

# CBMM Pool: A Constant Burn Market Maker

@mmatdev  
Access Protocol  
vladislav@accessprotocol.co

November 25, 2025

## Abstract

This whitepaper describes the CBMM (Constant Burn Market Maker) pool, an improved version of the standard constant product market maker (CPMM) that introduces a token burn mechanism alongside with automated buybacks resulting in a positive impact on the token's price and more continuous demand. Moreover, this provides a mechanism for off-chain actions to have a direct impact onto the beans price.

## 1 Introduction

Constant function market makers (CFMMs) have become a foundational primitive for decentralized exchanges [1]. Variants include constant product market makers (CPMMs) such as Uniswap v1 [2], concentrated liquidity market makers (CLMMs) like Uniswap v3 [3], and dynamic automated market makers (DAMMs) [4]. Bonding curves [5], popularized by platforms like Pump.fun, represent another approach where token price is determined by a deterministic curve based on supply.

Existing mechanisms provide no built-in way to translate off-chain activity into on-chain price impact. The only mechanism available to drive positive price action is manual buy-and-burn operations, where participants purchase tokens and burn them to reduce supply. This requires coordination, creates friction, and does not automatically link rewards to verifiable off-chain behavior.

We present the Constant Burn Market Maker (CBMM), a mechanism that enables tying off-chain events to underlying asset supply reduction. CBMM utilizes a virtual token reserve that supports controlled burns without violating pool invariant constraints and enables creating one-sided launch pools. These burns directly increase price proportionally to the amount of beans held outside the pool and can be tied to off-chain events. Moreover, to compensate for the virtual reserve reduction that accompanies burns, we implement Continuous Conditional Buybacks (CCB), which route a portion of trading fees into the real token reserve, effectively increasing collateralization and increasing price further.

The remainder of this paper is structured as follows. Section 2 develops the mathematical model, derives trading and burning formulas, and analyzes price impact and mitigation of possible attacks. Section 3 describes the on-chain program implementation considerations, including safety mechanisms and configuration considerations.

## 2 Mathematical Model

In this section we will describe the mathematical model of the CBMM pool. Let us first define the key terms used throughout this section. Let there be a CPMM pool with a real quote token reserve  $A$  (backed by actual assets), real base token reserve  $B$  and virtual quote token reserve  $V$  (not backed by assets, used to set the initial price and liquidity). The pool trading mechanics are

defined by the constant product invariant  $k = (A + V)B$ . We say that this pool is a **CBMM pool** if it implements the base token burn functionality tied to the virtual reserve reduction as described in subsection 2.2.

The CBMM pool is said to be **insolvent** if its quote token reserves are insufficient to accommodate the sale of all outstanding base tokens; otherwise it is said to be **solvent**. The goal is to always keep the CBMM pool solvent which can be achieved by adjusting the virtual reserve after each burn.

In this whitepaper the CBMM is designed as a base token rollout mechanism, so we do not need to account for any existing supply outside the pool. Moreover, we assume that the initial real quote token reserve  $A$  is zero and, therefore, the virtual quote token reserve  $V$  sets the initial price of the base token.

## 2.1 Trading

Buys and sells in CBMM follow the mechanics of the standard CPMM with a quote token virtual reserve; we include them here for completeness. The pre-trade (buy or sell) values are

$$A = A_0, \quad B = B_0, \quad V = V, \quad k = (A_0 + V)B_0.$$

During trading, the invariant  $k$  and the virtual reserve  $V$  do not change. The amount of base tokens  $b$  received by the user when spending  $\Delta A$  quote tokens follows from

$$k = (A_0 + \Delta A + V)(B_0 - b), \quad (1)$$

which gives:

$$b = B_0 - \frac{k}{A_0 + \Delta A + V}. \quad (2)$$

Similarly, the amount of quote tokens  $a$  received by the user when spending  $\Delta B$  base tokens follows from

$$k = (A_0 - a + V)(B_0 + \Delta B), \quad (3)$$

which gives:

$$a = A_0 + V - \frac{k}{B_0 + \Delta B}. \quad (4)$$

The price of base tokens  $P$  (quote tokens per base token) is derived from the marginal rate of exchange. Starting with the invariant  $k = (A + V)B$  and holding  $k$  constant, we differentiate with respect to  $B$ :

$$\frac{d}{dB}[(A + V)B] = \frac{dA}{dB} \cdot B + (A + V) = 0, \quad (5)$$

which yields  $\frac{dA}{dB} = -(A + V)/B$ . The price is the negative of this derivative:

$$P = -\frac{dA}{dB} = \frac{A + V}{B}. \quad (6)$$

## 2.2 Base Token Reserve Burning

This section describes the necessary conditions for the CBMM pool to remain solvent after a base token reserve reduction.

Let the initial state of the pool be

$$A_0 = 0, \quad B_0 = T, \quad V_0 = V, \quad k_0 = (0 + V)T = VT.$$

Assume, without loss of generality, that trading occurs before the burn, lowering the base token reserve by  $x \geq 0$ . The post-trade values are

$$A_1 = \frac{Vx}{T-x}, \quad B_1 = T-x, \quad V_1 = V, \quad k_1 = VT.$$

Now, if we burn  $y$  base tokens with  $0 \leq y < T-x$ , the post-burn state is

$$A_2 = \frac{Vx}{T-x}, \quad B_2 = T-x-y, \quad V_2 \text{ to be determined}, \quad k_2 = (A_2 + V_2)B_2.$$

Let's find the condition for the virtual reserve  $V_2$  to ensure the pool is solvent. This means that if everyone sells their base tokens back to the pool, there must be sufficient quote tokens to satisfy the sale. The post-sale state must satisfy

$$A_3 \geq 0, \quad B_3 = T-y, \quad V_3 = V_2, \quad k_3 = (A_3 + V_3)B_3.$$

From  $k_2 = k_3$  and  $k_3 \geq V_3 B_3$ , we obtain a bound on  $V_2$  that ensures the pool is solvent:

$$V_2 \leq \frac{V(B_1 - y)}{B_1} = \frac{V(T-x-y)}{T-x}. \quad (7)$$

Thus, after every burn, the virtual reserve must be adjusted downward to ensure solvency. A natural choice is to set  $V_2$  to the maximum value satisfying the bound, which minimizes the price impact of the virtual reserve reduction.

### 2.2.1 Price impact of the burn

Denote the price before burn as  $P_1 = \frac{A_1 + V_1}{B_1}$  and the price after burn as  $P_2 = \frac{A_2 + V_2}{B_2}$ . Substituting the values from the previous section, where  $B_1 = T-x$  and  $V_2 = \frac{V(T-x-y)}{T-x}$ , we obtain

$$P_1 = \frac{A_1 + V_1}{B_1} = \frac{\frac{Vx}{T-x} + V}{T-x} = \frac{VT}{(T-x)^2},$$

$$P_2 = \frac{A_2 + V_2}{B_2} = \frac{\frac{Vx}{T-x} + \frac{V(T-x-y)}{T-x}}{T-x-y} = \frac{V(T-y)}{(T-x)(T-x-y)}.$$

The relative price impact of the burn is then:

$$\frac{P_2 - P_1}{P_1} = \frac{xy}{T(T-x-y)}. \quad (8)$$

This formula shows that the relative price impact is proportional to the product  $xy$  of base tokens held outside the pool  $x$  and base tokens burned  $y$ , divided by  $T(T-x-y)$ . The impact increases with  $x$ , meaning burns have greater price impact when more base tokens have been purchased. If  $x = 0$  (no base tokens purchased), the burn has no price impact, as expected.

The solvency adjustment  $V_2 \leq V(T-x-y)/(T-x)$  ensures that  $V_2 < V$  for any  $y > 0$ . This reduction in the virtual reserve can affect the worst-case exit price. The worst-case exit price refers to the marginal price when all outstanding base tokens (post-burn) are sold back to the pool, driving the real quote token reserve  $A$  toward zero. At this limit, the price approaches  $V_2/(T-y)$ . When  $x > 0$  and  $y > 0$ , this worst-case exit price can be below the initial anchor price  $V/T$  set by the starting virtual reserve  $V$ . However, when  $x = 0$  (no base tokens purchased before the burn), the worst-case exit price equals the initial price, as  $V_2 = V(T-y)/T$  and  $V_2/(T-y) = V/T$ .

Operationally, burns reduce the base token reserve  $B$ , tightening depth and available liquidity at the new state.

### 2.3 Token Reserve Top-up

Burns reduce the base token reserve  $B$  and force a downward adjustment of the virtual reserve  $V$ . This causes a liquidity reduction and price raise offset by the price curve getting steeper and the initial price being reduced. This unwanted side effect can be undone by adding quote tokens to the real quote token reserve  $A$  which we call **top-up**. This subsection derives the exact top-up amount needed to restore the base token starting price without changing the logic introduced earlier.

Let  $T_i$  denote the total base token supply after  $i$  operations. The pre-trade state is

$$A_0 = 0, \quad B_0 = T_0, \quad V_0 = V, \quad T_0 = T, \quad k_0 = (A_0 + V_0)B_0 = V_0T_0.$$

After trades that extract  $x$  base tokens from the pool, the state is

$$A_1 = A_0 + \frac{(A_0 + V_0)x}{B_0 - x} = \frac{Vx}{T - x}, \quad B_1 = B_0 - x, \quad V_1 = V_0, \quad T_1 = T_0, \quad k_1 = k_0.$$

Next, burn  $y$  base tokens ( $0 \leq y < B_1$ ). Let  $V_2$  denote the virtual reserve enforced by the solvency condition in Section 2.2. The post-burn state is

$$A_2 = A_1, \quad B_2 = B_1 - y, \quad V_2 \leq V_1, \quad T_2 = T_1 - y, \quad k_2 = (A_2 + V_2)B_2.$$

Our target is to find a virtual reserve  $V_{\text{opt}}$  and a real quote token amount  $A_{\text{opt}}$  that keep the worst-case exit price – the price when all outstanding base tokens are sold back – at the original value  $P_0 = V_0/T_0$ . Let's denote the needed top-up amount to achieve the optimal state as  $M$ . If we manage to achieve this top-up, the pool state will be

$$A_3 = A_2 + M = A_{\text{opt}}, \quad B_3 = B_2, \quad V_3 = V_{\text{opt}}, \quad T_3 = T_2, \quad k_3 = k_{\text{opt}} = (A_{\text{opt}} + V_{\text{opt}})B_2.$$

If everyone exits after the top-up, the pool holds  $B_3 = T_2$  base tokens and zero real quote tokens, so the price becomes  $P_3 = V_{\text{opt}}/T_2$ . Enforcing  $P_3 = P_0$  yields

$$\frac{V_0}{T_0} = \frac{V_{\text{opt}}}{T_2} \implies V_{\text{opt}} = \frac{T_2}{T_0}V_0 = \frac{T - y}{T}V_0. \quad (9)$$

With this  $V_{\text{opt}}$  value, the invariant that corresponds to the desired price profile is

$$k_{\text{opt}} = (0 + V_{\text{opt}})T_2 = V_{\text{opt}}T_2 = (A_{\text{opt}} + V_{\text{opt}})B_2. \quad (10)$$

This implies the target real quote token reserve

$$A_{\text{opt}} = \frac{k_{\text{opt}}}{B_2} - V_{\text{opt}} = \frac{T_1 - y}{T_1}V_1 \frac{T_1 - B_1}{B_1 - y}. \quad (11)$$

The second equality follows by substituting  $k_{\text{opt}} = V_{\text{opt}}T_2$ ,  $B_2 = B_1 - y$ ,  $T_2 = T_1 - y$ ,  $T_1 = T$ , and  $T_1 - B_1 = x$  into the first expression. The required quote token top-up amount  $M$  is

$$M = A_{\text{opt}} - A_2 = \frac{T - y}{T}V \frac{T - (B_0 - x)}{(B_0 - x) - y} - \left( A_0 + \frac{(A_0 + V)x}{B_0 - x} \right). \quad (12)$$

Or in other terms for simple on-chain calculation:

$$M = \frac{A_1^2}{(A_1 + V_1)(B_1 - y)}. \quad (13)$$

### 2.3.1 Trading impact on required top-up amount

If the top-up does not happen atomically with the burn and is delayed, some trading may occur in the meantime. This impacts the required top-up amount. Suppose that after the burn, instead of applying the top-up immediately, trades shift the base token reserve from  $B_2$  to  $B' = B_2 + \Delta B$  with  $-B_2 < \Delta B \leq T_2 - B_2$ . Positive  $\Delta B$  corresponds to net sells back into the pool, while negative  $\Delta B$  captures net buys.

The target optimal reserve  $A'_{\text{opt}}$  that preserves the desired price profile at the new base token reserve  $B'$  is tied to  $k_{\text{opt}}$  and  $V_{\text{opt}}$  which remain unchanged by trading:

$$A'_{\text{opt}} = \frac{k_{\text{opt}}}{B'} - V_{\text{opt}}. \quad (14)$$

Meanwhile, the actual quote token reserve  $A'_{\text{real}}$  at base token reserve  $B'$  induced by the current invariant  $k_2 = (A_2 + V_2)B_2$  becomes

$$A'_{\text{real}} = \frac{k_2}{B'} - V_2. \quad (15)$$

The updated top-up requirement after the trades is therefore

$$M' = A'_{\text{opt}} - A'_{\text{real}} = \frac{k_{\text{opt}} - k_2}{B'} + (V_2 - V_{\text{opt}}). \quad (16)$$

Buy ( $\Delta B < 0$ ) shrink  $B'$ , amplifying the first term and increasing  $M'$ ; sell ( $\Delta B > 0$ ) expand  $B'$  and dampen the same term, but the additive offset  $(V_2 - V_{\text{opt}}) < 0$  keeps the gap positive. At the boundary case  $\Delta B = T_2 - B_2$  the expression is undefined. In that state no base tokens are left outside the pool, so the pool price can be without any unwanted side effect reset to the original initial price by setting  $V = V_{\text{opt}}$  without any real quote token top-up.

### 2.3.2 Partial Top-ups

Suppose only  $M' < M$  quote tokens are available for the top-up immediately after the burn. Injecting  $M'$  raises the real quote token reserve to  $A_{\text{new}} = A_2 + M'$  while the base token reserve stays at  $B_2$ . The invariant is therefore

$$k_{\text{new}} = (A_{\text{new}} + V_{\text{new}})B_2, \quad (17)$$

where  $V_{\text{new}}$  is the virtual reserve after the partial top-up and is unknown. We do not target a specific  $k_{\text{new}}$ ; instead we push the post-injection price as high as pool solvency requirement allows. When every outstanding base token ( $T_2 - B_2$  in total) is sold back, the goal is for the pool to reach  $A = 0$ . This corresponds to the invariant

$$k_{\text{new}} = V_{\text{new}}T_2. \quad (18)$$

Equating both expressions for  $k_{\text{new}}$  yields the reserve closest to  $V_{\text{opt}}$  under a partial top-up that keeps the pool solvent:

$$V_{\text{new}} = \frac{A_{\text{new}}B_2}{T_2 - B_2}. \quad (19)$$

When  $T_2 > B_2$ , this choice keeps the pool solvent and maximizes the achievable price lift given the available collateral. Any subsequent injection simply repeats the calculation with updated  $A_{\text{new}}$  and moves  $V_{\text{new}}$  closer to  $V_{\text{opt}}$  until the full top-up is completed.

As already discussed in Section 2.3.1, if trades drive  $T_2 = B_2$  (no base tokens outside the pool),  $V_{\text{new}}$  can be reset directly to  $V_{\text{opt}}$  because no outstanding holders remain.

## 2.4 Value Extraction Considerations and Mitigation Strategies

A critical concern for CBMM pools is that adversaries may attempt to extract value from the real quote token reserve  $A$  through strategic manipulation of burns. An attacker can execute a buy–burn–sell loop: purchase base tokens, trigger a burn (which reduces the virtual reserve  $V$ ), and then sell the base tokens back, potentially realizing a net profit at the expense of the pool’s real collateral. This section quantifies the exact profit that an adversary can realize through such attacks and outlines mitigation strategies. We first analyze the basic attack without fees, then show how symmetric quote-token fees can suppress the attack, and finally demonstrate that when top-ups are involved, additional constraints such as per-burn caps are necessary to prevent profitable exploitation.

### 2.4.1 Attack model and payoff derivation

Consider a pool in state  $(A, B, V)$  with invariant  $k = (A + V)B$  and some base tokens in circulation. Total base token supply is  $T > B$ . An adversary executes the following steps:

1. **Buy  $x$  base tokens.** The adversary buys  $x$  base tokens paying  $A_{\text{in}}$  quote tokens. According to the standard CBMM trading logic

$$A_{\text{in}} = (A + V) \frac{x}{B - x}, \quad 0 < x < B. \quad (20)$$

The post-buy reserves are  $A_1 = A + A_{\text{in}}$  and  $B_1 = B - x$ .

2. **Trigger a burn of  $y$  base tokens** from the pool (with  $0 < y < B - x$ ), enforcing solvency by reducing the virtual reserve to

$$V_2 = \frac{V(B - x - y)}{B - x}, \quad (21)$$

so that the new invariant remains  $k_2 = (A_1 + V_2)(B - x - y)$ .

3. **Sell the  $x$  base tokens back.** Using the sell formula from Section 2, the attacker receives

$$A_{\text{out}} = \frac{(A_1 + V_2)x}{B - y} \quad (22)$$

quote tokens while the pool returns to base token reserve  $B - y$ .

Combining Equations (20) to (22) yields a closed-form expression for the net quote token profit:

$$\Pi(x, y) = A_{\text{out}} - A_{\text{in}} = \frac{Axy}{(B - x)(B - y)}. \quad (23)$$

The profit formula Equation (23) reveals several properties of the attack. Most critically, whenever  $A > 0$ , the profit  $\Pi(x, y) \geq 0$  for all admissible  $x$  and  $y$ , meaning the attack is always profitable (or at least break-even) whenever there is any real reserve. This is the fundamental vulnerability: any real quote token reserve can be extracted through a buy–burn–sell loop. The extractable value is zero only when  $A = 0$ , meaning no real reserve implies no profit opportunity. Profit grows jointly with both  $x$  and  $y$ , reaching its maximum as both approach  $B$ . In the limit  $\lim_{x, y \rightarrow B} \Pi(x, y) = A$ , an adversary buying and burning nearly all tokens from the pool reserve can extract almost the entire real reserve. The attack strictly consumes existing collateral: after selling back, the pool retains only  $A - \Pi$  quote tokens. This implies that any mechanism increasing  $A$  must implement a compensation logic to prevent the adversary from profiting.

### 2.4.2 Attack model with symmetric base-token fees

To actively suppress the buy–burn–sell loop, we impose an  $N\%$  fee on every trade, collected entirely in the quote token. This fee penalizes extraction attempts in the following way. Define the fee multiplier

$$q = \frac{100}{100 - N} \quad (24)$$

All fees are skimmed from the trader’s quote token transfers: buy-side fees are removed before quote tokens enter the pool, and sell-side fees are shaved off before proceeds hit the attacker’s wallet.

**Step 1: buy  $x$  base tokens.** To withdraw  $x$  base tokens ( $0 < x < B$ ), the trader must send  $S_{\text{buy}}$  quote tokens so that the post-fee deposit equals the CBMM requirement. Solving

$$(A + V)B = (A + S_{\text{buy}}/q + V)(B - x) \quad (25)$$

gives

$$S_{\text{buy}} = q \frac{(A + V)x}{B - x}. \quad (26)$$

The pool itself still sees  $A_1 = A + \frac{(A+V)x}{B-x}$  and  $B_1 = B - x$ , identical to the no-fee path, while the attacker cost is increased by the fees.

**Step 2: burn  $y$  base tokens.** Burning  $y$  base tokens ( $0 < y < B_1$ ) forces the same solvency adjustment as before:

$$A_2 = A_1, \quad B_2 = B - x - y, \quad V_2 = \frac{V(B - x - y)}{B - x}.$$

**Step 3: sell the  $x$  base tokens back.** The on-chain invariant after the burn is

$$k_{\text{burn}} = (A_2 + V_2)B_2. \quad (27)$$

Trading  $x$  base tokens back into the pool yields the pre-fee quote token outflow  $S_{\text{pool}}$  defined by

$$(A_2 + V_2)B_2 = (A_2 - S_{\text{pool}} + V_2)(B_2 + x), \quad (28)$$

which solves to

$$S_{\text{pool}} = \frac{x}{(B - x)(B - y)} [B(A + V) - Vy]. \quad (29)$$

The fee clips a factor  $1/q$  from the output, so the attacker actually receives

$$S_{\text{sell}} = \frac{1}{q} S_{\text{pool}} = \frac{1}{q} \frac{x}{(B - x)(B - y)} [B(A + V) - Vy]. \quad (30)$$

**Net payoff and burn threshold.** The round-trip profit becomes

$$\Pi_{\text{fee}}(x, y) = S_{\text{buy}} - S_{\text{sell}} = \frac{x}{q(B - x)(B - y)} [q^2(A + V)(B - y) - B(A + V) + Vy]. \quad (31)$$

Fees suppress the attack whenever the bracketed term is non-negative. Solving the linear inequality for  $y$  yields the maximum burn size that still keeps the attacker under water:

$$y \leq y_{\text{max}} = \frac{(q^2 - 1)B(A + V)}{q^2(A + V) - V}, \quad (32)$$

The main observation is that the profitability condition depends only on the burn size  $y$  relative to the pool parameters; the buy size  $x$  affects the magnitude of profit but not its sign, so the threshold  $y_{\max}$  is independent of the attacker's purchase amount.

Assuming  $A > 0$  (otherwise there is nothing to extract) keeps the denominator positive. The bound scales linearly with the base token reserve  $B$ ; the multiplier depends on the fee rate (via  $q$ ), the virtual reserve  $V$ , and the real quote token reserve  $A$ . Setting  $N = 0$  (so  $q = 1$ ) collapses the bound to  $y_{\max} = 0$ , matching the logic in Equation (23). For  $q > 1$ , the attack is eliminated whenever the configured burn allowance lies below  $y_{\max}$ .

Even when fees prevent the attacker from profiting (i.e., when  $y \leq y_{\max}$ ), the pool can still be negatively impacted by the attack. The burn reduces the base token reserve and forces a downward adjustment of the virtual reserve, which reduces liquidity and can lower the worst-case exit price. Moreover, the fees collected during the attack may not fully compensate for these negative effects. Therefore, it would be reasonable to consider allowing only smaller burn sizes  $y$  than  $y_{\max}$  to further limit the pool's exposure to such attacks, even when they are not profitable for the attacker.

### 2.4.3 Attack model with symmetric base-token fees and topup

We extend the attack model to account for a quote token reserve top-up that occurs after the burn. The top-up increases the real quote token reserve and adjusts the virtual reserve for optimal utilization as described in Section 2.3. We keep the same notation as before: the pool has an initial state with real quote token reserve  $A \geq 0$ , base token reserve  $B > 0$ , virtual reserve  $V > 0$ , total base token supply  $T \geq B$ , and invariant  $k = (A + V)B$ . Fees are charged symmetrically on the quote token at rate  $N\%$ , with fee multiplier

$$q = \frac{100}{100 - N} > 1. \quad (33)$$

An adversary executes the buy–burn–topup–sell loop: (i) buys  $x$  base tokens ( $0 < x < B$ ) at cost  $S_{\text{buy}} = q(A + V)x/(B - x)$ , (ii) triggers a burn of  $y$  base tokens ( $0 < y < B - x$ ) followed by a top-up, and (iii) sells the  $x$  base tokens back. Since top-ups increase the real quote token reserve, they increase the attacker's profit from selling base tokens back. The worst case for the protocol occurs when the maximal top-up  $M$  (as defined in Section 2.3) is applied, as this maximizes the attacker's gain.

**Concrete example: profitable attack despite fees.** To see that fees alone do not suffice when top-ups are unconstrained, consider the following concrete example. Take

$$A = 100, \quad B = 100, \quad V = 10,$$

so that  $A + V = 110$  and the invariant is  $k = (A + V)B = 11,000$ . From  $k = VT$  we get a total supply  $T = k/V = 1100$ . Set a symmetric fee of  $N = 5\%$ , so  $q = 100/95 \approx 1.0526$ .

The attacker executes the following steps:

1. **Buy  $x = 90$  base tokens.** This costs  $S_{\text{buy}} \approx 1042.11$  quote tokens, leaving  $B_1 = 10$  base tokens in the pool.
2. **Burn  $y = 1$  base token.** The post-burn state has  $B_2 = 9$  base tokens in the pool. Note that  $y = 1$  satisfies the condition from the previous subsection: with  $q^2 = (100/95)^2 \approx 1.108$ , we have  $y_{\max} = \frac{(q^2 - 1)B(A + V)}{q^2(A + V) - V} \approx 10.62$  (calculated using the initial  $B = 100$ ). Since  $y = 1 < y_{\max} \approx 10.62$  and  $y = 1 < B_1 = 10$ , the burn is feasible and the attack would be unprofitable without top-ups.



3. **Apply maximal top-up  $M$ .** The top-up mechanism from Section 2.3 injects the optimal amount  $M \approx 120.01$  to the real quote token reserve to restore the target price profile and adjusts  $V = V_{\text{opt}} \approx 9.9909$
4. **Sell the  $x = 90$  base tokens back.** The attacker receives  $S_{\text{sell}} \approx 1053.64$  quote tokens after fees.

The net profit is approximately  $11.53 > 0$  quote tokens. This example satisfies all feasibility constraints, yet leads to a profitable buy–burn–topup–sell loop even with a 5% symmetric fee. The issue is that the top-up injects enough real quote tokens to more than offset the fee friction.

This motivates one additional design constraint besides adding a sufficiently large symmetric fee - per-burn limit on the number of base tokens that may be burned. The derivation of the formula describing the relationship of these two parameters follows.

**Lemma 2.1** (Profit formula with top-ups). Let there be a CBMM pool in state  $(A, B, V)$  with invariant  $k = (A + V)B$ , total base token supply  $T > B$  and symmetric quote-token fee  $N\%$  and fee multiplier  $q = 100/(100 - N) > 1$ . After an attacker buys  $x$  base tokens, triggers a burn of  $y$  base tokens, applies a top-up  $0 \leq N \leq M$  (as defined in Section 2.3) and sells the  $x$  base tokens back, the maximum possible round-trip profit is

$$\Pi_{\text{topup}}(x, y, N) = \frac{x(T - y)^2 V}{qT(B - y)(B - x - y)} - q \frac{(A + V)x}{B - x}. \quad (34)$$

and happens when  $N = M$ .

*Proof.* After the attacker buys  $x$  base tokens, the pool state is

$$A_1 = A + \frac{(A + V)x}{B - x}, \quad B_1 = B - x, \quad V_1 = V, \quad T_1 = T.$$

A burn of  $y$  base tokens followed by a top-up  $N \geq 0$  yields

$$A_2 = A_1 + N = A + \frac{(A + V)x}{B - x} + N, \quad B_2 = B - x - y, \quad T_2 = T - y.$$

To optimize the virtual reserve to fully utilize the top-up amount as described in Section 2.3, the virtual reserve is adjusted to

$$V_2 = \frac{A_2 B_2}{T_2 - B_2} = \frac{A_2(B - x - y)}{T - B + x}. \quad (35)$$

When the attacker sells the  $x$  base tokens back, the pre-fee quote token outflow  $S_{\text{pool}}$  is determined by invariance:

$$(A_2 + V_2)B_2 = (A_2 - S_{\text{pool}} + V_2)(B_2 + x).$$

Solving and using Equation (35) and  $B_2 + x = B - y$  gives

$$S_{\text{pool}} = \frac{A_2 \cdot x \cdot (T - y)}{(B - y)(T - B + x)}.$$

After applying the symmetric fee, the attacker receives

$$S_{\text{sell}} = \frac{1}{q} S_{\text{pool}} = \frac{x(T - y)}{q(B - y)(T - B + x)} \left( A + N + \frac{(A + V)x}{B - x} \right).$$

The net round-trip profit with top-up  $N$  is therefore

$$\Pi_{\text{topup}}(x, y, N) = S_{\text{sell}} - S_{\text{buy}} = \frac{x(T - y)}{q(B - y)(T - B + x)} \left( A + N + \frac{(A + V)x}{B - x} \right) - q \frac{(A + V)x}{B - x}.$$

Since  $\Pi_{\text{topup}}$  is increasing in  $N$ , the worst case for the protocol is the maximal top-up  $N = M$  induced in Section 2.3. Substituting  $N = M$  from Equation (12) gives

$$\begin{aligned}\Pi_{\text{topup}}(x, y, M) &= \frac{x(T-y)}{q(B-y)(T-B+x)} \cdot \frac{T-y}{T} V \cdot \frac{T-B+x}{B-x-y} - q \frac{(A+V)x}{B-x} \\ &= \frac{x(T-y)^2 V}{qT(B-y)(B-x-y)} - q \frac{(A+V)x}{B-x}.\end{aligned}\tag{36}$$

□

**Theorem 2.1** (Safety under capped burns and top-ups). Let there be a CBMM pool in state  $(A, B, V)$  with invariant  $k = (A+V)B$ , total base token supply  $T > B$  and symmetric quote-token fee  $N\%$  and fee multiplier  $q = 100/(100 - N) > 1$ . Fix a burn cap parameter  $\eta \in (0, 1)$  and require that each burn event satisfies

$$0 < y \leq \eta(B-x).\tag{37}$$

If

$$q \geq \frac{1}{1-\eta},\tag{38}$$

then for all  $A > 0$ ,  $V > 0$ ,  $B > 0$ ,  $0 < x < B$ , and all admissible burns  $y$  obeying Equation (37), the worst-case profit  $\Pi_{\text{topup}}(x, y, M)$  from Lemma 2.1 is non-positive.

*Proof.* Starting from Equation (34), the burn cap Equation (37) implies

$$B-x-y \geq B-x-\eta(B-x) = (1-\eta)(B-x),$$

so

$$\frac{1}{B-x-y} \leq \frac{1}{(1-\eta)(B-x)}.$$

Thus the first term in Equation (34) is bounded above by

$$\frac{x(T-y)^2 V}{qT(B-y)(1-\eta)(B-x)}.$$

Using  $(T-y)^2 \leq T^2$  (since  $0 \leq y < T$ ), we obtain the upper bound

$$\begin{aligned}\Pi_{\text{topup}}(x, y, M) &\leq \frac{xT^2 V}{qT(B-y)(1-\eta)(B-x)} - q \frac{(A+V)x}{B-x} \\ &= \frac{xTV}{q(B-y)(1-\eta)(B-x)} - q \frac{(A+V)x}{B-x}.\end{aligned}\tag{39}$$

Factoring out the positive quantity  $x/(B-x)$ , we see that  $\Pi_{\text{topup}}(x, y, M) \leq 0$  is guaranteed whenever

$$\frac{TV}{q(B-y)(1-\eta)} - q(A+V) \leq 0,\tag{40}$$

i.e.

$$TV \leq q^2(A+V)(B-y)(1-\eta).\tag{41}$$

Substituting  $TV = (A+V)B$  from  $k = (A+V)B = VT$  and cancelling  $A+V > 0$  yields

$$B \leq q^2(B-y)(1-\eta).\tag{42}$$

For any admissible burn we have  $y \leq \eta(B-x) \leq \eta B$ , hence  $B-y \geq (1-\eta)B$ . Since the right-hand side of Equation (42) is increasing in  $(B-y)$ , it is enough to check the worst case  $B-y = (1-\eta)B$ , which gives

$$B \leq q^2(1-\eta)^2 B \iff q^2(1-\eta)^2 \geq 1.$$

This is exactly Equation (38), i.e.  $q \geq 1/(1-\eta)$ . Under this condition we have  $\Pi_{\text{topup}}(x, y, M) \leq 0$  for all admissible  $(A, B, V, x, y)$ , completing the proof. □

**Parameter choice.** Theorem 2.1 is stated for a generic per-burn cap  $\eta$  and fee multiplier  $q$ . In terms of the fee rate  $N\%$ , the condition Equation (38) becomes

$$\frac{100}{100 - N} \geq \frac{1}{1 - \eta} \iff N \geq 100\eta.$$

Thus any symmetric quote-token fee of at least  $100\eta\%$  suffices to make the buy–burn–topup–sell loop unprofitable, regardless of the pool state and attack size, as long as each burn obeys the cap from Equation (37).

### 3 Implementation

This section describes an on-chain implementation of CBMM as a on-chain program. The trading principles in CBMM implementation use the same approach as standard CPMM pools and, therefore, are not repeated here. We first present the Continuous Conditional Buybacks mechanism, which partially offsets the required reduction in the virtual reserve  $V$  after burns. We then outline safety controls for burn operations and note key configuration considerations. The design objective is to minimize user friction while faithfully realizing the mathematical model from Section 2.

We are implementing this on SOLana, but most of the concepts are general and can be applied to other chains. For the initial version of the CBMM pool we’ve decided not to use an existing standard for the base token (like SPL or Token Extensions), but to use an on-chain counter, that works in the same way as the standard token would. This allows us to keep the system closed and not to worry about any side effects. However, the quote token is a standard SPL token. We still use the same name base token and quote token for consistency with the mathematical model.

#### 3.1 Continuous Conditional Buybacks

As discussed in Section 2.2, burns require a reduction of the virtual reserve from  $V_1$  to  $V_2$  (with  $V_2 < V_1$ ). This adjustment reduces the positive impact of the burn. We implement the quote token reserve top-up mechanism described in Section 2.3 as **Continuous Conditional Buybacks** (CCB) redirecting a portion of trading fees into the Real Token Reserve  $A$ .

Let  $\Delta V = V_1 - V_2 \geq 0$  denote the required reduction in virtual reserve implied by Section 2. The implemented CCB mechanism then works as follows:

- **Fee accumulation:** On each trade, a fixed fraction of token fees is routed to a dedicated on-chain fee vault (an associated token account controlled by the program).
- **Burn-time top-up:** Upon a burn event, the program strives to top-up the pool real and virtual quote token reserves to the target values  $A_{\text{opt}}$  and  $V_{\text{opt}}$  as described in Section 2.3. Any shortfall is stored in pool state as an outstanding liability  $L$ .
- **Continuous repayment:** While  $L > 0$ , fees contributed by the subsequent trades are immediately applied to reduce  $L$  until it reaches zero. Any overage remains in the fee vault and is handled by the Fee accumulation rule.

The Continuous repayment step differs for beans buys and sells. For buys the fees are applied before the operation itself, for sells the fees are applied after the operation. This directly incentivizes buys and penalizes sells. Moreover, the topup always happens with the lower outstanding amount, which lowers the required top-up amount to achieve the target price.

Technically, this is not a buyback, as no beans leave the pool. However, adding tokens to  $A$  increases the beans’ price and effectively buys back part of the burn’s price impact.

If not enough off-chain activity is observed, the trading fees are still accumulated and can be used to top-up the pool in the future. This motivates the off-chain activity to be persistent and ongoing.

### 3.2 Burn Safety Mechanisms

As we know from the attack section, we need to make sure that our burn and fee setup is ok so that noone can actually make money out of this. There are multiple ways this can be achieved - both on-chain and off-chain. The first important setup - CCB section - is that we top up immediately after buys and sells when applicable so then on the buy there is usually not such a big jump in the price - these happen continuously. The full topup from section about topups only happens in case there is a lot of trading activity and we have enough tokens in the dedicated pool on-chain fee vault. Even so, we need to make sure that the burn never goes over the limit. This is even worse, because this is not necessarily one single burn, but it might be multiple of them chained together. Therefore we've decided to employ a throttling mechanism that limits the number of burns that can be performed in a given time period. This way people (bots) have time to react to the burn sequence and dump on the attacker during the burns.

Moreover, we've decided to implement a centralized burn authority that is responsible for approving burns. This is done to ensure that all burns are tied to specific off-chain activities and that users have not simply automated the burn operation by calling the on-chain program directly. This mechanism is optional and can be turned off if decentralization is a priority. This allows for off-chain throttling as well, not only on-chain throttling.

Another important consideration is that we only allow burns of certain predefined sizes. In combination with throttling and fees we can ensure that even if there is an automated burn going on, the theoretical profit limit will only be reached in longer time period and therefore would be influenced by other trading activity and very hard to achieve.

For convenience we introduce daily burn limits per wallet as well, however, these do not protect against sibyl attacks of course.

To bound operational risk and align burns with off-chain incentives, we introduce two controls: (i) per-user daily burn limits and (ii) a centralized burn authority.

#### 3.2.1 Daily burn limits

We introduce a lightweight on-chain counter, `UserBurnAllowance`, uniquely identified by the user's wallet address. It records the number of burns over a 24-hour window and the timestamp of the most recent burn. Account creation is permissionless (similarly to SPL Associated Token Accounts) so any party can create and fund [6] an account to bootstrap a user's allowance. To avoid rent leakage, accounts associated with users inactive for  $\geq 24$  hours may be closed and rent returned to the original funding wallet. If a user exceeds the configured daily limit, additional burn attempts are rejected until the window resets. The impact of each burn is calculated as a hardcoded percentage of the pool's base token reserve. For more significant impact it is possible to burn a percentage of the total supply if the pool reserve is sufficient to do so.

For minimal funding costs and better scalability, equivalent functionality could be realized via State Compression using Concurrent Merkle Trees [7], at the cost of off-chain infrastructure dependence.

#### 3.2.2 Burn authority

On every burn we require a signature from a Burn Authority. This ensures that all burns are tied to specific off-chain activities and that users have not simply automated the burn operation by calling the on-chain program directly. This mechanism is optional and can be turned off if decentralization is a priority.

### 3.2.3 Burn cap and fee parameter choice

The safety theorem Theorem 2.1 establishes conditions on the burn cap parameter  $\eta$  (from Equation (37)) and the symmetric base-token fee rate  $N\%$  (from Equation (38)) that guarantee the buy–burn–topup–sell loop is unprofitable. In our deployment we choose

$$\eta = 0.01 \quad \text{and} \quad N = 5\%.$$

The burn cap  $\eta = 0.01$  means each burn removes at most 1% of the post-buy pool, while the fee condition from Theorem 2.1 requires only  $N \geq 1\%$ . Our actual fee of 5% (so  $q = 100/95 \approx 1.0526$ ) comfortably satisfies Equation (38), leaving a wide safety margin. Even when the maximal top-up  $M$  is applied after every burn, the loop remains strictly unprofitable for all admissible parameters.

### 3.3 Pool Configuration Considerations

The main decision that influences the pool behavior is the initial virtual reserve  $V$ . This reserve directly sets the token initial price and is proportional to the trading volume needed to be able to top-up the pool fully. Moreover, the higher initial reserve the more expensive it is to snipe a large amount of base tokens at the starting price.

## 4 Conclusion

This work set out to design a market-making mechanism that can translate verifiable off-chain activity into on-chain price impact through controlled burns. We constructed a mathematical model for CBMM with a virtual reserve  $V$  and outlined the closed-form expressions for trading, burning, and the induced price impact. On the implementation side, we described a Solana program architecture that implements these mechanics, introduces Continuous Conditional Buybacks (CCB) to partially offset the virtual-reserve reduction after burns, and adds safety controls to regulate burn frequency and authorization.

The model reveals a clear separation of roles. First, burns directly increase price with a relative impact proportional to the amount of beans outside the pool and the burned amount. The pool trading volume by itself does not directly contribute to the price increase but complements burns by compensating for the virtual reserve deterioration through the Continuous Conditional Buybacks mechanism. Because burning reduces the in-pool supply, it increases price but also increases volatility. The virtual reserve acts here as a bridge mechanism to facilitate a slow transition from a purely virtual scaling system into a progressively collateralized one. In healthy conditions, as  $V$  is reduced toward its minimum, the pool matures by price becoming increasingly supported by real collateral rather than virtual scale.

From a design perspective, the initial virtual reserve  $V$  sets the starting price and determines how much trading volume is required to fully top up the pool via fees. Larger  $V$  makes early sniping more costly.

We have investigated alternative invariant types beyond the constant product form. For instance, Power CBMM uses an invariant of the form  $k = (A + V)B^p$  with a power parameter  $p > 0$ , which allows fine-tuning the bonding curve shape:  $p > 1$  produces a steeper curve that concentrates price impact at earlier purchases (potentially mitigating sniping risk), while  $0 < p < 1$  produces a flatter curve. However, more complex invariant types yield virtual reserve adjustment formulas that are overly complex for convenient on-chain implementation using integer arithmetic and conveniently proving the safety theorems.

This paper describes the CBMM pool as a base token rollout mechanism. Other directions for future work include generalizing CBMM to other asset pairs, allowing third-party liquidity provision (LP) in a controlled manner, and analyzing CBMM pool with nonzero initial quote

token reserve that should be preserved by the burn mechanism. Investigation of the CBMM as a part of a broader financial ecosystem where there are other pools that would create arbitrage opportunities between them is also an interesting direction for future work. These mechanisms might require  $V < 0$  and are left for future study.

## References

- [1] Guillermo Angeris and Tarun Chitra. “Improved Price Oracles: Constant Function Market Makers”. In: *arXiv preprint arXiv:2003.10001* (2020). arXiv: 2003.10001 [q-fin.TR]. URL: <https://arxiv.org/abs/2003.10001>.
- [2] Hayden Adams. *Uniswap Whitepaper*. 2020. URL: <https://hackmd.io/@HaydenAdams/HJ9jLsfTz>.
- [3] Hayden Adams et al. *Uniswap v3 Core*. <https://app.uniswap.org/whitepaper-v3.pdf>. Mar. 2021.
- [4] Meteora Team. *What’s DAMM v2? (Product Overview)*. Accessed Nov 5, 2025. 2025.
- [5] Eyal Hertzog, Guy Benartzi, and Galia Benartzi. *Bancor Protocol: Continuous Liquidity and Asynchronous Price Discovery for Tokens through their Smart Contracts (aka “Smart Tokens”)*. <https://resources.bancor.network/pages/BancorProtocolWhitepaper.pdf>. Describes the constant-reserve-ratio bonding curve mechanism; archived translations available on GitHub. 2018.
- [6] *Rent on Solana*. <https://docs.solana.com/developing/programming-model/accounts#rent>. 2022.
- [7] Jarry Xiao et al. *Compressing Digital Assets with Concurrent Merkle Trees*. Whitepaper. Concurrent Merkle Tree whitepaper; accessed 2025-11-06. Solana Labs, 2023. URL: <https://drive.google.com/file/d/1B0pa50Fmara50fTvL0VIVYjtg-qzHCVc/view>.