Bluefin Pro Margining System

Introduction

The core margining engine in this system operates under a strict framework, recognizing only three fundamental actions: **Deposit, Withdraw, or Trade** (which includes Orders, Liquidation, and Auto-Deleveraging (ADL)). This ensures a clear and deterministic approach to margin accounting and risk management.

A key distinction to understand is that Open Orders hold no direct impact on the on-chain margining system unless they result in an **Orders Trade**. This means that while Open Orders may influence a trader's potential exposure, they do not contribute to margin calculations or risk assessments until they execute as a trade.

Definitions

- Any time you see an arbitrary-looking "-1" it is a strong signal that this is the "simplified" version of an equation. The most common transform is from $(x - y)/y \Rightarrow (x/y) - 1$.
- We use `Quantity` as our preferred field to represent quantities. The only exception is the position struct, where we use `Size` . This is to keep with industry convention. The position size is the net result of many order quantities.
- We use the prefixes *total* when summing values over all positions and *cross* when summing over cross positions.
  - Isolated positions are meant to act independently and therefore do not have aggregating variables defined where we sum over many isolated positions.
- We use the prefix *total* when summing over all collateral assets as well.

Oracle Price

A VWAP (volume-weighted average price) of off-chain exchange prices, used to avoid unfair liquidations. Provided by a trusted oracle (e.g., Pyth).

Mark Price

Built off of oracle price, taking market price into consideration as well to prevent against oracle price manipulation. It's used in all margining calculations.

Cross Effective Balance

The **cross effective balance** is the total value of spot assets (USDC, ETH, BTC, SUI) in the perpetual account, adjusted by asset weights (i.e., discounts on non-USDC assets). It's calculated as:

$$\text{crossEffectiveBalance} = \sum_{a=1}^{n} \text{quantity}_a \times \text{price}_a \times \text{weight}_a$$

All deposited funds not allocated to isolated positions are available to cross positions, which share a common balance and don't hold funds individually.

Isolated positions, on the other hand, move funds out of the cross balance at creation, locking them in the position's `margin` field. These positions only support USDC as collateral.

Weights are defined in an asset config when it is being introduced as a collateral.

Note for now Bluefin only supports USDC as collateral.

Unrealized PnL

Unrealized Profit and Loss (PnL) indicates the potential profit or loss on a position if it were closed at the current mark price. It is computed as the difference between the mark price and the entry price, adjusted for position size and trade side. (*m is the subscript for each market)*

$$\text{unrealizedPnl}_m = (\text{markPrice}_m - \text{entryPrice}_m) \times \text{positionSize}_m \times \text{side}_m$$
$$\textit{where } \text{side} = +1 \textit{ for long positions, } -1 \textit{ for short positions}$$
$$\textit{and } \text{m} \textit{ is each market}$$

Pending Funding Payment

On **cross positions**, hourly funding payments are added/subtracted from the total effective balance. On **isolated positions**, they are instead added/subtracted directly from the margin allocated to the position.

If either the total effective balance (cross position) or position margin (isolated position) ever approach a 0 value such that a negative funding payment would overdraw the balance, the remaining delta is instead stored in a position level field called `pending funding payment`.

$$\text{pendingFundingPayment}_m$$

Cross Account Value

The total value of the cross account, combining the cross effective balance and unrealized PnL across all cross positions, and subtracting any pending funding payments on any cross position.

$$\text{crossAccountValue} = \text{crossEffectiveBalance} + \sum_{c=1}^{k} \text{unrealizedPnl}_c - \sum_{c=1}^{k} \text{pendingFundingPayment}_c$$

*where **c = cross position markets***

Isolated Position Value

The value of a given isolated position. Unlike the cross account value, this is exclusively the margin allocated to the isolated position along with any unrealized Pnl and pending funding payments.

$$\text{isolatedPositionValue}_i = \text{positionMargin}_i + \text{unrealizedPnl}_i - \text{pendingFundingPayment}_i$$

*where **i = isolated position market***

Initial Margin Ratio (IMR)

A constant value defined for a market denoting the minimum ratio a user must marginate a position to open it. For example, if the IMR for the BTC market is 5%, to place any order a user must have at least 5% of the **positionNotionalValue** (position size*markPrice) available to trade.

Initial Margin Required

The minimum initial margin required to open a position, calculated based on the position value and the **IMR** for that market *m*.

A **cross position** margin requirement is based off of the **mark price:**

$$\text{initialMarginReq}_c = \text{positionSize}_c \times \text{IMR}_m$$

where **c = *cross position*** on ***market m***

The total cross initial margin requirement can be represented as:

$$\text{crossInitialMarginReq} = \sum_{c=1}^{k} \text{initialMarginReq}_c$$

An **isolated position** margin requirement is based off of the **avg entry price** and therefore the Initial Margin Required is computed using the **avg entry price:**

$$\text{initialMarginReq}_i = \text{positionSize}_i \times \text{positionAvgEntryPrice}_i \times \text{IMR}_m$$

where *i* ***= isolated position*** on ***market m***

Margin Required (Cross)

The margin required to open a position and maintain it based on the **user's set leverage**. A **cross position** margin requirement is based off of the **mark price** and is therefore constantly fluctuating.

$$\text{marginReq}_c = \frac{\text{positionSize}_c \times \text{markPrice}_c}{\text{leverage}_c}$$

Aggregated over cross positions:

$$\text{crossMarginReq} = \sum_{c=1}^{k} \text{marginReq}_c$$

where **c = *cross position markets***

Margin Required (Isolated)

An **isolated position** margin requirement is based off of the **avg entry price**, and unlike cross margin requirement stays constant if not adjusted

$$\text{marginReq}_i = \frac{\text{positionSize}_i \times \text{positionAvgEntryPrice}_i}{\text{leverage}_i}$$

where *i* ***= isolated position markets***

Maintenance Margin Required

The margin required to maintain a position, calculated based on the position value and the **MMR** for that market *m*.

A **cross position** maintenance margin requirement is based off of the **mark price:**

$$\text{maintMarginReq}_c = \text{positionSize}_c \times \text{MMR}_m$$

where **c = *cross position*** on ***market m***

The total cross maintenance margin requirement can be represented as:

$$\text{crossMaintMarginReq} = \sum_{c=1}^{k} \text{maintMarginReq}_c$$

An **isolated position** maintenance margin required is based off of the **avg entry price:**

$$\text{maintMarginReq}_i = \text{positionSize}_i \times \text{positionAvgEntryPrice}_i \times \text{MMR}_m$$

where *i* ***= isolated position*** on ***market m***

Effective Leverage (Isolated)

The margin allocated to an isolated position is initially equivalent to the margin required (isolated) defined earlier. However it changes due to a user adding margin, removing margin and/or hourly funding payments.

$$\text{effectiveLeverage}_i = \frac{\text{positionSize}_i \times \text{positionAvgEntryPrice}_i}{\text{margin}_i}$$

where ***i = isolated position markets***

When a user modifies an existing position by increasing or decreasing size, the `effectiveLeverage` is kept constant to calculate the updated margin requirement.

Maintenance Margin Ratio (MMR)

A constant value defined for a market denoting the minimum ratio of margin that must be maintained in the position before liquidation. The MMR is always < than the IMR.

Maintenance Margin Available (Cross)

For a cross account, this is the amount of margin available before liquidation, calculated by subtracting the maintenance margin required from the account value.

$$\text{crossMaintMarginAvailable} = \text{crossAccountValue} - \sum_{c=1}^{k} \text{maintMarginReq}_c - \sum_{c=1}^{k} \text{unrealizedLoss}_c$$

where **c *= cross position markets***

Note that maintenance margin available is independent of the user's set leverage for each market. It assumes that the maximum amount of margin deposited into the exchange is available to marginate a position.

This also means that in cross the user's set leverage is purely a valuable tool to manage risk, by capping the size of positions they can enter and the margin ratio they enter at.

Max Withdrawal Quantity

Maximum quantity the user can withdraw from the system while keeping positions above margin requirements. Unrealized profits are not allowed to be withdraw, however, unrealized loss is discounted.

$$\text{maxWithdrawalQty} = \max(\text{crossEffectiveBalance} - \sum_{c=1}^{k} \text{unrealizedLoss}_c - \sum_{c=1}^{k} \text{pendingFundingPayment}_c - \sum_{c=1}^{k} \text{marginReq}_c, 0)$$

where **c *= cross position markets***

This formulas is designed to work for only USDC collateral

Add / Remove Margin (Isolated)

Add Margin

There is no upper bound on adding margin to an isolated position. It is theoretically possible to collateralize the position even above 100% margin ratio, (effectively even lower than 1x leverage) so long as there is sufficient margin available pull from.

$$\text{maxAdd}_i = \infty$$

where ***i = isolated position markets***

Remove Margin

When removing margin, due to having added margin earlier or the accrual of a lot of positive funding payments, the user may withdraw until th**e** margin required.

$$\text{maxRemove}_i = \max(0, \text{margin}_i - \text{marginReq}_i - \text{uLoss}_i)$$

where ***i = isolated position markets.***

Margining Engine & Health Checks

A user may have a **Cross Account** plus zero or more **Isolated Positions**:

- **Cross Account** (single entity containing all funds and multiple cross-margin positions)
- **Isolated Positions** (dedicated margin transferred from the cross account)

Each action modifies one or both of these domains (e.g., opening an isolated position reduces cross margin), and each domain involved undergoes final post-action health checks.

At a high level, the possible actions to the margin engine are simply Deposit, Withdraw and Trade (Orders, Liquidation, and ADL are all forms of Trade). For clarity of understanding, let's break them down into all the possible permutations of actions applied to each of the domains.

- **Cross Account Actions**:
  1. **Deposit** into Cross
  2. **Orders Trade** on Cross (opens, increases, or decreases cross positions)
  3. **Liquidation Trade** on Cross (user is maker side of the liquidation)
  4. **ADL Trade** on Cross (user is maker or taker side of auto-deleveraging)
  5. **Withdraw** from Cross

- **Isolated Position Actions** (implicitly also a deposit or withdraw from the Cross Account):
  1. **Orders Trade** on Isolated
     - Opening/Increasing transfers margin from Cross to the isolated-account.
     - Reducing/Closing can release unused margin back to Cross.
  2. **Liquidation Trade** on Isolated (user is maker side)
     a. May transfer margin back to Cross if any leftover
  3. **ADL Trade** on Isolated (user is maker or taker side)
  4. **Add Margin** (transfer margin from Cross to an isolated-position)
  5. **Remove Margin** (transfer margin out of the isolated-position back to Cross)
  6. **Increase Leverage** (increase the user-set leverage. This may release margin back to the cross account)

2. Post-Action Health Checks

After **any** action is performed, the system examines both the **initial state** (account value, positions, margin requirements) and the **final state** produced by the action. We apply consistent checks to the **Cross Account and, if relevant, an Isolated Position.**

- **Initial State**: Reflects the domain's balances/positions before the action. This determines whether the chosen action is permissible (e.g., if the account is in a Margin Call, it can only do certain things).
- **Final State**: Reflects the domain's new balances/positions after the action has been applied. We then validate the resulting health. If it ends in a state not compatible with the action taken, the transaction reverts.

Note that the system utilizes post-action health checks to enforce solvency, but the system will also avoid meaningless or exploitative liquidations from the start.

Cases

1. **Case 1: Healthy**
   - **Final State Condition:**

     $$\text{crossAccountValue} - \text{crossInitialMarginReq} \geq 0$$

     When an isolated position is involved, also:

     $$\text{isolatedPositionValue}_i - \text{initialMarginRequired}_i \geq 0$$

   - **Allowed Actions (if end in this case):** All actions (deposits, withdrawals, trades, margin adjustments, liquidations, ADL) are permitted.
     - Note: liquidation is valid since it may have brought the account into a healthy state. The system separately uses **Preemptive Blocking of Invalid Actions** to prevent a liquidation on an account that is clearly healthy at the time of order placement.
2. **Case 2: Margin Call**
   - **Final State Condition:**

     $$\begin{gathered}\text{crossAccountValue} - \text{crossInitialMarginRequired} < 0 \\ \text{and} \\ \text{crossAccountValue} - \text{crossMaintenanceMarginRequired} \geq 0\end{gathered}$$

     When an isolated position is involved:

     $$\begin{gathered}\text{isolatedPositionValue}_i - \text{initialMarginRequired}_i < 0 \\ \text{and} \\ \text{isolatedPositionValue}_i - \text{maintenanceMarginRequired}_i \geq 0\end{gathered}$$

   - **Allowed Actions:**
     - **Deposit** (or transfer margin in) to improve the account value.
     - **Orders Trade reducing position** (must not flip sides; only reduce size of the current direction).
   - **Disallowed Actions:**
     - Withdrawing or removing margin.
     - Orders Trade Increasing position size, opening a new position, flipping the position.
3. **Case 3: Below Maintenance**
   - **Final State Condition:**

     $$\begin{gathered}\text{crossAccountValue} - \text{crossMaintMarginReq} < 0 \\ \text{and} \\ \text{crossAccountValue} \geq 0\end{gathered}$$

     When an isolated position is involved:

     $$\begin{gathered}\text{isolatedPositionValue}_i - \text{maintenanceMarginRequired}_i < 0 \\ \text{and} \\ \text{isolatedPositionValue}_i \geq 0\end{gathered}$$

- **Allowed Actions:**
  - **Deposit** margin (or transfer in) to improve account value.
  - **Act as maker** in a liquidation or ADL trade
- **Disallowed Actions:**
  - All orders trades (even those that reduce or close the position).
  - Withdrawing or removing margin.

4. **Case 4: Bankruptcy**
   - **Final State Condition:**

$$\text{crossAccountValue} < 0$$

When an isolated position is involved:

$$\text{isolatedPositionValue}_i < 0$$

- **Allowed Actions:**
  - **Deposit** margin (or transfer in) to improve account value.
  - **Act as maker** in a liquidation or ADL trade
- **Disallowed Actions:**
  - All orders trades (even those that reduce or close the position).
  - Withdrawing or removing margin.

3. Preemptive Blocking of Invalid Actions

In addition to post-action checks, the system blocks certain requests based on the **initial** health classification. This saves compute and prevents wasteful transactions. The key principle is to reject actions that are guaranteed to end in a revert:

Actions Not Caught by Post-Checks

- **Liquidation on a Margin Call Maker Domain or Better**: If the Maker Cross Account or Isolated Position is initially in **Case 2 (Margin Call) or better**, a liquidation request is blocked.
- **ADL on a Below Maintenance Maker Domain**: If the Maker Cross Account or Isolated Position is initially in **Case 3 (Below Maintenance) or better**, an ADL request is blocked.
  - Note that ADL on taker account in healthy state are still allowed due to system-wide reasons
- **Orders Trades in Below Maintenance or Worse**: If the domain is initially **Case 3 or lower**, the user cannot participate in any Orders Trade. The system identifies this before applying the transaction to avoid unnecessary calculations.
  - This additionally catches the case where a **Bankrupt Account** places an **Orders Trade** that closes position and takes Account Value to 0, passing post-checks (healthy state) but allowing bad debt to enter the system.
- **Trade with RealizedPnl greater than Effective Balance:** It is possible that an account is healthy initially, and performs any trade with such large realized loss that their balance is exceeded but end up in a final healthy state since Cross Account Value ≥ 0. This must be explicitly checked for and prevented.
  - Note that the only exception here is a bankruptcy liquidation trade. In this scenario, there is known bad debt in the system that the bankruptcy liquidator account (taker in this trade) must cover.
- **Liquidation Trade must be liquidating highest unrealized profit position in maker account:** Any account being liquidated, i.e maker in a liquidation trade, must have its highest profit position liquidated first to ensure all profit is realized before approaching loss making positions. This helps cover edge cases where:
  - On a below maintenance account (case 3), unrealized loss on one position can be greater than account balance (account value is greater than 0 due to unrealized profit on another position). By liquidating the highest profit position first we ensure we never run into this case
  - On a below bankruptcy account (case 4), this ensures that all profit is realized before bankruptcy liquidators cover bad debt from the loss making positions. Not following this would result in the bankruptcy liquidator covering all bad debt on loss making position first and actually taking account to a healthy state and make their profitable position unliquidatable.

Actions Already Rejected by Post-Checks

- **Withdrawals in Margin Call or Worse**: If the domain is initially **Case 2 or lower**, the user cannot withdraw or remove margin. The system identifies this before applying the transaction to avoid unnecessary calculations.

Trade

At a protocol level, there are three core trade types that can occur on the on-chain margining system:

1. **Orders Trade**
2. **Liquidation Trade**
3. **ADL (Auto-Deleveraging) Trade** ***(out of scope currently)***

All three share a common set of inputs (such as the account(s) involved, market/perpetual details, and the quantity or size traded) and ultimately produce a corresponding event on-chain once executed. Below is an overview of each trade type, along with their parameters, and—importantly—the **events and field computations** detailing how positions and asset balances are updated.

Core Trade Parameters

Each of the three trade types takes in a similar set of parameters:

- **Market/Perpetual**: Identifies which perpetual market is involved (BTC-PERP, ETH-PERP, etc.).
- **Side/Direction**: Whether the account or position is going **long** (buying) or **short** (selling).
- **Quantity**: The size of the trade in contract units, subject to each market's `step_size`, `min_trade_qty`, and `max_trade_qty`.
- **Price**: The execution price or reference price used to settle the trade. This is either a fill price (in an Orders Trade) or a special discount/mark price (in Liquidation and ADL).
- **Position Type**: Whether the trade is on a **Cross** or **Isolated** position. This choice affects how margin is allocated (and is subject to whether the market allows cross positions or is limited to isolated only).
- **Leverage** (Isolated Only): The leverage factor for isolated positions if the trade is opening or increasing the position size. Cross positions store a `0` leverage, as all margin in the account is effectively used for collateral.

Orders Trade

An **Orders Trade** occurs when two orders (a Maker and a Taker) are matched by the matching engine.

1. **Parameters Specific to an Orders Trade**
   - **Maker Order**: Includes maker's address, price, quantity, side, and any expiration data.
   - **Taker Order**: Same fields as the maker but on the opposite side.
   - **Fill Quantity**: How much of each order is matched.
   - **Fill Price**: The final execution price.
   - **Timestamp/Expiry**: Orders may have an off-chain expiry and are validated to ensure they are not stale.
   - **Signatures**: Both orders are signed by their owners. The on-chain engine verifies the maker and taker have permission for those accounts.
2. **Events and Field Computations**

   Upon execution, a **TradeExecuted** event is emitted if the final post-action checks pass. Here's a breakdown of how state updated and final values in this event were calculated:

   - **market**: The perpetual symbol (e.g., BTC-PERP).
   - **maker_hash, taker_hash**: Unique identifiers referencing each matched order.
   - **fill_quantity, fill_price**: The final quantity and price at which the trade was settled.
   - **maker_position, taker_position**: Updated **Position** objects for each participant. These positions contain:
     - **size**: Adjusted by adding or subtracting `fill_quantity`. If the trade opens or increases a position in the same direction, `size` grows. If it offsets an existing position in the opposite direction, `size` might shrink or flip sides.
     - **average_entry_price**: Recalculated by weighting the old position with the new trade. Formally, when positions grow on the same side:

       ```
       newAvgEntryPrice =
         ( (oldAvgEntryPrice * oldSize) + (fillPrice  * fillQty) ) /
         (oldSize + fillQty)
       ```

       If the position flips sides partially:

       - A **realized PnL** is computed for the 'closing' portion:

         `realized_PnL=(fill_price - oldAvgEntryPrice) * closing_qty * side` (where side is +1 if a long is being closed, or -1 if a short is).
       - Any leftover portion (`leftover_qty = | old_position_size - fillQty |`) is now the new position `size` with an `angEntryPrice` equivalent to the `fillPrice`
     - **is_long**: May remain the same if the trade is in the same direction, or flip if an opposite-direction fill fully closes the prior side and reopens in the new side.
   - **margin** (Isolated Only): Updated so that the user has exactly `margin = positionSize * average_entry_price / effective_leverage` locked in that isolated position. Additional amounts are pulled from or returned to the cross account if needed.
     - **leverage** (Isolated Only): Remains zero for cross positions, or stored for isolated.
   - **maker_fee, taker_fee**: Computed by `fee_percent * (fill_price * fill_quantity)`. Maker fees are typically lower. In a **Cross** margin scenario, the fee is deducted directly from the user's cross deposit balance. In an **Isolated** margin scenario, the fee is first taken out of the isolated position's margin; if that margin does not fully cover the fee or if the position is closing, additional amounts are pulled in (or leftover amounts are returned) from the cross account. The final net fees then get credited to the Fee Pool.
   - **maker_assets, taker_assets**: Updated **DepositedAsset** vectors representing their cross accounts. In a cross scenario, **fees or realized PnL are added/subtracted from these deposit balances**. In an isolated scenario, leftover margin from partial closes or additional margin needed for expansions is also reflected.
   - **taker_side**: Whether the taker was `LONG` or `SHORT` on the fill.
   - **sequence_hash, sequence_number**: On-chain replay protection and ordering.

Liquidation Trade

A **Liquidation Trade** is triggered when an account or position (cross or isolated) falls below its maintenance margin requirement. A whitelisted **liquidator** steps in, buying or selling the losing position at a discount.

1. **Definition**

   Liquidation involves two parties:

   - **Liquidatee**: The user below maintenance. This is the **maker.**
   - **Liquidator**: The party taking over all or part of the position, typically at a discounted liquidation purchase price (`liq_purchase_price`). This is the **taker.**

   And utilizes the **Liquidator Purchase Price** formula to offer a position purchase discount to mark price.

   For **standard liquidations (case 3),** this is:

$$\text{liqPurchasePrice}_{\text{standard}} = \begin{cases} \max\left(\text{markPrice}_m \times \left(1 - \frac{\text{MMR}_m}{2}\right), \text{bankruptcyPrice}_m\right) & \text{if Long} \\ \min\left(\text{markPrice}_m \times \left(1 + \frac{\text{MMR}_m}{2}\right), \text{bankruptcyPrice}_m\right) & \text{if Short} \end{cases}$$

   where

$$\text{bankruptcyPrice}_m = \text{markPrice}_m - (\text{positionSide}_m * \frac{\text{crossAccountValue}}{\text{positionQuantity}_m})$$

   For **bankrupt liquidations (case 4)**, this is simply the markPrice:

$$\text{liqPurchasePrice}_{\text{bankruptcy}} = \text{markPrice}$$

2. **Parameters Specific to a Liquidation Trade**

   - **market**: Which perpetual is liquidated.
   - **hash**: Reference to the signed liquidation payload or unique ID.
   - **liquidatee_position, liquidator_position**: The resulting positions:
     - **size**:
       - **Liquidatee**: `size_new = size_old - quantity`. If it reaches zero, fully closed.
       - **Liquidator**: `size_new = size_old + quantity`. (Or newly opened if none existed.)
     - **average_entry_price:**
       - **Liquidatee** (settles at `liq_purchase_price`):
         - If partially closed while remaining in the same side, or completely closed,
           - **realized PnL** is `(liq_purchase_price - oldAvgEntryPrice) * closed_qty * side`, and the leftover portion keeps oldAvgEntryPrice.
             - Note that in a bankruptcy liquidation, the `liq_purchase_price` is the `mark_price` , and if this realizedPnl is greater than the account balance, the difference is stored as **bad debt** to be removed from the liquidator's realizedPnl.
       - **Liquidator** (uses `mark_price`):
         - The liquidator's `fill price` is effectively `mark_price`.
         - The difference `(mark_price - liq_purchase_price) * quantity` is computed as a **liquidation premium** into the liquidator's **realized PnL**, minus the portion sent to the insurance pool.
         - Note that in **bankruptcy liquidations**, `liq_purchase_price` is the `mark_price` so this premium value will always be 0. Instead in these scenarios, any non 0 **bad debt** from the **liquidatee** will be removed from the liquidator's **realized PnL.**

           Here are the three main scenarios for a liquidators new `avgEntryPrice` and effective `realizedPnl` after a **liquidation**:

           1. **No Prior Position:** The liquidator creates a new position at `average_entry_price = mark_price`, so no immediate PnL from that portion.
              a. The **premium** `(mark_price - liq_purchase_price) * quantity` is added to **realizedPnl**, with a fraction removed for insurance.
                 i. The final amount realized for the liquidator becomes `premium * (1 - insurance_fund_premium)`
              b. Any **bad debt**, from the **liquidatee** is removed from the **liquidator's realizedPnl**
           2. **Adding to an Existing Same-Side Position**: No immediate PnL from expanding. The new average entry price follows the standard formula from earlier `((oldAEP * oldSize) + (mark_price * fillQty)) / (oldSize + fillQty)` .
              a. Again any **premium** is added, and any **bad debt** is removed
           3. **Opposite-Side (Flipping)**: The liquidator closes part or all of their old position at mark_price, realizing `(mark_price - oldAvgEntryPrice) * closed_portion * side`. Any leftover changes direction or remains, recalculating average_entry_price if continuing.
              a. Again any **premium** is added, and any **bad debt** is removed
     - **margin (Isolated Only)**

       The liquidatee's isolated margin covers realized losses on the closed portion. If the position is fully closed, any leftover margin returns to the cross account. Meanwhile, if the liquidator opens or extends their own isolated position, margin is pulled from (or returned to) their cross account as needed to meet the position's collateral requirement.
     - **is_long**:
       - **Liquidatee:** Will remain the same as the original position given liquidation can only reduce or close a position for the liquidatee, not flip.

        - **Liquidator:** Depending on original position, may remain the same if the position takeover quantity is in the same direction, or reduce and flip if in opposite-direction
    - **leverage** (Isolated Only): Remains zero for cross positions, or stored for isolated.
- **insurance_pool_premium_portion**

  A fraction of the liquidator's realized gain (from `(mark_price - liq_purchase_price) * quantity * insurance_fund_precent`) that is diverted to the insurance pool if not bankrupt.