recovery process is controlled by **γj** and **tj**, ensuring that backup data sources can quickly take over data flow and restore market data during a failure.

Chapter 3: Market Traceability Module

1. Multi-Source Data Validation Mechanism

Data Source Integration

The A-Force platform integrates multiple data sources, including but not limited to major cryptocurrency exchanges (such as Binance, Coinbase, Kraken, etc.) and financial market data providers. By integrating data from multiple sources, the platform can provide a more comprehensive and accurate market view, avoiding potential errors from a single source. To describe how the A-Force platform uses **multi-source data integration** to provide a more comprehensive and accurate market view, the following formula combines factors such as **multiple data source integration**, **data fusion**, **error correction**, and **market view optimization**. By integrating data from multiple exchanges and financial market data sources, the platform can analyze the market more accurately and avoid potential errors from a single source.

$$T_{\text{integrated}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{data}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{error}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{error}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation:

• **Tintegrated**: The total processing time after data integration, representing the time required to process market data through multi-source data integration.

• **NN**: The number of data sources, representing the amount of data integrated from multiple exchanges, financial market providers, and other sources.

• **Wi**: The weight of the i-th data source, representing the importance of that source in the data integration process. The weight can be dynamically adjusted based on factors such as source reliability and data quality.

• **Tdata(i)**: The data transmission and processing time for the i-th data source, representing the time required to retrieve and process data from the source.

• **Terror(i)**: The error value for the i-th data source, representing the deviation or error that might occur during data retrieval from that source.

• **Tthreshold**: The error tolerance threshold, representing the maximum acceptable error for the system. When the error from a data source exceeds this threshold, the system will take corrective actions, such as readjusting the weight of that source or correcting the data.

• **σerror²**: The standard deviation of data errors, representing the fluctuation of errors in data sources, ensuring that sources with larger errors are properly adjusted.

• **MM**: The number of tasks requiring data fusion analysis, representing the number of tasks that need to merge multiple data sources to generate a complete market view.

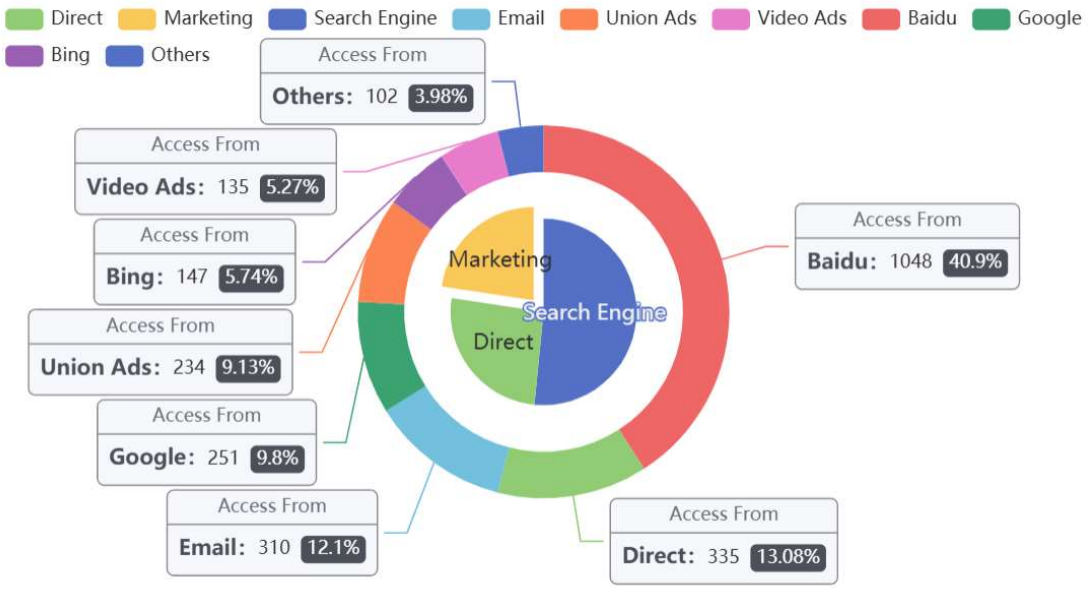
• **θj**: The weight of the j-th data fusion task, representing the importance of the task in the

data integration process.

• **Rj**: The execution time of the j-th data fusion task, representing the total processing time after merging multiple data sources.

• **γj**: The delay factor in data fusion, representing the impact of processing delays from data sources on the data fusion process.

• **tj**: The processing delay for the j-th data fusion task, representing the extra time required before completing the data fusion.



Formula Analysis:

• **Multi-Source Data Integration and Weight Adjustment**: The transmission time **Tdata(i)** and error value **Terror(i)** for each data source are dynamically adjusted based on the importance of the source through the weight **Wi**. The system uses a Gaussian decay factor **exp(−((Terror(i)−Tthreshold)²/σerror²))** to control the impact of sources with larger errors, ensuring that errors from a single data source do not affect the overall market view's accuracy.

• **Data Source Fusion and Error Correction**: By merging data from different sources, the system corrects potential errors **Terror(i)** for each source and ensures the combined data is more accurate based on the set error threshold **Tthreshold**. This ensures that market data collected from multiple exchanges (such as Binance, Coinbase, Kraken, etc.) provides a more comprehensive and accurate market view.

• **Data Fusion Tasks and Optimization**: Through the data fusion tasks **Rj**, the system integrates and optimizes the outputs from multiple data sources to generate a complete and comprehensive market view.

Accuracy Validation

By cross-validating data from multiple sources, A-Force can identify and correct price discrepancies between different exchanges or platforms. This validation mechanism helps the platform ensure that the market data used is highly accurate, providing a reliable basis for trading decisions. To describe how the A-Force platform uses **cross-validation** of multiple data sources to identify and correct price discrepancies between exchanges, the following formula combines key factors such as **data source cross-validation**, **price discrepancy correction**, **accuracy improvement**, and **decision optimization**. Through this mechanism, the platform ensures highly accurate market data and provides a reliable basis for trading decisions.

$$T_{\text{validation}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{data}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{price}}^{(i)} - T_{\text{reference}})^2}{\sigma_{\text{price}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation:

• **Tvalidation**: The total processing time after data accuracy validation, representing the time taken to process market data after cross-validation.

• **NN**: The number of data sources, representing the number of market data sources being cross-validated (e.g., data from multiple exchanges).

• **Wi**: The weight of the i-th data source, representing the importance of that source in the data validation process.

• **Tdata(i)**: The raw data from the i-th data source, representing unvalidated data.

• **Tprice(i)**: The price data from the i-th data source, representing the price information provided by that exchange.

• **Treference**: The reference price, representing the benchmark price calculated through cross-validation across multiple data sources.

• **σprice²**: The standard deviation of price differences, representing the fluctuation in price between different data sources, used to calculate the tolerance for price discrepancies.

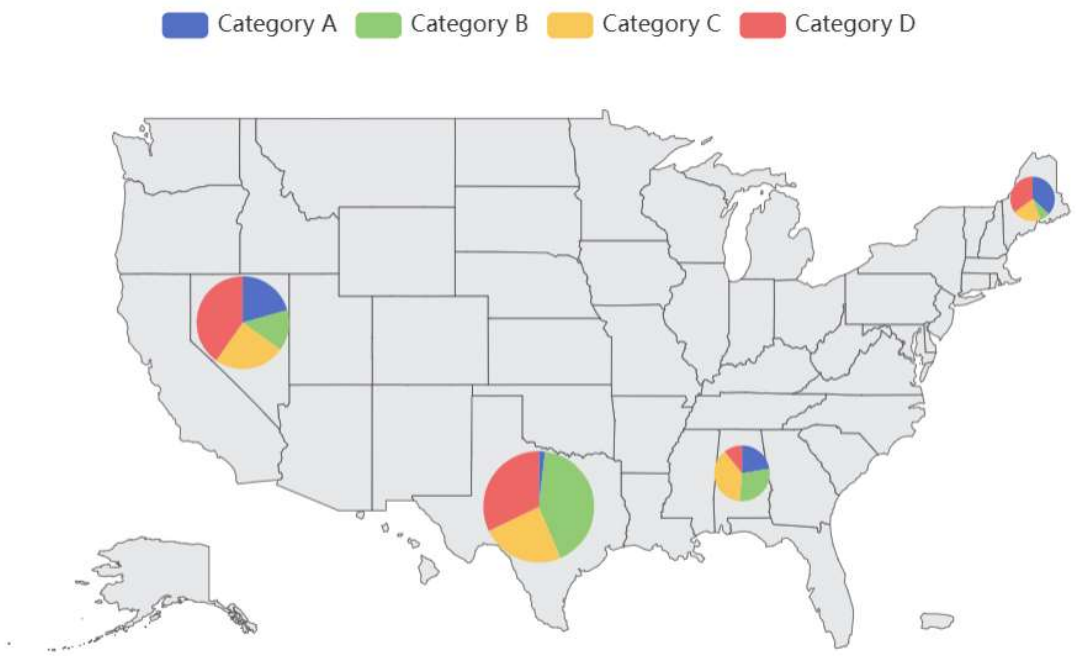
• **MM**: The number of tasks requiring price discrepancy correction through cross-validation.

• **θj**: The weight of the j-th price correction task, representing the importance of that task in the market data accuracy validation.

• **Rj**: The execution time of the j-th task, representing the time required to correct price discrepancies.

• **γj**: The delay factor, representing the time increase due to data corrections or system delays.

• **tj**: The processing delay of the j-th correction task, representing the time required from correcting the data to final validation.



Formula Analysis:

• **Cross-validation and Price Discrepancy Correction**: The system cross-validates data from multiple exchanges, especially comparing price discrepancies **Tprice(i)** with the reference price **Treference**. Using a Gaussian decay factor **$\exp(-((T_{price}(i)-T_{reference})^2/\sigma_{price}^2))$**, the system identifies and corrects price differences between platforms, improving the accuracy of price data. Larger price discrepancies lead to adjustments in data source weights, ensuring that the most accurate data is prioritized.

• **Price Discrepancy Volatility Control**: The standard deviation of price discrepancies **σ_{price}^2** is used to control the tolerance for price fluctuations between data sources. If a data source's price deviates from the reference price beyond a set range, the system will correct that data source's price and dynamically adjust its weight **Wi** in the system.

• **Task Parallelization and Delay Control**: By processing multiple correction tasks **Rj** in parallel, the system can correct price discrepancies from multiple data sources simultaneously. The delay in the correction process is influenced by the delay factor **γj** and the delay time **tj**, ensuring the data correction process is efficient.

Anomaly Detection and Correction

The platform employs efficient anomaly detection algorithms to automatically identify inconsistencies in market data (such as abnormal price fluctuations, missing data, etc.) and automatically correct or flag these anomalies, improving the reliability of market data. To describe how the A-Force platform uses **efficient anomaly detection algorithms** to automatically detect and correct inconsistencies in market data, the following formula combines key factors such as **anomaly detection**, **data correction**, and **quality control**. Through these mechanisms, the platform can identify issues such as price fluctuations and data loss, and automatically correct or flag them, thereby enhancing the credibility and accuracy of the market data.

$$T_{\text{exception}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{data}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{anomaly}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{anomaly}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} (\theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j)))$$

Formula Explanation:

• **Texception**: The total processing time after anomaly detection and correction, representing the time required to process and correct data using the anomaly detection algorithm.

• **NN**: The number of data sources, representing the number of market data sources used for anomaly detection.

• **Wi**: The weight of the i-th data source, representing the importance of that source in anomaly detection.

• **Tdata(i)**: The raw data from the i-th data source, representing uncorrected market data obtained from the source.

• **Tanomaly(i)**: The anomaly value for the i-th data source, representing the anomaly detected in market data (e.g., price fluctuations, data missing, etc.).

• **Tthreshold**: The anomaly detection threshold, representing the standard for recognizing and correcting anomalous data. When market data exceeds this threshold, the system will either flag it as an anomaly or correct it.

• **σanomaly²**: The standard deviation of anomalous data, representing the fluctuation in anomaly data, controlling the tolerance for anomalies to ensure that only clearly anomalous values are corrected.

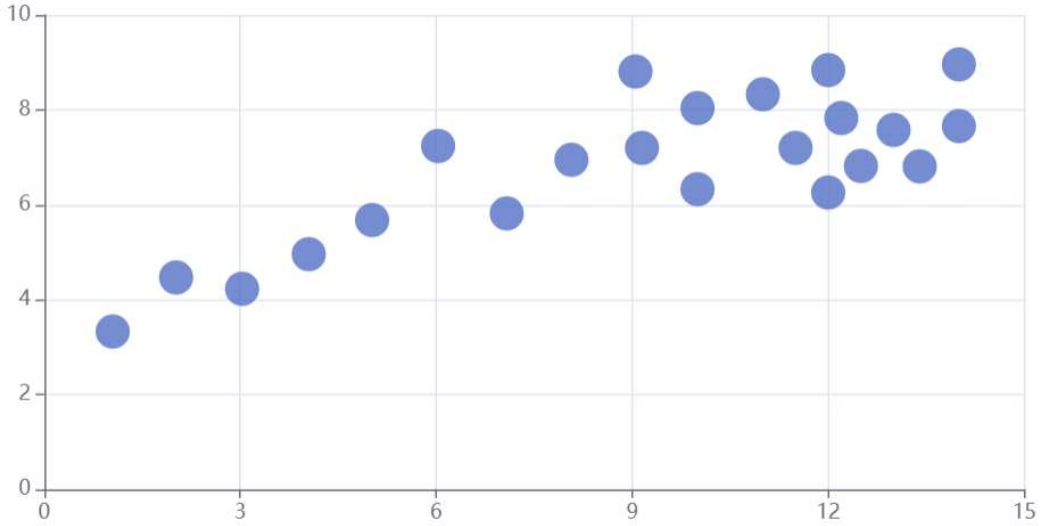
• **MM**: The number of anomaly data tasks that require correction or flagging.

• **θj**: The weight of the j-th anomaly correction task, representing the priority of that task in the anomaly correction process.

• **Rj**: The execution time of the j-th correction task, representing the time required to correct an anomaly from identification to resolution.

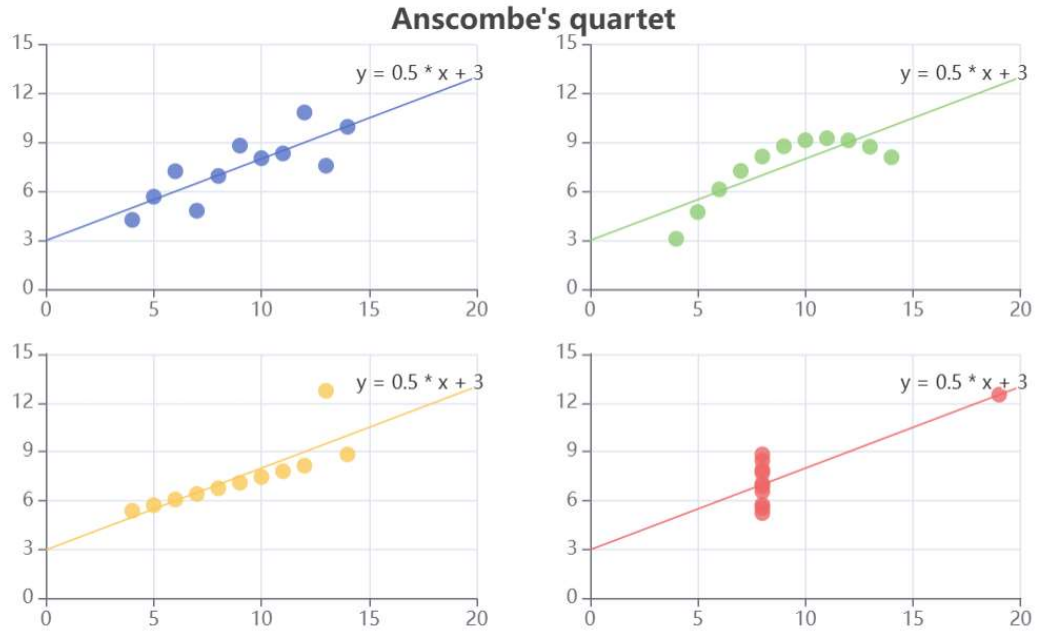
• **γj**: The delay factor, representing potential delays during the correction process.

• **tj**: The processing delay for the j-th task, representing the delay between flagging or correcting the anomaly and completing the task.

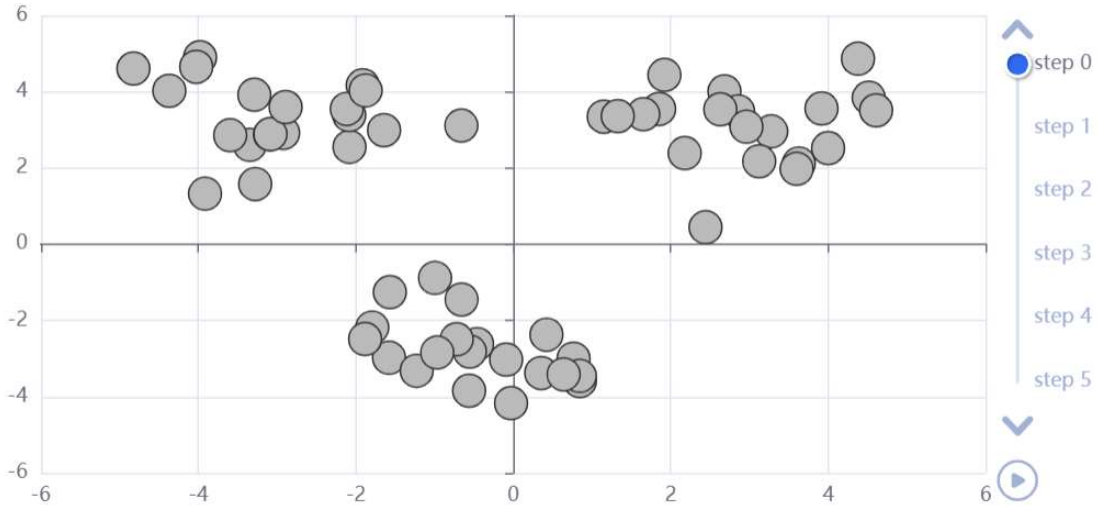


Formula Analysis:

• **Anomaly Detection and Correction**: The system uses anomaly detection algorithms to identify inconsistencies in market data (e.g., abnormal price fluctuations, data loss). By comparing with the set threshold **Tthreshold**, when the anomaly value **Tanomaly(i)** exceeds the threshold, the system flags or automatically corrects the data. The Gaussian decay factor **exp(−((Tanomaly(i)−Tthreshold)²/σanomaly²))** is used to optimize the anomaly detection and correction process, ensuring that only clearly anomalous data is corrected.



• **Anomaly Fluctuation and Standard Deviation Control**: The standard deviation of anomalous data **σanomaly²** is used to measure data fluctuations. If the data fluctuation exceeds the normal range, the system flags it as an anomaly. By controlling the standard deviation, the system can adapt to different market volatilities, avoiding over-correction of minor fluctuations.



• **Correction Tasks and Delay Optimization**: Through correction tasks **Rj** and delay factors **γj**, the system can promptly correct data flagged as anomalies. The priority of tasks is adjusted by **θj**, ensuring that critical anomalies are processed first. The delay **tj** in the correction process is controlled to minimize processing time and ensure the system responds quickly.

2. Blockchain Data Traceability

A-Force utilizes blockchain technology to store the hash values of key transaction data, ensuring the immutability and transparency of all transactions. By recording data on the blockchain, A-Force can provide complete traceability for each transaction, allowing the authenticity and historical records of transactions to be verified at any time. To describe how the A-Force platform uses **blockchain technology** to ensure **immutability** and **transparency** of transaction data, the following formula combines factors such as **data hash storage**, **transaction traceability**, **transparency assurance**, and **historical record verification**. By storing the hash values of transaction data on the blockchain, the platform ensures the authenticity of transactions and provides complete access to historical record queries.

$$T_{\text{blockchain}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{data}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{hash}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{hash}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation:

• **Tblockchain**: The data processing time based on blockchain technology, representing the time required to acquire data, store it on the blockchain, and validate it.

• **NN**: The number of data sources, representing the multiple data sources used on the platform where blockchain technology is applied for storage.

• **Wi**: The weight of the i-th data source, representing the importance of that source in blockchain data storage and validation.

• **Tdata(i)**: The raw data from the i-th data source, representing transaction or market data.

• **Thash(i)**: The hash value of the i-th data source, representing the encrypted hash of the transaction or data used to ensure data integrity and immutability.

• **Tthreshold**: The hash value verification threshold, representing the permissible fluctuation range of the hash values. When the hash value differs from the one stored on the blockchain, the system will consider the data as tampered with.

• **σhash²**: The standard deviation of hash value verification, representing the tolerance for fluctuations in the hash values.

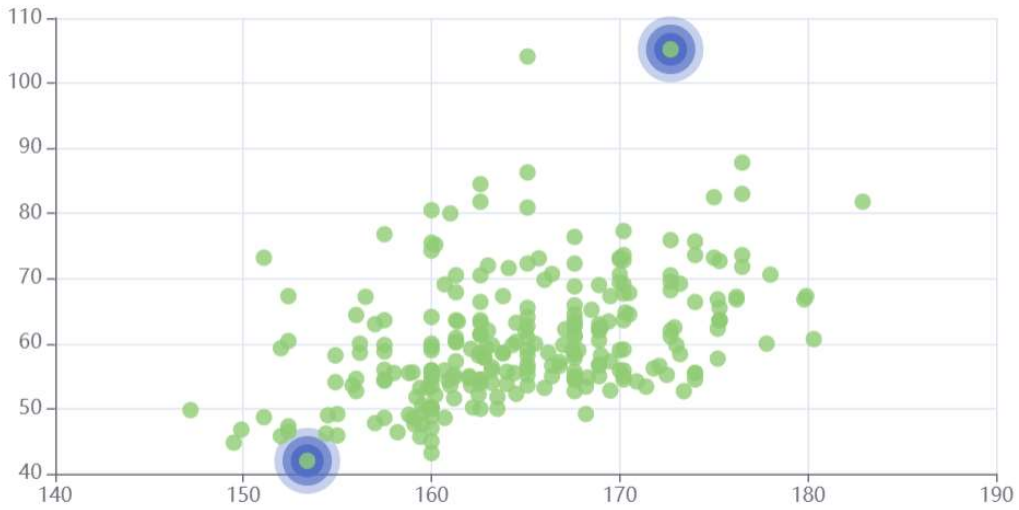
• **MM**: The number of tasks requiring transaction or data verification.

• **θj**: The weight of the j-th transaction verification task, representing the importance of that task in blockchain verification.

• **Rj**: The execution time of the j-th transaction, representing the time required to generate the hash value, store the data, and complete validation.

• **γj**: The delay factor, representing the impact of delay during the blockchain validation process on transaction handling.

• **tj**: The processing delay for the j-th task, representing the delay from storing data to validating the hash value.



Formula Analysis:

• **Blockchain Storage and Data Hashing**: The system computes the hash value **Thash(i)** for each data source and stores it on the blockchain. Hash verification is done by comparing the hash value with the value stored on the blockchain, ensuring the integrity and immutability of the data. Using a Gaussian decay factor $\exp(-((\text{Thash}(i)-\text{Tthreshold})^2/\sigma\text{hash}^2))$, the system optimizes the verification process to ensure that the data has not been tampered with.

• **Blockchain Transaction Verification**: Each time a new transaction or data needs to be stored, the system generates the transaction's hash value, then records the hash on the blockchain to ensure the immutability of the transaction. Thanks to the transparency of the blockchain, any user can verify the authenticity of the transaction and review its historical records at any time.

• **Transaction Task Processing and Delay Control**: Each transaction verification task **Rj** is processed in parallel. The platform optimizes task execution using the delay factor **γj** and the delay time **tj**, ensuring that despite potential delays in the blockchain verification process, transactions are validated and stored in a reasonable amount of time.

Smart Contract Execution

To further enhance transparency, A-Force uses smart contracts to automate the execution and validation of trades. The conditions and results of each transaction can be verified through smart contracts on the blockchain, ensuring compliance and automatic execution of transactions. To describe how the A-Force platform uses **smart contracts** to automate transaction execution and validation, the following formula incorporates key factors such as **smart contract conditions**, **execution validation**, **blockchain assurance**, and **automated trade execution**. Through smart contracts, the platform ensures the compliance of each trade and automatically verifies and executes transactions on the blockchain.

$$T_{\text{contract}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{transaction}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{compliance}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{compliance}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation:

• **Tcontract**: The total transaction time after smart contract execution, representing the time required for the smart contract to automatically execute and validate the transaction.

• **NN**: The number of transaction conditions, representing the number of rules or conditions defined in the smart contract.

• **Wi**: The weight of the i-th transaction condition, representing the importance of that condition in transaction validation.

• **Ttransaction(i)**: The execution time of the i-th transaction, representing the time from meeting the smart contract conditions to the beginning of the transaction execution.

• **Tcompliance(i)**: The compliance check time for the i-th transaction, representing the time required to check if the transaction meets the contract terms.

• **Tthreshold**: The compliance threshold, representing the standard for compliance checks. When transaction conditions do not meet this threshold, the transaction is flagged as non-compliant.

• **σcompliance²**: The standard deviation of compliance, representing the volatility of transaction conditions, ensuring that contract conditions are executed consistently across different trading environments.

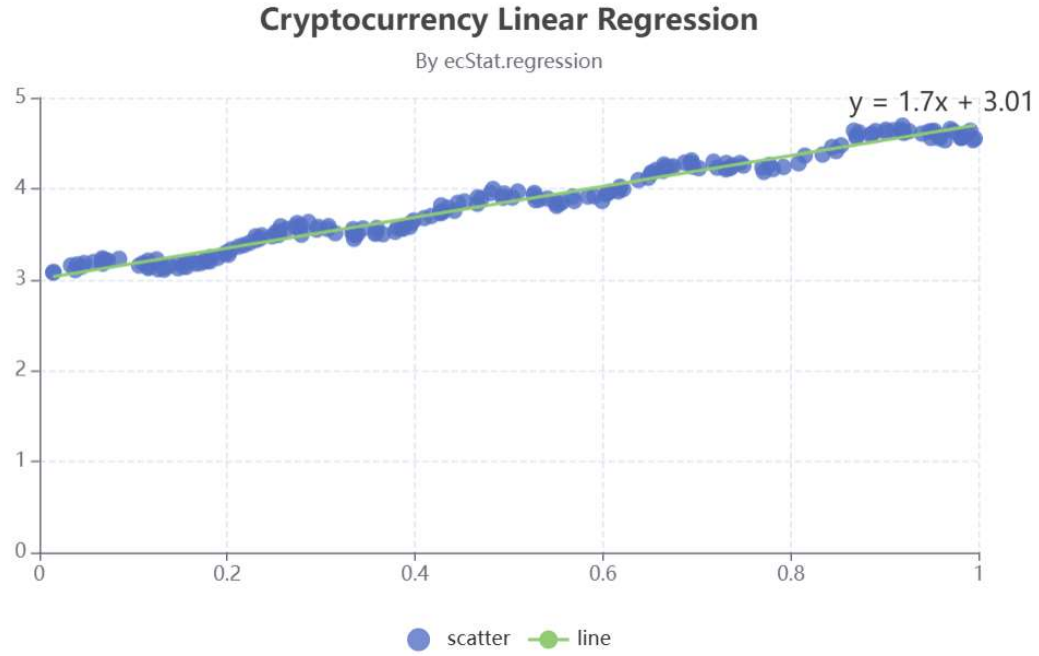
• **MM**: The number of transactions requiring smart contract execution.

• **θj**: The weight of the j-th smart contract execution task, representing the priority of that task in smart contract execution.

• **Rj**: The execution time of the j-th task, representing the time required to execute the smart contract.

• **γj**: The delay factor, representing the impact of delays in the contract execution process on overall transaction time.

• **tj**: The processing delay for the j-th task, representing the time from smart contract execution to the final completion of the transaction.



Formula Analysis:

• **Smart Contract Conditions and Compliance Check**: When executing a transaction, the system verifies the transaction based on the conditions defined in the smart contract (such as market price, time window, order size, etc.) **Tcompliance(i)**. When the transaction conditions meet the standards set by the smart contract, the transaction proceeds; otherwise, it is flagged as non-compliant and blocked. The Gaussian decay factor **$\exp(-((T_{compliance}(i)-T_{threshold})^2/\sigma_{compliance}^2))$** is used to ensure the stability of the compliance check process.

• **Compliance and Execution Delay Optimization**: During smart contract execution, the delay factor **γj** and processing delay **tj** affect the execution time of each transaction. The system optimizes these delays to ensure efficient execution of smart contracts, preventing the loss of trading opportunities due to delays.

• **Automated Execution and Validation of Smart Contracts**: Smart contracts not only automatically execute trades but also validate each transaction's compliance on the blockchain. Each contract clause and condition is predefined and stored on the blockchain, ensuring transparency and automated execution. When the transaction meets the contract conditions, the system automatically submits the transaction instruction, reducing human intervention and ensuring compliance with trading requirements.

Irreversibility of Transaction Data

The immutability provided by blockchain technology means that once transaction data is recorded, it cannot be modified or deleted. This feature significantly enhances the security of the system and reduces the occurrence of fraudulent activities. To describe how the A-Force platform uses **blockchain immutability** to enhance the security of transaction data, the following formula combines key factors such as **data recording**, **blockchain validation**, **data immutability**, and **security assurance**. Through blockchain, the platform ensures that once transaction data is recorded, it cannot be altered or deleted, reducing fraud and improving system transparency and security.

$$T_{\text{immutability}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{data}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{integrity}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{integrity}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation:

• **Timmutability**: The total processing time for data immutability, representing the time required from data recording to blockchain validation and ensuring the data's immutability.

• **NN**: The number of data sources, representing the number of transaction data sources that undergo blockchain immutability verification.

• **Wi**: The weight of the i-th data source, representing the importance of that data source in ensuring the immutability of transaction data.

• **Tdata(i)**: The raw transaction data from the i-th data source, representing unprocessed transaction data.

• **Tintegrity(i)**: The integrity verification time for the i-th data source, representing the time required to verify the integrity of transaction data to ensure it has not been altered.

• **Tthreshold**: The integrity verification threshold, representing the maximum permissible integrity error. If data integrity falls below this threshold, the system will mark the data as invalid.

• **σintegrity²**: The standard deviation of integrity verification, representing the tolerance for fluctuations in data integrity, ensuring consistency during data verification.

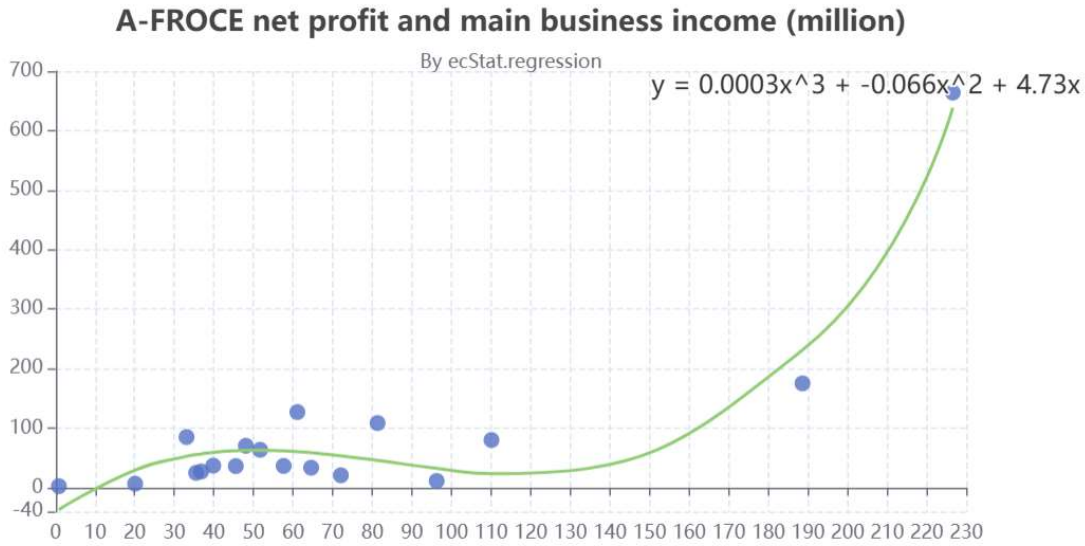
• **MM**: The number of tasks requiring immutability verification for transactions.

• **θj**: The weight of the j-th verification task, representing the importance of that task in ensuring the immutability of transaction data.

• **Rj**: The execution time of the j-th task, representing the time required to perform immutability verification for transaction data.

• **γj**: The delay factor, representing the impact of delays during the blockchain validation process on transaction processing.

• **tj**: The processing delay for the j-th task, representing the delay from data storage to the completion of verification.



Formula Analysis:

• **Data Recording and Integrity Verification**: The system first records transaction data and generates a hash value for each transaction, storing it on the blockchain. Through integrity verification **Tintegrity(i)**, the system ensures that transaction data cannot be modified or deleted at any time. The Gaussian decay factor **exp(−((Tintegrity(i)−Tthreshold)² / σ integrity²))** is used to ensure the stability of the data verification process, ensuring that only transactions meeting the integrity standard are accepted.

• **Immutability and Data Security**: Through the immutability feature of blockchain, transaction data once recorded cannot be altered or deleted. This feature enhances the security of the system, ensuring that all transaction records are authentic and reliable, and cannot be tampered with or influenced by external interference. The system uses the blockchain validation mechanism to verify each transaction, ensuring its integrity and immutability.

• **Delay Optimization and Transaction Execution**: Although there may be some delays in the blockchain verification process, the platform optimizes data processing through delay factors **γj** and delay time **tj** to ensure that transactions are validated and confirmed to be immutable within a reasonable time. The execution time of each verification task **Rj** is also optimized to ensure the security and real-time nature of transaction data.

3. Real-time Data Cleaning and Version Control

Real-time Data Cleaning

A-Force's data cleaning system processes market data in real time through algorithms to remove noise and errors. For example, in the case of data disconnections, duplicates, or erroneous transactions, the system will automatically repair or discard them, ensuring the purity of the market data. To describe how the A-Force platform uses **real-time data cleaning** to remove noise and errors from market data, the following formula combines key factors such as **data noise recognition**, **error correction**, **data modification**, and **cleaning efficiency**. Through these mechanisms, the platform can process and correct issues like disconnections, duplicates, or erroneous data in real time, ensuring data purity and reliability.

$$T_{\text{cleaning}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{data}}^{(i)} \cdot \exp\left(- \frac{(T_{\text{noise}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{noise}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation:

• **Tcleaning**: The total processing time after data cleaning, representing the time required for real-time cleaning algorithms to remove noise and correct erroneous data.

• **NN**: The number of data sources, representing the number of market data sources that require real-time cleaning.

• **Wi**: The weight of the i-th data source, representing the importance of that source in the cleaning process.

• **Tdata(i)**: The raw market data from the i-th data source, representing the uncleaned data.

• **Tnoise(i)**: The noise value for the i-th data source, representing the detected noise or errors in the market data (such as disconnections, duplicates, erroneous data, etc.).

• **Tthreshold**: The noise detection threshold, representing the tolerance for noise in the data. Data exceeding this threshold will be marked as noise and corrected.

• **σnoise²**: The standard deviation of noise, representing the fluctuation range of noise in the data, used to dynamically adjust the sensitivity of noise recognition.

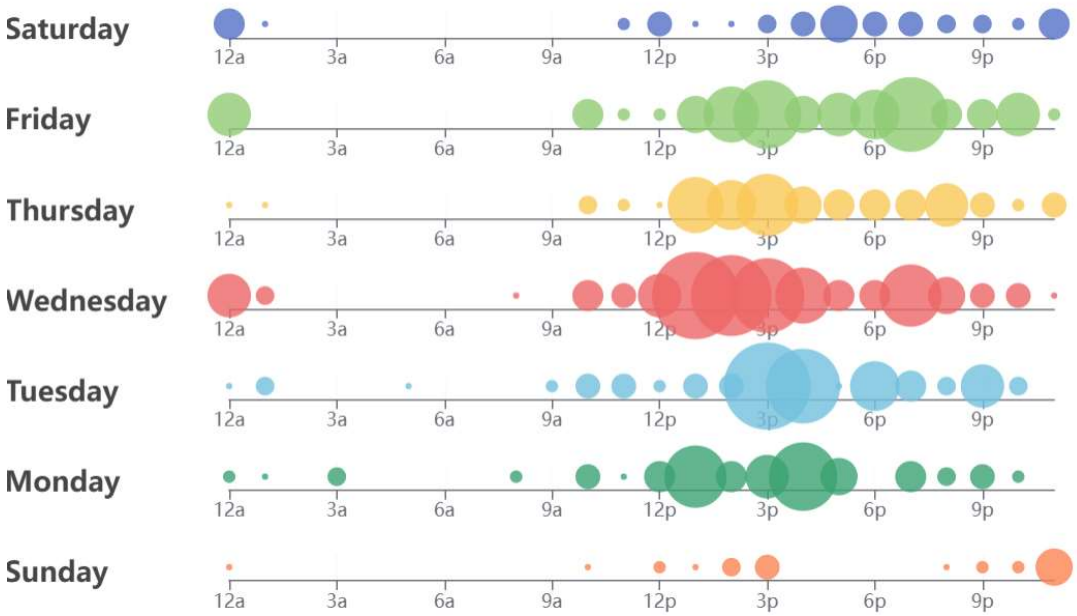
• **MM**: The number of error data tasks that need to be corrected or discarded.

• **θj**: The weight of the j-th data correction task, representing the priority of that task in the data cleaning process.

• **Rj**: The execution time of the j-th task, representing the time required to correct erroneous data or clean noise.

• **γj**: The delay factor, representing the impact of delays during the data cleaning process on system responsiveness.

• **tj**: The processing delay for the j-th task, representing the delay from data repair or discard to the completion of cleaning.



Formula Analysis:

• **Noise Recognition and Cleaning**: The system identifies inconsistencies or errors in the market data, such as disconnections, duplicate transactions, or abnormal fluctuations, using the noise value **Tnoise(i)**. By comparing with the set threshold **Tthreshold**, the system accurately identifies and removes the noise. The Gaussian decay factor **$\exp(-((T_{noise}(i)-T_{threshold})^2/\sigma_{noise}^2))$** is used to smooth the noise recognition process, ensuring that the cleaning process does not mistakenly remove valid data.

• **Data Repair and Discarding**: For data marked as erroneous, the system corrects it through correction tasks **Rj** or discards invalid data that cannot be fixed. The execution time of each correction task **Rj** and the delay factor **γj** ensure that data is repaired or discarded promptly, ensuring the accuracy and timeliness of market data.

• **Parallel Processing of Cleaning Tasks**: By processing multiple cleaning tasks in parallel, the platform can clean market data from different sources simultaneously, ensuring real-time data processing and integrity. The priority of tasks is dynamically adjusted through weights **θj**, ensuring that important erroneous data is processed first.

Version Control Mechanism

Every data point entering the system is assigned a unique version number, and the platform records the version history of every data update. This allows users to query historical versions at any time, ensuring data traceability and integrity. In situations where market fluctuations are significant, the system preserves multiple historical versions for later backtracking analysis. To describe how the A-Force platform uses a **version control mechanism** to assign a unique version number to each data entry and records each data update's version history, the following formula incorporates key factors such as **data version management**, **version history tracking**, **traceability assurance**, and **backtracking analysis**. Through version control, the platform ensures the integrity and traceability of data, supporting the querying of historical versions and market change analysis.

$$T_{\text{version}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{data}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{version}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{version}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} (\theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j)))$$

Formula Explanation:

• **Tversion**: The data processing time after version control, representing the time taken to apply the version control mechanism during data processing.

• **NN**: The number of data sources, representing the number of sources whose data will be managed through version control.

• **Wi**: The weight of the i-th data source, representing the importance of that source in version control.

• **Tdata(i)**: The raw data from the i-th data source, representing the data received by the platform.

• **Tversion(i)**: The version number for the i-th data source, representing the unique version number assigned to the data when it enters the system.

• **Tthreshold**: The version verification threshold, representing the maximum allowable version deviation. When the data version exceeds this threshold, the system flags it as a historical or inconsistent version.

• **σversion²**: The standard deviation of version deviation, representing the tolerance for fluctuations in data versions, ensuring stability between versions.

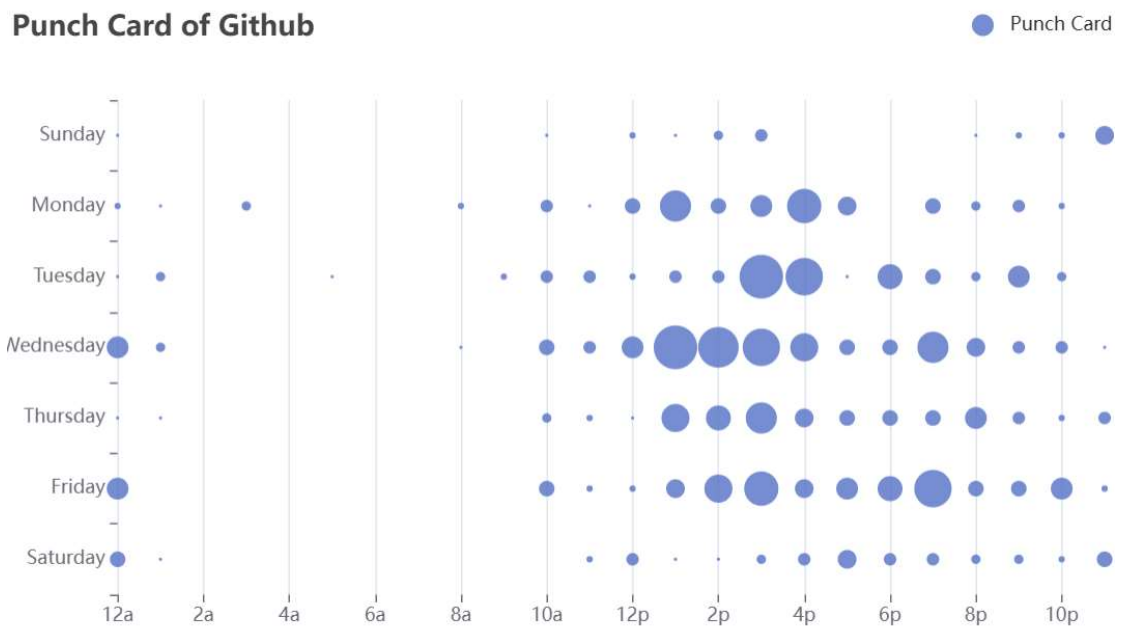
• **MM**: The number of historical version query tasks.

• **θj**: The weight of the j-th historical version query task, representing the influence of the task on querying historical versions.

• **Rj**: The execution time of the j-th version query task, representing the time required to query historical versions and complete analysis.

• **γj**: The delay factor, representing the impact of delays during the historical version query process on the results.

• **tj**: The processing delay for the j-th task, representing the delay from querying to obtaining the historical version response.



Formula Analysis:

• **Data Version Control and Management**: The platform assigns a unique version number **Tversion(i)** to each data source to ensure traceability for every data update. Using a Gaussian decay factor **$\exp(-((T_{version}(i)-T_{threshold})^2/\sigma_{version}^2))$**, the system ensures that version deviation is reasonably managed during version control, preventing data errors or inconsistencies caused by version differences.

• **Historical Version Management and Querying**: By preserving multiple historical versions, the system can perform backtracking analysis when significant market changes occur. Historical version query tasks **Rj** and delay factors **γj** are used to optimize the query process, ensuring timely access to relevant historical version data for analysis when needed.

• **Data Integrity and Traceability**: Version control not only ensures data traceability but also guarantees the integrity of data at different points in time. The platform records every data update history to provide complete historical versions for analysis when necessary. This is critical for market trend analysis, risk control, and auditing purposes.

Historical Data Management

The platform utilizes a historical data management tool to store and organize each version of transaction data. Even in the event of data source issues, A-Force can rely on its version control system to ensure that any historical data can be recovered or restored, preventing data loss or corruption. To describe how the A-Force platform uses a **historical data management tool** to ensure that each version of transaction data is stored and organized, the following formula combines key factors such as **data version management**, **data recovery**, **version control system**, and **historical data restoration**. Through these mechanisms, the platform ensures that even if a data source experiences issues, historical data can still be restored or recovered, preventing data loss or corruption.

$$T_{\text{history}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{data}}^{(i)} \cdot \exp\left(- \frac{(T_{\text{restore}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{restore}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation:

• **Thistory**: The total recovery time after historical data management, representing the time required to restore historical data using the version control system when a data source issue occurs.

• **NN**: The number of data sources, representing the number of data sources stored and managed through the historical data management tool.

• **Wi**: The weight of the i-th data source, representing the importance of that source in historical data management.

• **Tdata(i)**: The raw transaction data from the i-th data source, representing the uncorrected market data.

• **Trestore(i)**: The historical data restoration time for the i-th data source, representing the time from storing the data to its recovery.

• **Tthreshold**: The data recovery threshold, representing the maximum allowed recovery time. If recovery exceeds this time, the data will be marked as recovery failure.

• **σrestore²**: The standard deviation of restoration time, representing the fluctuations that may occur during the recovery process, ensuring stability in the restoration process.

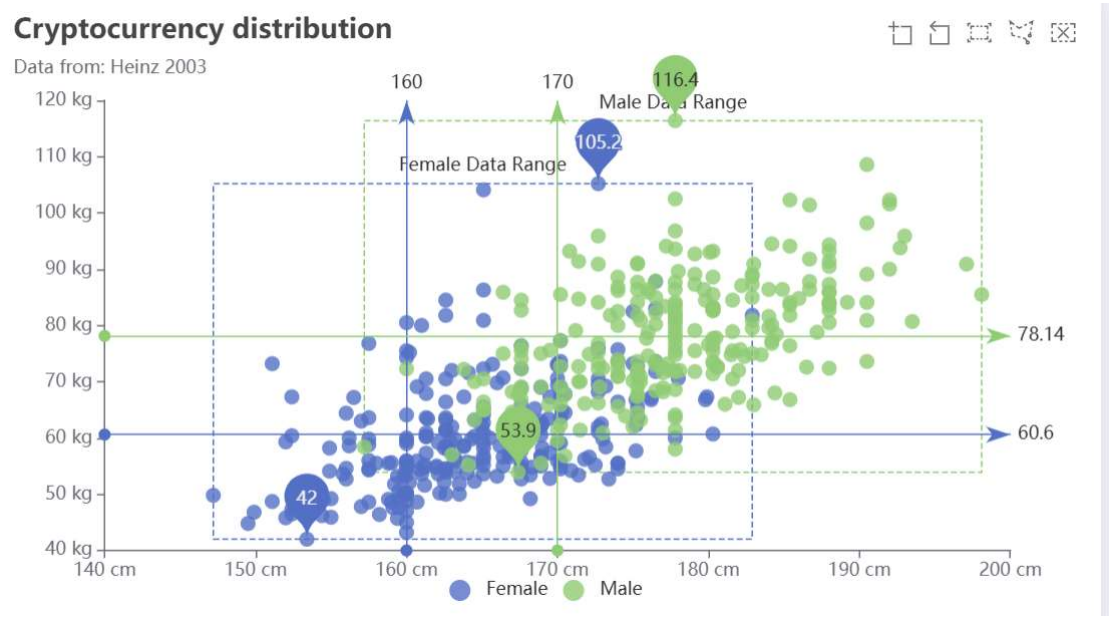
• **MM**: The number of historical data tasks that need to be restored, representing the number of data entries that need to be recovered through the historical data management tool.

• **θj**: The weight of the j-th historical data recovery task, representing the priority of that task in the data recovery process.

• **Rj**: The execution time of the j-th task, representing the time required to restore specific historical data.

• **γj**: The delay factor, representing the impact of delays during the recovery process on data restoration time.

• **tj**: The processing delay for the j-th task, representing the time from the data recovery request to the final completion of data restoration.



Formula Analysis:

• **Historical Data Storage and Recovery**: The system stores and organizes each version of transaction data through the historical data management tool, allowing for recovery using the version control system when data source issues occur. The Gaussian decay factor **$\exp(-((T_{restore}(i)-T_{threshold})^2/\sigma_{restore}^2))$** is used to ensure that the recovery time **$T_{restore}(i)$** stays within a reasonable range, preventing excessive delays during the recovery process.

• **Data Recovery Process and Optimization**: For historical data restoration tasks **R_j**, the system automatically optimizes the recovery process based on priority and delay factors **γ_j**. The execution time of each task **R_j** and delay time **t_j** will affect recovery efficiency, and the platform uses optimization algorithms to ensure that historical data is restored as quickly as possible in the event of failure or data corruption.

• **Version Control and Integrity Assurance**: The platform uses the version control system to ensure that every version of historical data can be accurately restored. Even in the event of data source failure, the management and recovery mechanisms for historical versions ensure data integrity, preventing interruptions in trading strategies caused by data loss or corruption.

4. Precision Order Placement Module

1. Dynamic Order Depth Optimization

Market Depth Analysis

A-Force analyzes the market's buy and sell order book depth in real time, dynamically adjusting the price and quantity of orders based on the market's supply and demand. By monitoring market depth, the platform determines the optimal order placement, ensuring the price is set to facilitate successful trades while avoiding slippage caused by setting prices too high or too low. To describe how the A-Force platform uses **market depth analysis** to dynamically adjust the price and quantity of orders, the following formula incorporates key factors such as **order book depth**, **supply and demand analysis**, **price adjustment**, and **slippage avoidance**. By continuously monitoring market depth, the platform can determine the best order placement, ensuring trade success and avoiding slippage caused by price fluctuations.

$$T_{\text{depth}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{order}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{spread}}^{(i)} - T_{\text{optimal}})^2}{\sigma_{\text{spread}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} (\theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j)))$$

Formula Explanation:

• **Tdepth**: The order optimization time after market depth analysis, representing the time required to optimize order price and quantity based on market supply and demand conditions.

• **NN**: The number of data sources, representing the number of buy and sell order book data sources from different markets or exchanges.

• **Wi**: The weight of the i-th data source, representing the importance of that market order book data in market depth analysis.

• **Torder(i)**: The raw order data from the i-th data source, representing the buy and sell order information obtained from the order book.

• **Tspread(i)**: The price spread for the i-th data source, representing the current market's bid-ask spread.

• **Toptimal**: The optimal price spread, representing the ideal price difference in market depth analysis, used to avoid excessive price difference and slippage.

• **σspread²**: The standard deviation of the price spread, representing the volatility of market depth, used to ensure flexibility and stability during price adjustments.

• **MM**: The number of trading tasks requiring market depth analysis.

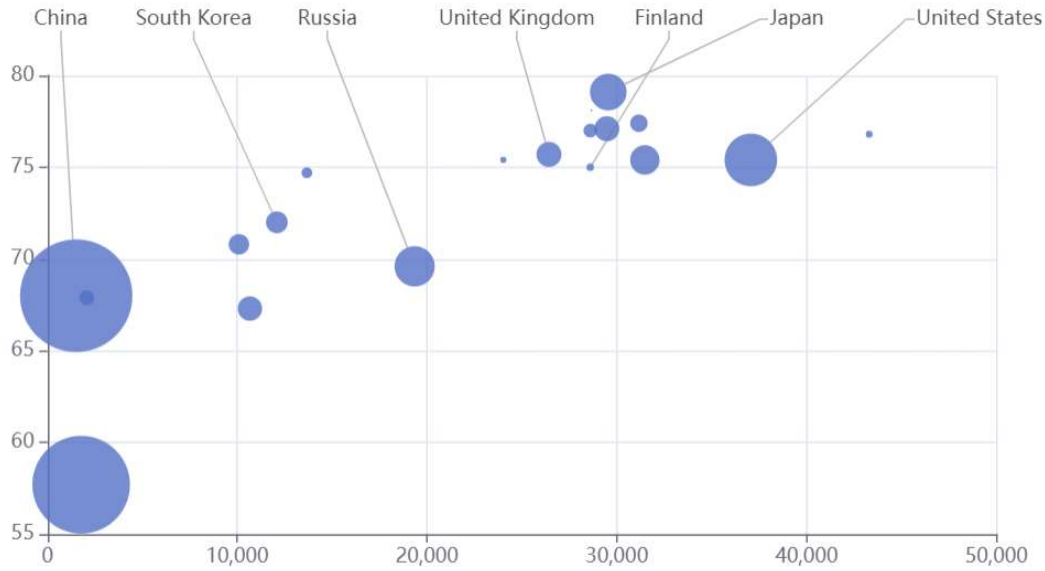
• **θj**: The weight of the j-th order adjustment task, representing the priority of that task in adjusting the order price.

• **Rj**: The execution time of the j-th task, representing the time required to adjust the order.

• **γj**: The delay factor, representing the impact of price adjustment delays caused by market

fluctuations.

• **tj**: The processing delay for the j-th task, representing the time from market depth analysis to order adjustment execution.



Formula Analysis:

• **Market Depth Analysis and Price Adjustment**: The platform analyzes the market depth by monitoring the buy and sell order books **Torder(i)** to calculate the current bid-ask spread **Tspread(i)**. By comparing with the ideal optimal price spread **Toptimal**, the platform dynamically adjusts the order price and quantity, ensuring that the transaction is successful while avoiding slippage caused by excessively low or high prices. The Gaussian decay factor $\exp(-((Tspread(i)-Toptimal)^2/\sigma spread^2))$ is used to optimize the price spread and ensure effective control over market depth volatility.

• **Slippage Avoidance and Order Optimization**: To avoid slippage caused by overly low or high order prices, the system adjusts the order placement based on current market supply and demand conditions. By adjusting the order price, the platform ensures that trades can be executed successfully without triggering slippage. The priority **θj** and execution time **Rj** of each order adjustment task affect the effectiveness of market depth analysis and order price optimization.

• **Delay Optimization and Real-time Adjustment**: The platform optimizes the response time for market depth analysis and order adjustment using delay factors **γj** and delay time **tj**. Despite market fluctuations potentially causing delays, the platform ensures that orders can quickly adjust to market changes, preventing adverse price adjustments caused by delays.

Auto Order Adjustment

Based on market fluctuations, A-Force automatically adjusts the order price and quantity. For example, when market volatility increases, the platform will automatically adjust the price range of the orders to avoid orders being left unfilled due to sharp price changes. To describe how the A-Force platform automatically adjusts the order price and quantity based on market fluctuations, the following formula incorporates key factors such as **market volatility monitoring**, **order adjustment**, **price range optimization**, and **trade execution strategies**. Through this mechanism, the platform can automatically adjust the order price range when market volatility increases, ensuring that orders are executed even during periods of sharp price movements.

$$T_{\text{order-adjustment}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{order}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{volatility}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{volatility}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} (\theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j)))$$

Formula Explanation:

• **Torder-adjustment**: The total processing time after order adjustment, representing the time required to automatically adjust the order price and quantity when market volatility changes.

• **NN**: The number of data sources, representing the number of market volatility data sources monitored by the platform.

• **Wi**: The weight of the i-th data source, representing the importance of that source in order adjustment.

• **Torder(i)**: The raw order data from the i-th data source, representing the order information obtained by the platform.

• **Tvolatility(i)**: The market volatility indicator for the i-th data source, representing the current volatility level in the market.

• **Tthreshold**: The volatility threshold, representing the point at which the platform begins adjusting order prices and quantities when market volatility reaches this threshold.

• **σvolatility²**: The standard deviation of market volatility, representing the fluctuation range of volatility, used to optimize the adjustment of the order price range.

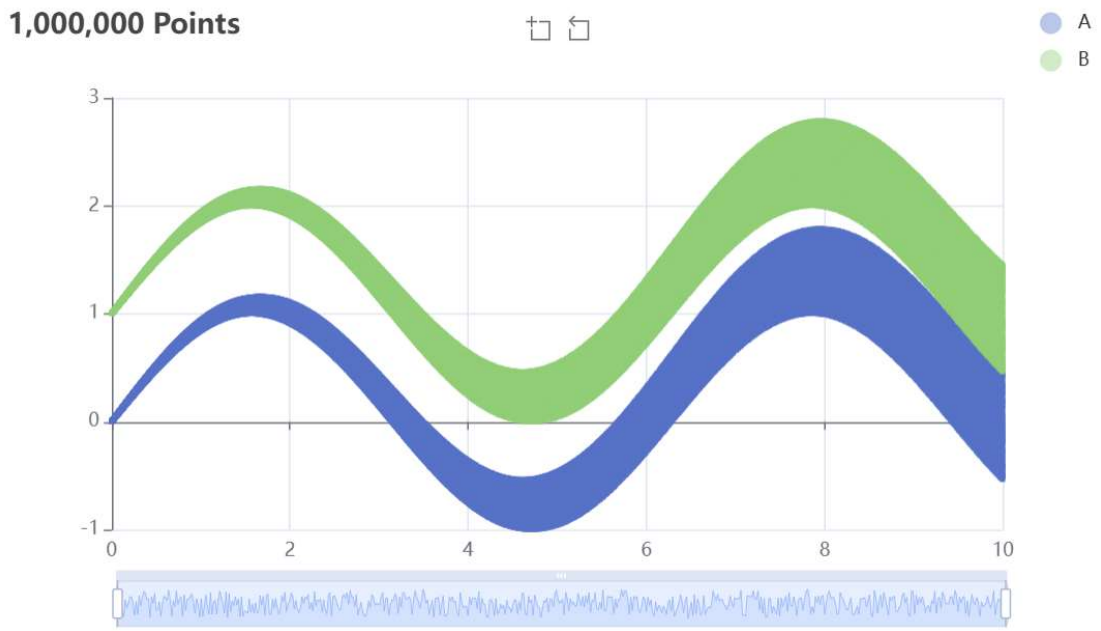
• **MM**: The number of order adjustment tasks, representing the number of orders that need to be adjusted by the platform.

• **θj**: The weight of the j-th order adjustment task, representing the influence of that task on price adjustments.

• **Rj**: The execution time of the j-th task, representing the time required to adjust the order price and quantity.

• **γj**: The delay factor, representing the impact of market fluctuations on order adjustment delays.

• **tj**: The processing delay for the j-th task, representing the time from market volatility change to the completion of order adjustment.



Formula Analysis:

• **Market Volatility Monitoring and Order Adjustment**: The platform monitors market volatility **Tvolatility(i)** in real-time and compares it to the set volatility threshold **Tthreshold**. When market volatility exceeds the threshold, the system adjusts the order price and quantity according to the volatility, ensuring that orders can be successfully filled even in a volatile market. The Gaussian decay factor **exp(−((Tvolatility(i)−Tthreshold)²/σvolatility²))** is used to optimize the sensitivity of price range adjustments, ensuring more flexible adjustments during periods of significant market volatility.

• **Order Price Range Optimization**: To avoid orders being unfilled due to sharp price fluctuations, the platform dynamically adjusts the price range of orders. The system optimizes order prices based on volatility data, ensuring orders can be executed promptly during high volatility and avoiding excessive prices that may cause slippage. The execution time **Rj** and delay factor **γj** for each order adjustment task ensure that the adjustment responds quickly to market changes.

• **Task Priority and Delay Optimization**: By adjusting task priority **θj** and optimizing the delay for order adjustments, the platform ensures that important orders are prioritized for adjustment when market volatility increases. The delay time **tj** controls the responsiveness of order adjustments, ensuring that order prices and quantities quickly adapt to market fluctuations.

Slippage Control

The platform combines market depth and liquidity data to intelligently predict potential slippage risks, and by adjusting the order price and quantity, it minimizes the occurrence of slippage, thus improving trade execution efficiency and stability. To describe how the A-Force platform combines **market depth** and **liquidity data** to intelligently predict potential slippage risks and reduce slippage through adjustments in order prices and quantities, the following formula incorporates key factors such as **market depth analysis**, **liquidity prediction**, **price adjustment**, and **slippage reduction strategies**. Through this mechanism, the platform can optimize trade execution efficiency and stability.

$$T_{\text{slippage}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{depth}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{slippage-risk}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{slippage-risk}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} (\theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j)))$$

Formula Explanation:

• **Tslippage**: The total trade execution time after slippage control, representing the time required to reduce slippage by adjusting the order price and quantity.

• **NN**: The number of data sources, representing the number of market depth and liquidity data sources analyzed by the platform.

• **Wi**: The weight of the i-th data source, representing the importance of that source in slippage prediction.

• **Tdepth(i)**: The market depth data from the i-th data source, representing the buy and sell depth from the market order book.

• **Tslippage-risk(i)**: The slippage risk value for the i-th data source, representing the slippage risk predicted by the platform based on market depth and liquidity data.

• **Tthreshold**: The slippage risk threshold, representing the maximum acceptable slippage risk. If the slippage risk exceeds this threshold, the platform will take measures to reduce slippage.

• **σslippage-risk²**: The standard deviation of slippage risk, representing the volatility of slippage risk, used to control the flexibility of risk prediction and adjustment.

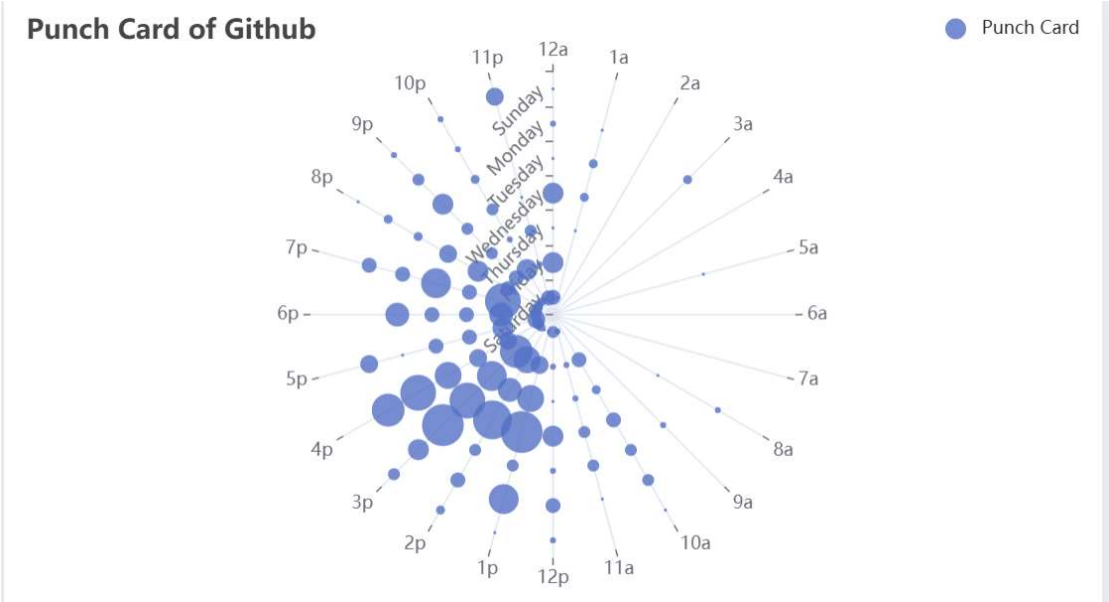
• **MM**: The number of trading tasks requiring slippage control.

• **θj**: The weight of the j-th slippage adjustment task, representing its priority in reducing slippage.

• **Rj**: The execution time of the j-th task, representing the time required to adjust the order price and quantity based on market depth and liquidity.

• **γj**: The delay factor, representing the impact of price adjustment delays caused by market fluctuations or data delays.

• **tj**: The processing delay for the j-th task, representing the time from slippage prediction to the adjustment of order price and quantity.



Formula Analysis:

• **Market Depth and Slippage Risk Prediction**: The system predicts slippage risk **Tslippage-risk(i)** by analyzing the buy and sell depth data **Tdepth(i)** and liquidity data, and compares it to the set threshold **Tthreshold**. When the slippage risk exceeds the threshold, the system adjusts the order price and quantity based on market conditions to reduce the occurrence of slippage. The Gaussian decay factor **$\exp(-((T_{slippage\text{-}risk}(i)-T_{threshold})^2/\sigma_{slippage\text{-}risk}^2))$** is used to optimize the accuracy of risk prediction.

• **Order Price and Quantity Adjustment**: When the system predicts high slippage risk, the platform automatically adjusts the order price and quantity to ensure that the trade can proceed smoothly without being affected by price fluctuations. The execution time **Rj** of each adjustment task and the delay factor **γj** ensure that these adjustments are implemented in a timely manner.

• **Slippage Control and Delay Optimization**: By using the delay factor **γj** and the processing delay **tj**, the system optimizes the response time during slippage control, ensuring that the adjusted order price matches market depth and liquidity as quickly as possible, avoiding slippage caused by delays.

2. Arbitrage Order Strategy

Cross-Market Arbitrage

A-Force monitors price discrepancies in real time across different exchanges, utilizing an arbitrage order strategy to perform arbitrage between markets. For example, when a cryptocurrency's price is lower on one exchange than on another, the platform automatically buys in the lower-priced market and sells in the higher-priced market to achieve risk-free arbitrage. To describe how the A-Force platform achieves risk-free arbitrage between exchanges using a **cross-market arbitrage** strategy, the following formula combines key factors such as **market price difference monitoring**, **arbitrage order strategy**, **price difference utilization**, and **arbitrage execution efficiency**. These mechanisms enable the platform to buy in lower-priced markets and sell in higher-priced markets to execute arbitrage.

$$T_{\text{arbitrage}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{price}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{spread}}^{(i)} - T_{\text{optimal}})^2}{\sigma_{\text{spread}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} (\theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j)))$$

Formula Explanation:

• **Tarbitrage**: The total trade time for cross-market arbitrage, representing the time from identifying market price differences to executing the arbitrage trade.

• **NN**: The number of data sources, representing the number of price difference data sources monitored in real time across different exchanges.

• **Wi**: The weight of the i-th data source, representing the importance of that data source in executing the arbitrage strategy.

• **Tprice(i)**: The market price for the i-th exchange, representing the current cryptocurrency price on that exchange.

• **Tspread(i)**: The price difference for the i-th exchange, representing the price difference between this exchange and others.

• **Toptimal**: The optimal price difference, representing the ideal price difference between two markets under arbitrage conditions.

• **σspread²**: The standard deviation of price differences, representing the tolerance for price fluctuations, used to control the flexibility of the arbitrage strategy.

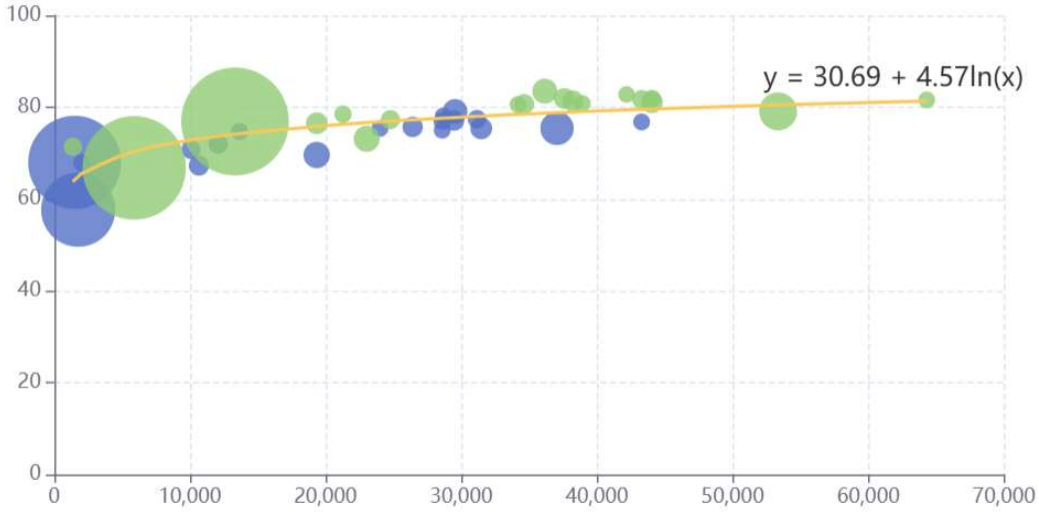
• **MM**: The number of arbitrage order tasks that need to be executed, representing the number of arbitrage trades that need to be performed between different markets.

• **θj**: The weight of the j-th arbitrage task, representing the priority of that task in executing the arbitrage trade.

• **Rj**: The execution time of the j-th task, representing the time required from identifying price differences to executing the arbitrage trade.

• **γj**: The delay factor, representing the impact of market fluctuations on arbitrage execution delays.

• **tj**: The processing delay for the j-th task, representing the time from identifying price differences to executing the arbitrage order.



Formula Analysis:

• **Market Price Difference and Arbitrage Execution**: The platform monitors the price difference **Tspread(i)** in real time across exchanges and compares it with the optimal price difference **Toptimal**. When it identifies a lower price in one market compared to others, the system buys in the lower-priced market and sells in the higher-priced market to execute arbitrage. The Gaussian decay factor **$\exp(-((Tspread(i)-Toptimal)^2/\sigma spread^2))$** optimizes the execution conditions of the arbitrage strategy, ensuring that arbitrage trades are executed quickly when there is a large price difference.

• **Arbitrage Order Strategy and Price Adjustment**: When an arbitrage opportunity arises, the platform automatically places buy orders in the lower-priced market and sell orders in the higher-priced market. The execution time **Rj** for each arbitrage task and the delay factor **γj** ensure that the arbitrage orders are executed quickly, maximizing the profit from the arbitrage opportunity.

• **Task Priority and Execution Optimization**: To ensure the efficiency of cross-market arbitrage, the platform dynamically adjusts the execution order based on task priority **θj**. The system optimizes delays, reducing arbitrage losses caused by market fluctuations or data transmission delays, ensuring that tasks are completed in the shortest time possible.

Bid-Ask Order Strategy

To maximize arbitrage opportunities, A-Force places both buy and sell orders on two different markets to ensure successful transactions before price discrepancies narrow. With this strategy, the platform can quickly capture arbitrage opportunities arising from price fluctuations and ensure that trades are not hindered by market liquidity issues. To describe how A-Force maximizes arbitrage opportunities through the **bid-ask order strategy**, the following formula combines key factors such as **buy and sell orders**, **price difference monitoring**, **liquidity analysis**, and **arbitrage execution efficiency**. By placing both buy and sell orders on two markets simultaneously, the platform can swiftly capture arbitrage opportunities from price fluctuations while ensuring smooth trade execution and avoiding execution failure due to market liquidity issues.

$$T_{\text{bid-ask}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{order}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{spread}}^{(i)} - T_{\text{optimal}})^2}{\sigma_{\text{spread}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation:

• **Tbid-ask**: The total transaction time for the bid-ask order strategy, representing the time required to implement arbitrage by placing buy and sell orders simultaneously.

• **NN**: The number of data sources, representing the number of order book data sources monitored by the platform from different markets or exchanges.

• **Wi**: The weight of the i-th data source, representing the importance of that market's order book data in the bid-ask order strategy.

• **Torder(i)**: The raw order data for the i-th market, representing the buy and sell prices and quantities of orders.

• **Tspread(i)**: The price difference for the i-th market, representing the price difference between the buy and sell orders.

• **Toptimal**: The optimal price difference, representing the ideal buy-sell price difference for the arbitrage strategy.

• **σspread²**: The standard deviation of price differences, representing the impact of market volatility on the price difference, used to control the flexibility of order price adjustments.

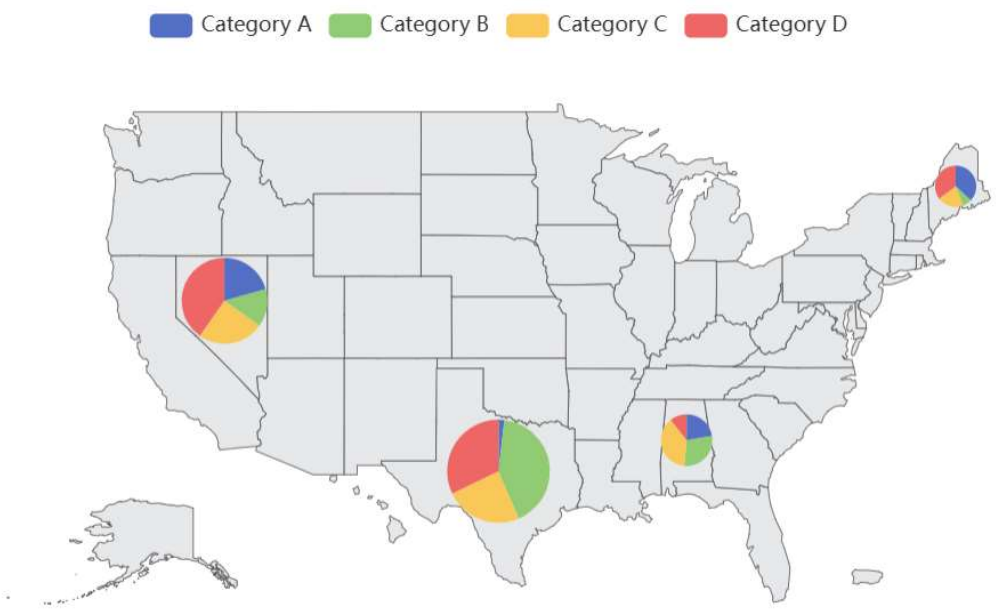
• **MM**: The number of bid-ask order strategy tasks to execute, representing the number of buy and sell orders that need to be executed by the platform.

• **θj**: The weight of the j-th order adjustment task, representing its importance in the bid-ask order strategy.

• **Rj**: The execution time of the j-th task, representing the time required from executing the order strategy to completing the trade.

• **γj**: The delay factor, representing the delay in order execution caused by market liquidity issues or fluctuations.

• **tj**: The processing delay for the j-th task, representing the time from order generation to actual trade execution.



Formula Analysis:

• **Bid-Ask Order Execution and Price Adjustment**: The platform places buy and sell orders on two markets simultaneously and monitors the market's buy-sell spread **Tspread(i)** in real-time. It compares the spread with the optimal price difference **Toptimal**. The system dynamically adjusts the order price using the Gaussian decay factor **$\exp(-((\text{Tspread}(i)-\text{Toptimal})^2/\sigma\text{spread}^2))$** to ensure that orders are filled when the price difference reaches the set condition.

• **Liquidity Assurance and Trade Execution**: The system evaluates market depth using **market liquidity analysis** to ensure that order prices do not fail due to insufficient market liquidity. The execution time **Rj** for each order adjustment task and the delay factor **γj** ensure that when liquidity is low, the order price can automatically adjust to prevent slippage or order failure caused by liquidity issues.

• **Task Priority and Delay Optimization**: To ensure the successful execution of the bid-ask order strategy, the system optimizes the execution order based on task priority **θj** and delay control **tj**. By optimizing task delays, the platform can quickly adjust orders in response to market fluctuations and liquidity changes, ensuring that arbitrage opportunities are captured in a timely manner.

Real-Time Adjustment

When market prices change, A-Force's arbitrage order strategy immediately reacts by adjusting the quantity and price of the orders to ensure that the user remains profitable throughout the arbitrage process. To describe how A-Force ensures profitability in arbitrage orders through **real-time adjustments** in response to market price changes, the following formula incorporates key factors such as **market price fluctuations**, **order adjustments**, **arbitrage strategy optimization**, and **profitability guarantee**. Through this mechanism, the platform can instantly respond to market price changes and automatically adjust the order quantities and prices, ensuring that the user remains profitable during the entire arbitrage process.

$$T_{\text{adjustment}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{price}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{adjustment}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{adjustment}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} (\theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j)))$$

Formula Explanation:

• **Tadjustment**: The total transaction execution time after real-time adjustments, representing the time required for the platform to adjust arbitrage order quantities and prices when market prices change.

• **NN**: The number of data sources, representing the number of market price and price fluctuation data sources monitored by the platform.

• **Wi**: The weight of the i-th data source, representing the importance of that source in the real-time adjustment process.

• **Tprice(i)**: The market price of the i-th data source, representing the current price information of the market.

• **Tadjustment(i)**: The adjustment value for the i-th market order, representing the quantity or price adjustment needed based on market price changes.

• **Tthreshold**: The adjustment threshold, representing the market price change value at which the system will immediately adjust the order.

• **σadjustment²**: The standard deviation of the adjustment value, representing the flexibility of the order adjustment to ensure that adjustments are made within a reasonable range.

• **MM**: The number of arbitrage tasks requiring adjustments, representing the number of orders that need to be adjusted based on market price changes.

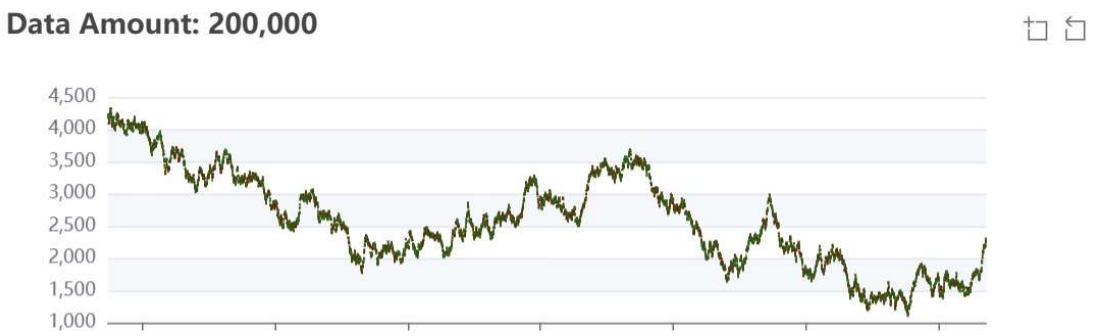
• **θj**: The weight of the j-th adjustment task, representing the priority of the task in the arbitrage adjustment strategy.

• **Rj**: The execution time of the j-th task, representing the time required to adjust the orders based on market price changes.

• **γj**: The delay factor, representing the adjustment delay caused by market fluctuations or data transmission delays.

• **tj**: The processing delay for the j-th task, representing the time between market price changes and the actual adjustment of the orders.

Formula Analysis:



• **Real-Time Price Fluctuation Monitoring and Order Adjustment**: The platform monitors market prices **Tprice(i)** and compares them with the preset price change threshold **Tthreshold**. When the market price changes beyond the threshold, the system automatically adjusts the order quantity and price based on the current market price. The Gaussian decay factor **exp(−((Tadjustment(i) − Tthreshold)² / σadjustment²))** optimizes the sensitivity of the order adjustments, ensuring that price adjustments align with market changes.

• **Order Adjustment and Profitability Guarantee**: By adjusting order prices and quantities in real-time, the platform ensures that users maintain profitability during the arbitrage process. When market prices change, the system quickly adjusts, preventing the loss of arbitrage opportunities due to price fluctuations. The execution time **Rj** and the delay factor **γj** for each adjustment task optimize the response speed of order adjustments, ensuring that users' arbitrage strategies are executed promptly.

• **Delay Optimization and Market Fluctuation Control**: Through the delay factor **γj** and processing delay **tj**, the system optimizes the response time for real-time adjustments, ensuring that the order adjustments are made promptly to mitigate risks posed by market fluctuations, thereby reducing slippage and unexecuted orders.

3. Distributed Order Splitting Strategy

Large Order Splitting

A-Force employs a distributed order splitting strategy, breaking large orders into multiple smaller ones. This approach reduces the impact of large orders on the market, minimizing the risk of market participants detecting large-scale trades. To describe how A-Force utilizes the **distributed order splitting strategy** to break large orders into multiple smaller ones, the following formula incorporates key factors such as **order splitting**, **market impact minimization**, **risk control**, and **trade concealment**. This strategy enables the platform to reduce the market impact of single large orders, decreasing the risk of other market participants noticing large trades.

$$T_{\text{split}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{order}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{impact}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{impact}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{i=1}^{M} (\theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j)))$$

Formula Explanation:

- **Tsplit**: The total transaction execution time after the order split, representing the time required to execute multiple smaller orders after splitting a large order.
- **NN**: The number of data sources, representing the number of market order book data sources monitored by the platform.
- **Wi**: The weight of the i-th data source, representing its importance in the order splitting strategy.
- **Torder(i)**: The original order data of the i-th market, representing the order information obtained by the platform.
- **Timpact(i)**: The market impact value of the i-th data source, representing the potential impact a large order has on the market.
- **Tthreshold**: The market impact threshold, indicating the value above which a large order's impact exceeds acceptable limits and triggers splitting into smaller orders.
- **σimpact²**: The standard deviation of market impact, representing the fluctuation of the market's reaction to large orders, used to control the flexibility of the splitting strategy.
- **MM**: The number of large orders that need to be split, representing how many orders require splitting into smaller ones.
- **θj**: The weight of the j-th splitting task, indicating the priority of that task within the order splitting strategy.
- **Rj**: The execution time of the j-th task, representing the time required to split and execute smaller orders.
- **γj**: The delay factor, representing the delay caused by market fluctuations or other factors during the splitting task.
- **tjt**: The processing delay of the j-th task, representing the time from order splitting to executing smaller orders.

Formula Analysis:

- **Order Splitting & Market Impact Minimization**: The platform analyzes the market impact of each order **Timpact(i)** and compares it to the set threshold **Tthreshold**. When the impact of a large order exceeds the threshold, the system splits it into multiple smaller orders to prevent a significant market disturbance. The Gaussian decay factor **exp(−(Timpact(i)−Tthreshold)² / σimpact²)** optimizes the sensitivity of the splitting strategy, ensuring that large-impact orders are split promptly.
- **Smaller Order Execution & Concealment**: By splitting large orders into multiple smaller ones, the platform effectively reduces the detection of large-scale trades in the market, thereby increasing the trade's concealment. The execution time **Rj** and delay factor **γj** of each splitting task ensure that the smaller orders are executed quickly, avoiding delays caused by the splitting process.
- **Task Priority & Execution Optimization**: To ensure the smooth execution of order splitting, the platform adjusts the task priority **θj** and optimizes the execution sequence of the splitting tasks. By controlling the delay **tjt**, the system ensures that orders are split and executed as quickly as possible, minimizing market fluctuations caused by delays.

Distributed Order Strategy

Order Splitting and Distribution

A-Force adopts a distributed order splitting strategy, where each split order is placed across different price ranges. This approach reduces the impact of market fluctuations on the orders and prevents large-scale orders from causing excessive market volatility. The strategy ensures that the market is not overly affected by the concentration of large orders, which improves the probability of a successful trade execution.

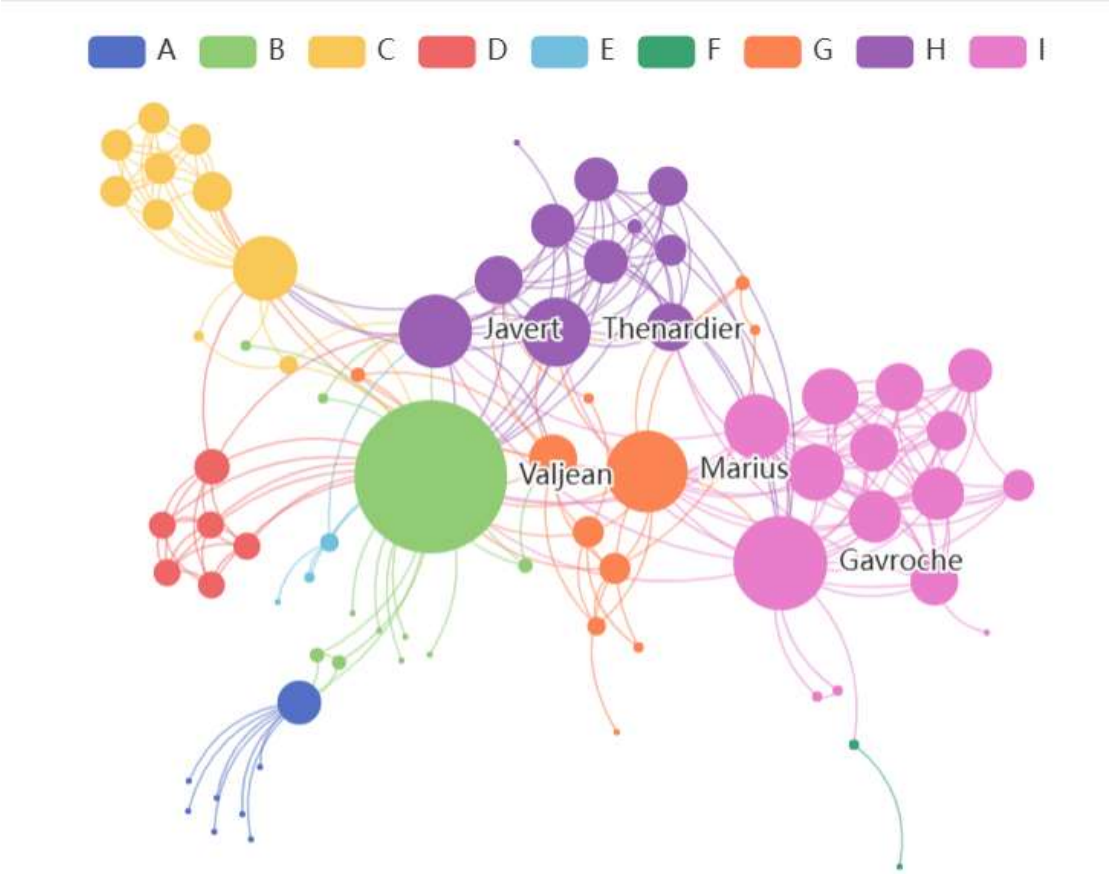
$$T_{\text{distributed}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{order}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{impact}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{impact}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} (\theta_j \cdot (R_j \cdot \exp(-\gamma_j \cdot t_j)))$$

Formula Explanation:

- **Tdistributed**: The total execution time under the distributed order strategy, representing the time needed to execute orders by distributing them across different price ranges to minimize market fluctuation impact.
- **NN**: The number of data sources, representing the different market order book data sources monitored by the platform.
- **Wi**: The weight of the i-th data source, representing the importance of that market's order book data in the distributed order strategy.
- **Torder(i)**: The original order data of the i-th market, representing the order information retrieved from the order book.
- **Timpact(i)**: The market impact value of the i-th data source, representing the potential market impact of a single order.
- **Tthreshold**: The market impact threshold, representing the maximum allowable market impact value. Orders exceeding this threshold will be split and distributed.
- **σimpact²**: The standard deviation of market impact, representing the volatility in the market's response to large orders, used to control the flexibility of the order distribution strategy.
- **MM**: The number of distributed order tasks that need to be executed, representing the number of orders that need to be placed across different price ranges.
- **θj**: The weight of the j-th distribution task, representing the priority of that task within the distributed order strategy.
- **Rj**: The execution time of the j-th task, representing the time required to execute a distributed order.
- **γj**: The delay factor, representing the delay in order execution due to market fluctuations or data transmission delays.
- **tjt**: The processing delay of the j-th task, representing the time from the generation of the order to its actual execution.

Formula Analysis:

- **Order Distribution & Market Impact Minimization**: The platform distributes each split small order across different price ranges **Torder(i)**, using a Gaussian decay factor **exp(−(Timpact(i)−Tthreshold)² / σimpact²)** to reduce the market impact of each order. When the market impact **Timpact(i)** exceeds the set threshold **Tthreshold**, the system automatically adjusts the order and distributes it across different price ranges, minimizing the effect of market volatility.



- **Price Range Optimization & Order Execution**: By optimizing the price range for

placing orders, the platform ensures that orders are successfully executed even during times of high market volatility. At the same time, it avoids the market impact caused by large, concentrated orders. The execution time **Rj** and delay factor **γj** for each distribution task ensure that orders are executed quickly and avoid missing the best execution time due to fast market movements.

- **Task Priority & Delay Control**: The system optimizes the execution order of tasks based on the task priority **θj** and processing delay **tjt**. By controlling delays, the system ensures that in times of significant market fluctuation, the most important orders are prioritized, maximizing trade execution efficiency.

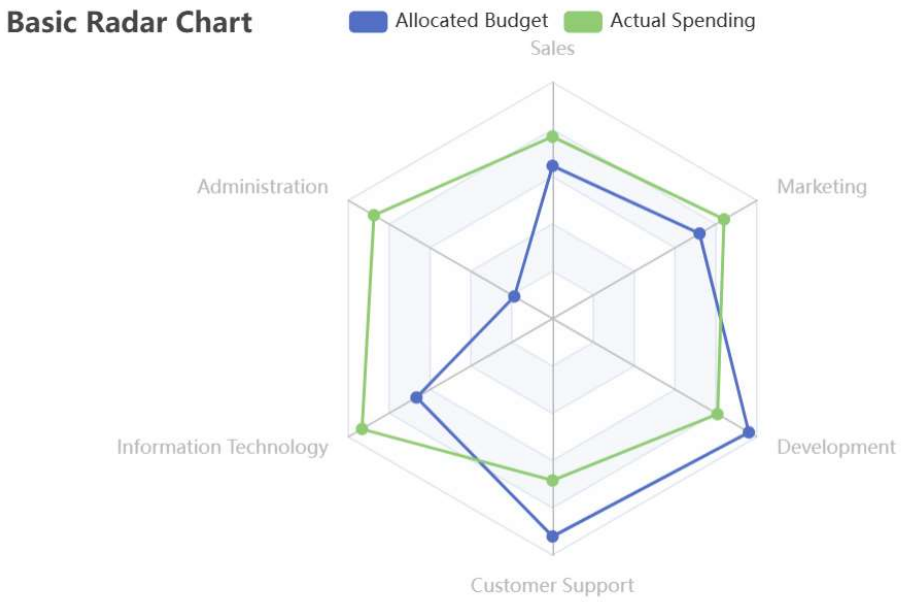
Market Impact Avoidance

Through the use of distributed order placement and order splitting strategies, A-Force effectively mitigates the impact of large single transactions on market prices, ensuring that trades can be executed in a stable market environment. This approach helps avoid significant price fluctuations caused by large transactions, ensuring successful trades in an environment with reduced volatility.

$$T_{\text{impact-avoidance}} = \frac{\sum_{i=1}^{N} W_i \cdot \left(T_{\text{order}}^{(i)} \cdot \exp\left(-\frac{(T_{\text{impact}}^{(i)} - T_{\text{threshold}})^2}{\sigma_{\text{impact}}^2} \right) \right)}{\sum_{i=1}^{N} W_i} + \sum_{j=1}^{M} \left(\theta_j \cdot \left(R_j \cdot \exp\left(-\gamma_j \cdot t_j \right) \right) \right)$$

Formula Explanation:

- **Timpact-avoidance**: The execution time after market impact avoidance, representing the time needed to reduce market impact through distributed order placement and order splitting.
- **NN**: The number of data sources, representing the number of market data sources utilized by the platform for order distribution and splitting strategies.
- **Wi**: The weight of the i-th data source, representing the importance of the market data in the impact avoidance strategy.
- **Torder(i)**: The original order data of the i-th market, representing the price and quantity of orders in that market.
- **Timpact(i)**: The market impact value of the i-th data source, representing the potential market impact caused by a single large order.
- **Tthreshold**: The market impact threshold, representing the maximum allowable market impact. Orders that exceed this threshold are automatically split and distributed.
- **σimpact²**: The standard deviation of market impact, representing the volatility in market response to large orders, used to optimize the impact minimization strategy.
- **MM**: The number of tasks requiring market impact avoidance, representing the number of orders to be split and distributed.
- **θj**: The weight of the j-th order adjustment task, representing its priority in reducing market impact.
- **Rj**: The execution time of the j-th task, representing the time needed for order splitting and execution after distribution.
- **γj**: The delay factor, representing the delay caused by market fluctuations or data transmission.
- **tjt**: The processing delay of the j-th task, representing the time from order splitting and distribution to execution.



Formula Analysis:

- **Order Distribution & Market Impact Control**: The platform avoids market impact from large orders through distributed order placement and splitting strategies. The market impact value **Timpact(i)** is compared to the threshold **Tthreshold**. When market impact is significant, orders are split and distributed across price ranges. The Gaussian decay factor **exp(-(Timpact(i)-Tthreshold)² / σimpact²)** ensures the flexibility of order splitting to optimize market impact control.
- **Market Stability & Liquidity Guarantee**: By distributing large orders across multiple price ranges, the platform prevents the market from experiencing significant volatility due to concentrated large orders. The execution time **Rj** and delay factor **γj** ensure that the split orders are executed quickly, minimizing the risk of liquidity issues and ensuring the orders are filled as expected.
- **Task Priority & Execution Optimization**: To ensure smooth order splitting, the system optimizes the execution order based on task priority **θj** and processing delay **tjt**. By controlling delays, the system ensures that critical orders are executed first during times of high market impact, minimizing the effect of market fluctuations on trade success.

Note

The explanation of some driving functions cannot be displayed in LATEXIS input format due to system limitations. Please use LATEX to calculate and input the formulas.

There may be ambiguities or inaccuracies in the translation, and the material in English shall prevail.