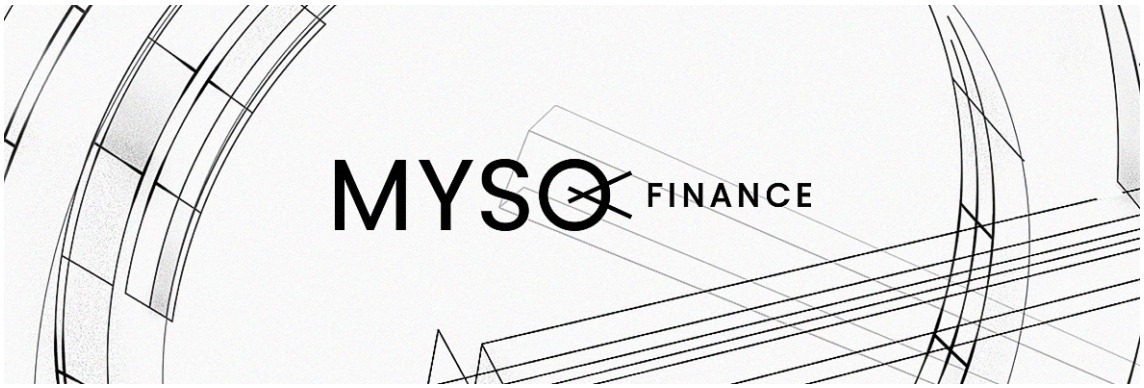
MYSO v1 Core

A Trust-Minimized Protocol for Zero-Liquidation Loans

October 2022

MYSO Finance Association
Aetienne Sardon
as@myso.finance



Abstract

MYSO v1 is a trust-minimized DeFi protocol implemented for the EVM that enables crypto-collateralized loans without liquidations, so called Zero-Liquidation Loans (ZLLs). The protocol allows borrowers to take out ZLLs through non-custodial liquidity pools, which on the other side Liquidity Providers (LPs) can fund. Liquidity pools can be created by anyone using MYSO's open source v1 core smart contracts. Each pool is defined by (i) an asset pair, (ii) a maximum loan amount per pledged collateral unit, (iii) a loan tenor and (iv) an interest rate model. After a pool has been created and funded, borrowers can take out ZLLs by pledging into the liquidity pool an amount of collateral token (e.g. `wETH`) and borrowing an amount of loan token against it (e.g. `USDC`). Borrowers then have an option (but no obligation) to reclaim their previously pledged collateral if they pay a fixed repayment amount, which is determined autonomously by the pool contract and predominantly depends on current liquidity supply and demand. If a borrower doesn't repay before the ZLLs expires, they forfeit their ability to reclaim their collateral, thereby making it available to LPs for claiming. By design, MYSO v1 doesn't involve any liquidations but instead allows LPs to earn yield for bearing the risk that a loan may become undercollateralized and not be repaid. The associated risk-reward profile resembles that of an in-the-money covered call position, which is a well-known conservative yield enhancement strategy.

Contents

1 Introduction

Despite the recent crypto bear market, borrowing and lending protocols remain one of the most critical components in Decentralized Finance (DeFi) market infrastructure. These protocols allow users to borrow funds by using their crypto assets as collateral and, on the other side, enable lenders to earn yield for collectively funding such loans. Most loans in DeFi today are overcollateralized[1], essentially resembling Lombard loans in Traditional Finance (TradFi).[2] Currently, the primary use cases for crypto-collateralized loans are carry trades, increasing capital efficiency and leveraged trading [1]. With a combined Total Value Locked (TVL) of $15bn as of October 2022 [3], the protocols Maker, Aave and Compound are the most well-known players in the space. Though distinct differences exist between these protocols, one key aspect that they share is that they follow a *liquidation-centric* design. This means that borrowers get liquidated if their collateral falls below a certain liquidation threshold, in which case they also incur a liquidation penalty fee. For example, between April 2019 and April 2021 there were over 2,011 liquidators with an average profit of $31.62k [13], meaning that in aggregate over $60m in liquidation penalty fees were paid by users during this time. More recently, in January 2022, users on Maker paid over $15.5m in liquidation penalty fees in just a single day [7]. For borrowers, this represents a significant pain point, requiring them to continuously monitor their loans to avoid being liquidated. And perhaps even more precarious, liquidation-centric borrowing systems introduce several systemic risks to the overall crypto market, such as cascading liquidations [4], concentration risk [8][5], liquidation related Maximal Extractable Value (MEV) [6], *overliquidations* [13] and oracle manipulations [11]. Zero-Liquidation Loans (ZLLs), as first described in [15], provide a novel approach to DeFi borrowing and lending that can help mitigate some of the aforementioned systemic risks. While some protocols have started exploring this field [14][19][20], it has remained largely underdeveloped until now.

2 Protocol

2.1 Goal

The goal of the MYSO v1 protocol is twofold: (1) simplify crypto loans for borrowers, and (2) provide yield enhancement opportunities for Liquidity Providers (LPs). ZLLs allow us to link these two goals together and make them complementary to each other. To be more specific, ZLLs act as a mutually beneficial risk transfer mechanism, in which the loan liquidation risk is transferred from the borrower to the lender, and the lender can earn yield for bearing this risk. From the borrower's perspective, removing liquidations eliminates one of the biggest friction points of current crypto loan offerings, where users typically have to constantly monitor their health factors and liquidation thresholds. From the LP's perspective, funding such loans provides exposure to an in-the-money covered call[3], which has been a well-known conservative yield enhancement strategy in TradFi

[1] While undercollateralized loans also exist in DeFi, they currently represent a smaller portion of the overall credit market and questions are raised around their permissionlessness, however, they may complement overcollateralized loans in the future [1].

[2] Lombard loans allow borrowers to cover short-term liquidity needs by using account balances or life insurance policies as collateral without having to sell these assets [12]. What differentiates DeFi crypto-collateralized loans from Lombard loans is that the former are facilitated and settled through open decentralized networks (e.g. Ethereum), enabling permissionless borrowing for anyone at any time. Moreover, since a protocol's smart contracts are typically open sourced and publicly verifiable users have full transparency about the underlying asset and liability management mechanics. This is different to both TradFi and Centralized Finance (CeFi) lending platforms, where these systems are usually closed and cannot be inspected by the public.

[3] Or, equivalently, a synthetic short put.

[9][18] and with similar strategies becoming increasingly adopted in DeFi through Decentralized Option Vaults (DOVs).[4] Moreover, MYSO v1 can contribute to make crypto markets overall more resilient and robust against systemic risks such as cascading liquidations or oracle failures and reduce externalities associated with liquidation-related MEV. As such, MYSO v1 can serve as a viable complement to today's predominantly liquidation-centric borrowing and lending platforms.

2.2 Overview

The MYSO v1 protocol is designed to efficiently facilitate ZLLs between borrowers and LPs through non-custodial liquidity pools. The process by which this happens is as follows:

1. Creating pools: first, a pool needs to be created. This can be done by anyone using the MYSO v1 `BasePool.sol` template contract and deploying a derived asset-pair specific pool implementation contract. At deployment, the creator of the pool must specify the intended asset pair, loan tenor and maximum loan amount per pledged collateral unit. These values are immutable and cannot be changed after deployment.

2. Adding liquidity: after a pool has been deployed, LPs can add liquidity and thereby acquire a share in the given pool.

3. Borrowing: borrowers can choose the pool that best matches their preferred collateral and loan currency combination, loan tenor etc. and borrow from it. Loans are then funded proportionally by all active LPs of the given pool, who thereby become entitled to a pro-rata share of the resulting loan proceeds (peer-to-pool model).

4. Repaying or defaulting: once a borrower has taken out a loan they can either (a) repay and reclaim their collateral or (b) let the loan expire. In either case, the corresponding loan is then marked as settled and the associated proceeds are made available to LPs.

5. Claiming: after a loan has been settled, LPs can claim a pro-rata share of the corresponding proceeds. In case the loan was repaid, the proceeds are (a) a pro-rata share of the corresponding repayment amount, and otherwise (b) a pro-rata share of the unclaimed collateral amount.[5]

6. Removing liquidity: LPs can remove liquidity that hasn't (yet) been lent to borrowers.[6]

2.3 Pool Creation

MYSO v1 provides an abstract `BasePool.sol` contract, which contains generic and asset pair agnostic functions for adding liquidity, borrowing, repaying, removing liquidity, etc. As an abstract contract, users cannot deploy `BasePool.sol` itself but instead must create asset pair specific pool implementation contracts that inherit from `BasePool.sol` (see fig. 8).[7][8] The MYSO v1 core repository provides examples for such pool implementation contracts, which can be used at users' own risk.

[4] What is noteworthy is that with ZLLs the opposite side of the covered call position is taken from borrowers, who are natural call buyers and provide organic order flow. This stands in contrast to many DOVs, where the other side of the trade is often times taken by informed and potentially toxic order flow.

[5] Note that the resulting proceeds aren't automatically reinvested back into the pool (similar to Uniswap v3).

[6] Note that removing liquidity is subject to a minimum liquidity provisioning period of 120 seconds.

[7] In particular, the implementation contract must overwrite the virtual functions `getCollCcyTransferFee()` and `getLoanCcyTransferFee` to handle potential transfer fees of the given asset pair.

[8] Note that users should carefully assess and test whether their intended asset pair is compatible with `BasePool.sol`,

When a user deploys a new pool the following immutable parameters are set:

- `collCcyToken`: the collateral currency accepted by the pool (e.g. `wETH`).[9]
- `loanCcyToken`: the loan currency accepted by the pool (e.g. `USDC`).
- `loanTenor`: the loan tenor for which users can borrow (e.g. 30 days).
- `maxLoanPerColl`: the maximum loan currency amount one can borrow per pledged collateral unit.[10]
- `minLoan`: the minimum loan size for which borrowing shall be possible.[11]
- `r1, r2, liquidityBnd1, liquidityBnd2`: interest rate model parameters that define how the pool determines its rates (see section 2.5.3).
- `baseAggrBucketSize`: the base aggregation bucket size according to which loans are batched together to enable more efficient claiming (see section 2.4.3).
- `creatorFee`: a fee of at most 30 bps, which is deducted from the collateral that borrowers pledge into the pool and which is sent to the creator.[12]
- `minLiquidity`: a minimum amount of loan currency that is retained within the pool.[13]

Note that after a pool has been created, neither the creator nor any other third party can control the pool or change any parameters. If parameters aren't properly set or become outdated (e.g., `maxLoanPerColl` in case the collateral price changes significantly) users may need to deploy a new pool instance with more suitable parameters.[14]

for example, w.r.t. token decimals, or upgradable behavior.

[9] Note that pools are not symmetric w.r.t. the asset pair they support, i.e., a pool with `wETH` as collateral and `USDC` as loan currency can only be used to borrow `USDC` against `wETH`, but not vice versa.

[10] For example, if `maxLoanPerColl=1000` this means for every pledged `wETH` unit one can at most borrow 1000 `USDC` (see section 2.5.2).

[11] This is enforced to prevent DoS attacks.

[12] The deployer of a pool is automatically its creator. Note that the creator fee is immutable and cannot be changed after deployment.

[13] This is done to mitigate potential overflows of the amount of total pool shares in case of large liquidity injections that are directly followed by large borrows.

[14] Note that duplicate pool instances are possible, in particular, MYSO v1 core doesn't include any factory contract. Moreover, note that it might happen that pool parameters are only temporarily off, for example, if the collateral price falls below `maxLoanPerColl` (which would create an arbitrage opportunity and cause liquidity to be removed from the pool) but later the price rises above that level again.

2.4 Liquidity Provisioning

2.4.1 Adding Liquidity

LPs can add liquidity to pools to fund future incoming ZLLs and earn a pro-rata share of the associated proceeds. When a LP adds y in loan currency to a pool, they receive the following amount of pool shares $s(y)$:

$$s(y) = \begin{cases} \frac{y}{L} \cdot N_{shares} & \text{if } L > 0 \\ \frac{1000 \cdot y}{L_{min}} & \text{else} \end{cases} \tag{1}$$

where L is the amount of total liquidity in the pool before the LP adds, N_{shares} is the currently total outstanding number of shares, and L_{min} denotes a minimum liquidity amount that is retained within a pool and which is configured at deployment time (see section 2.3). Note that when a LP adds liquidity into an empty pool, their liquidity contribution is normalized by $\frac{L_{min}}{1000}$ to reduce the amount of pool shares that need to be minted.[15]

2.4.2 Removing Liquidity

LPs can remove liquidity from a pool by redeeming s pool shares to receive the following pro-rata amount $y(s)$ of loan tokens:

$$y(s) = \frac{s \cdot (L - L_{min})}{N_{shares}} \tag{2}$$

where L is the amount of total liquidity in the pool that hasn't yet been lent to borrowers, L_{min} is a minimum liquidity amount retained within a pool, and N_{shares} is the total amount of outstanding pool shares. Note that LPs must wait for a minimum lockup period of 120 seconds before they can remove liquidity.[16]

2.4.3 Claiming

Once a loan has been repaid (or defaulted), LPs can claim a pro-rata share of the corresponding repayment (or left collateral) amount. To keep track of the loans for which a LP can claim, liquidity pools have a global `loanIdx` counter that is incremented after each new loan. Whenever a LP adds or removes liquidity, the pool stores the `loanIdx` at which this happened. To be more precise, for every LP the pool maintains a `struct LpInfo` with the following members:

- `fromLoanIdx`: the lower bound loan index (incl.) from which the LP is entitled to claim. Initially, this is set to the current loan index at which the LP first added liquidity and then gets updated every time the LP claims, i.e., is set to the highest loan index plus one on which the LP claimed last.[17]
- `earliestRemove`: earliest timestamp after which a LP is allowed to remove liquidity again. This is always 120 seconds after the LP added liquidity and is enforced to prevent flash liquidity provisioning.

[15] This helps mitigate possible overflows of N_{shares} without significantly losing precision on pro-rata loan proceeds, especially for loan currencies with many decimals.

[16] This is done to prevent flash liquidity provisioning. Note that as soon as a LP adds (or tops-up) liquidity, their lockup period is renewed, requiring them to wait again for 120 seconds before they can remove liquidity.

[17] Note that the `fromLoanIdx` is strictly monotonically increasing. This means that if a LP claims on a 'too high'

- `sharesOverTime[]`: an array storing the amount of shares a LP had at different points in time. New elements are added for consecutive adding or removing of liquidity.
- `loanIdxsWhereSharesChanged[]`: an array containing the loan indices at which a LP changed their amount of shares (see `sharesOverTime[]`). The elements are used as upper loan index bounds (excl.) for constant share claiming intervals, i.e., LPs can claim until `loanIdxsWhereSharesChanged[i]` with `sharesOverTime[i]`. And if index `i` is out-of-bounds of `loanIdxsWhereSharesChanged[]` then the LP can claim up until latest global `loanIdx` with `sharesOverTime[i]`.
- `currSharePtr`: current share pointer to indicate which `sharesOverTime[]` element to be used for the interval `fromLoanIdx` to `loanIdxsWhereSharesChanged[currSharePtr]` or, if out-of-bounds, from `fromLoanIdx` to global `loanIdx`.

LPs can claim loan proceeds by using the following two functions:

- `claim()`: LPs can claim loan proceeds on an individual per-loan-index basis. For this, LPs need to specify a set of loan indices they want to claim on, e.g., $\{2, 3, 5, 7\}$.[18] The `claim()` function iterates over the given loan indices, checks whether the LP is entitled to these and, if so, returns the corresponding pro-rata repayment amounts (or left collateral amounts). Note that if there are many loan indices to iterate over, calling this function can lead to high gas costs, and users should consider using `claimFromAggregated()` instead.
- `claimFromAggregated()`: LPs can claim on whole loan index ranges, referred to as *buckets*, for example, from `loanIdx=1` to `loanIdx=100`. The valid interval buckets on which a LP can claim are defined by the given pool (see section 2.3). To be more precise, each pool aggregates loans into buckets of size `baseAggrBucketSize`, `10*baseAggrBucketSize` and `100*baseAggrBucketSize`. For example, if `baseAggrBucketSize=100` then loans are grouped into buckets of size $[1, 100), [100, 200), ...$ etc. as well as $[1, 1000), [1000, 2000), ...$ etc. and $[1, 10000), [10000, 20000), ...$ etc.

Note, however, that claiming loan proceeds is only possible on a set (or range) of loan indices during which the LP didn't change their position (i.e., didn't add or remove liquidity). If the LP changed their position they must do multiple claiming transactions, with each being over a loan index set (or range) for which their pool shares were constant.[19] For example, assume there's a LP who held 100 pool shares from loan index 1 to 28 and 200 pool shares from index 29 to 50. If the LP wants to use the `claim()` function for loan indices $\{1, 2, ..., 50\}$ they must call the function twice, i.e., once with $\{1, 2, ..., 28\}$ and once with $\{29, 30, ..., 50\}$. Similarly, if a LP wants to use the `claimFromAggregated()` function they can only do this if they were invested for the whole bucket they're trying to claim from, and if they maintained a constant pool share position throughout the whole bucket. For example, if a LP held 100 pool shares from loan index 1 to 98 and then afterwards added liquidity at loan index 99, they'll lose their ability to claim from the $[1, 100)$ aggregation bucket. Hence, LPs need to be mindful of how LP position changes can affect the intervals on which they can claim. Generally, LPs shouldn't change their LP position too often or at inopportune times to avoid transaction overhead and keep their claiming-related gas costs low.

loan index, their `fromLoanIdx` will be updated accordingly such that intermediate ones will be skipped and they'll not be able to claim on those afterwards anymore.

[18] Note that the provided loan indices must be in ascending order.

[19] Note that the construction of such loan index claiming transactions can be automated on the frontend side. [16]

2.4.4 Risks for LPs

When providing liquidity to a MYSO v1 pool, LPs should be aware of the following risks:

- Collateral price risk: LPs are exposed to collateral price risk, i.e., if during the lifetime of a loan the associated pledged collateral becomes worth less than the owed repayment amount, a borrower will not repay and LPs will not earn the repayment amount but instead receive the depreciated collateral asset (see section 2.7.2). Moreover, when a LP adds liquidity to a pool it may happen that during the liquidity provisioning time the collateral price falls below the `maxLoanPerColl` value (see section 2.3). In this case, LPs are exposed to arbitrage, where borrowers could borrow more from the pool than what the pledged collateral is worth. Hence, LPs need to actively monitor the pools they're invested in and remove liquidity if needed to prevent potentially being arbitraged.

- Unpredictability of yield: the yield a LP can earn by adding liquidity to a pool cannot be known in advance. This is because, depending on how the collateral price changes, borrowers will repay or not, either yielding the repayment amount or the depreciated collateral. Secondly, the effective interest rate at which loans are given to borrowers is dynamic and changes depending on liquidity supply and demand (see section 2.5.3). And lastly, the yield greatly depends on the borrower activity, i.e., how many loans are effectively taken out while the LP is active.

- Pool dilution: if other LPs add liquidity into a pool, previous LPs get diluted. This means that older LPs will fund new incoming loans at a lower pro-rata share, and, as a result, will also only be entitled to a lower pro-rata share of the associated loan proceeds. In case of large liquidity injections, dilution can be significant and potentially even cause claimable amounts to become negligible and prone to truncation errors.

- Variable APR: the rates at which users can borrow changes depending on liquidity supply and demand (see section 2.5.3). As a result, LPs cannot know in advance at which rate they will exactly fund an incoming ZLL. In particular, large liquidity injections by new LPs can cause the offered borrow rate to drop. However, the pool's interest rate model provides a lower bound r_2 that is guaranteed to never fall short of. Nonetheless, LPs should monitor the currently applicable borrow rate and remove liquidity in case they consider the current borrow rate inadequate.

- Minimum liquidity provisioning period: LPs must wait for a minimum liquidity provisioning period of 120 seconds before they can remove liquidity. During this time their capital is locked up.

- Overhead and costs of claiming: LPs must actively claim any loan proceeds. The process of claiming can come along with significant transaction overhead and gas costs (see section 2.4.3), especially if the LP is trying to claim proceeds from a larger number of loans. While the aggregation mechanism can help make claiming more efficient, it cannot be guaranteed that the LP will be able to take advantage of the full aggregation benefits. This is because the eligibility to claim from buckets depends on the time that the LP added liquidity. For example, if the LP added at loan index 100 and remains invested until loan index 200, they'll be able to claim from the full range from loan index 100 to 200. However, if they add liquidity at loan index 101, they'll have to claim individually by iterating over 99 loans, making claiming more costly for them. Lastly, some claiming functions allow LPs to overwrite certain claiming

provides some examples on how to construct such claiming intervals.

settings, which can lead to an irrevocable loss of entitled loan proceeds. Hence, LPs should call claiming related functions with great caution, especially when doing this programmatically or directly through etherscan.

- Non-transferability of LP position: in MYSO v1, LP positions are non-fungible and non-transferable, meaning that the only way to recoup a liquidity contribution is by removing any unused liquidity and claiming from all entitled loan proceeds. In particular, there's no secondary market through which a LP could convert their LP position into cash.
- Opportunity costs: if a LP adds liquidity into a pool it cannot be known in advance when the next borrower will arrive and when the LP's liquidity contribution can be utilized to fund the next loan. Hence, LPs can incur opportunity costs when there's little borrower activity in the pool. Moreover, once a loan has been settled, the corresponding loan proceeds aren't automatically reinvested but instead the LP needs to first actively claim them and reinvest if desired. Hence, unclaimed loan proceeds may sit idle in the pool and cause opportunity costs for LPs.

2.5 Borrowing

2.5.1 Zero-Liquidation Loans

A ZLL is a crypto-collateralized loan in which a user pledges collateral (e.g., `wETH`) to receive a loan amount (e.g., `USDC`) as well as an option to reclaim their collateral prior expiry of the loan if they pay a pre-agreed repayment amount. Users can choose which pool they want to borrow from according to their collateral and loan currency preferences, desired loan tenor, etc. When a borrower takes out a ZLL, the given pool assigns it a `loanIdx`[20] and associates it with the corresponding borrower address as well as a `struct LoanInfo` in which it stores the following information:

- `collateral`: the amount of collateral a borrower effectively pledges and can reclaim (post potential transfer and pool fees).
- `repayment`: the amount a user needs to repay to reclaim the pledged collateral amount (see section 2.5.3).
- `totalLpShares`: the number of total pool shares by which LPs need to later split the associated repayment and collateral amounts when claiming.
- `expiry`: the timestamp until repayment is possible and after which the borrower forfeits their collateral
- `repaid`: a flag indicating whether the loan was repaid or not.

2.5.2 Loan Amount Calculation

The amount $y(x)$ a user can borrow from a pool is defined as follows:

$$y(x) = \frac{x \cdot l \cdot L_a}{x \cdot l + L_a} \tag{3}$$

where x is the pledge amount (post transfer and pool fees), l is the maximum loanable amount per pledged collateral unit, and $L_a = L - L_{min}$ denotes the total available liquidity in a given pool. An alternative way to express eq. (3) is:

$$y(x) = \frac{(1 - u_x) \cdot x \cdot l + u_x \cdot L_a}{2} \tag{4}$$

where u_x denotes the pool's utilization and is given by:

$$u_x = \frac{x \cdot l}{x \cdot l + L_a} \tag{5}$$

From eq. (4) one can see that the loan amount $y(x)$ is a weighted average of $x \cdot l$ and L_a (see also fig. 1). In the extreme case where the pledge amount is zero (i.e., $x \to 0$ and $u_x = 0$) the infinitesimal loan amount per pledged collateral is:

$$\lim_{x \to 0} \frac{y(x)}{x} = \lim_{x \to 0} \frac{l \cdot L_a}{x \cdot l + L_a} = l \tag{6}$$

[20] The `loanIdx` starts at 1 and is incremented with every new incoming loan, making it unique per pool.

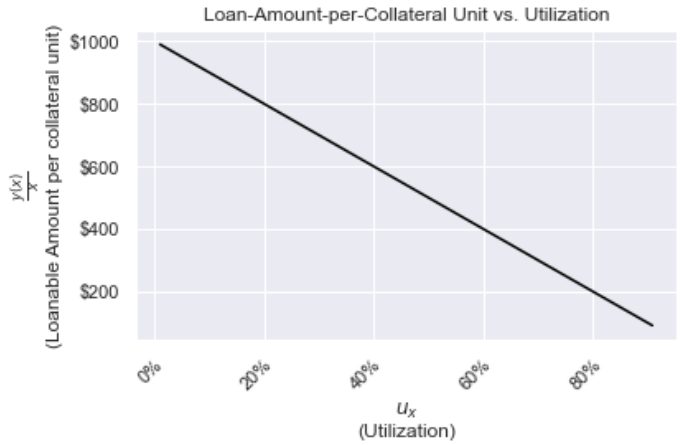


Figure 1: Example of how the loan-amount-per-pledged-collateral unit $\frac{y(x)}{x}$ changes for different pool utilization levels u_x, assuming $l = 1000$, $L = 10^6$, and $L_{min} = 1$.

Conversely, if a user pledges an infinite amount of collateral (i.e., $x \to \infty$ and $u_x \to 1$) the loan amount per pledged collateral unit converges to zero and the absolute loan amount to:[21]

$$\begin{aligned} \lim_{x\to\infty} y(x) &= \lim_{x\to\infty} \frac{x \cdot l \cdot L_a}{x \cdot l + L_a} \overset{L'Hôspital}{=} \lim_{x\to\infty} \frac{(x \cdot l \cdot L_a)'}{(x \cdot l + L_a)'} \\ &= \lim_{x\to\infty} \frac{l \cdot L_a}{l} = L_a \end{aligned} \tag{7}$$

Loan-to-Value
Let S denote the price of one collateral unit x. The Loan-to-Value (LTV) for a loan collateralized with x is then given by

$$\begin{aligned} LTV &= \frac{y(x)}{x \cdot S} \\ &= \frac{l \cdot L_a}{S \cdot (x \cdot l + L_a)} \end{aligned} \tag{8}$$

The more collateral x a user pledges, the closer the loan amount is to L_a (see eq. (7)), and the lower the LTV. The limit of the LTV as $\lim_{x\to\infty}$ is obviously 0. Note that a liquidity pool is oblivious of the collateral price S, i.e., if the price falls below the maximum loan amount per collateral unit, users can borrow at a LTV greater or equal 1.[22]

2.5.3 Repayment Amount & Applicable Interest Rate

If a user wants to reclaim their previously pledged collateral they must, prior to expiry of the loan, pay the following repayment amount to the pool:

$$RepaymentAmount = y(x) \cdot (1 + r) \tag{9}$$

[21] Figure 9 provides an illustration for the convergence behavior of $y(x)$ as $x \to \infty$.
[22] LPs are naturally incentivized to remove liquidity in these situations to prevent being arbitraged.

where r is a rate locked in at the time the user borrowed $y(x)$. The applicable rate r is given by:[23]

$$r = \frac{rate(L^*) + rate(L_a)}{2} \tag{10}$$

where $L^* = L_a - y(x)$ is the pool's liquidity after a user borrowed $y(x)$, and the underlying $rate(L_a)$ is defined as the following piecewise function:

$$rate(L_a) = \begin{cases} \frac{r_1 L_1}{L_a} & L_a < L_1 \\ r_2 + \frac{(r_1 - r_2)\cdot(L_2 - L_a)}{L_2 - L_1} & L_1 \leq L_a \leq L_2 \\ r_2 & L_2 < L_a \end{cases} \tag{11}$$

where $0 < L_1 < L_2$ and $0 < r_2 < r_1$ denote the pool's target liquidity and interest rate bounds, which are set at deployment of the given pool and are immutable. Note that the reason we use an average rate in eq. (10) is to ensure that for loans that occur within the target liquidity range, their corresponding repayment amounts perfectly match the integral under the $rate(L_a)$ curve, such that splitting up a larger loan into multiple smaller ones doesn't yield any benefit.[24] Figure 2 provides an illustration of the associated interest rate curve, where we can see that if the pool liquidity falls below the lower liquidity bound L_1 then $rate(L_a)$ is defined by a hyperbola. If the liquidity is within the target range $[L_1, L_2]$ then $rate(L_a)$ follows a linear function, and if the liquidity exceeds the upper liquidity bound L_2 then $rate(L_a) = r_2$ is constant.

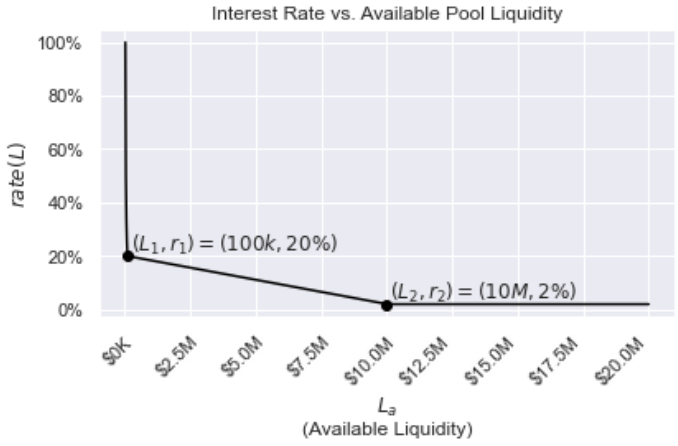


Figure 2: Example of how $rate(L)$ changes depending on the available pool liquidity L_a, assuming $(L_1, r_1) = (10^5, 0.2)$ and $(L_2, r_2) = (10^6, 0.02)$.

Generally speaking, the more liquidity there is (i.e., by LPs adding liquidity) the lower the interest rate, and conversely, the less liquidity there is (either by users borrowing or by LPs removing liquidity), the higher the interest rate. From the definition of eq. (11) we can see that rates can never fall below r_2, providing LPs a guaranteed lower bound at which their capital is lent to borrowers. However, $rate(L_a)$ isn't bound upwards to allow the market to find an interest rate equilibrium, even for highly volatile collateral assets.

[23] The rate r can easily be converted to an Annual Percentage Rates (APRs) given the tenor of the respective loan.
[24] Most loans are expected to be taken out within the target liquidity range, hence this property is only guaranteed

2.5.4 Rolling Loans

Borrowers can roll loans, i.e., instead of first repaying an old loan and then taking out a new one with the same collateral amount, they can do all of this in one transaction.[25] For this, borrowers only need to pay the difference between the previously owed repayment amount and the new loan amount they'd receive if they repledged their collateral. The logic according to which the new loan amount and repayment amount are determined is the same as previously described in section 2.5.2 and section 2.5.3. Note that for technical reasons, rolling a loan is only possible if the owed repayment amount exceeds the new loan amount, i.e., if $RepaymentAmount_{old} > y(x)_{new}$.[26] Moreover, note that when a borrower rolls their loan the associated repayment amount (not the difference $RepaymentAmount_{old} - y(x)_{new}$) becomes claimable for previously invested LPs.[27]

2.5.5 Risks for Borrowers

Generally speaking, borrowing from a MYSO v1 pool is less risky than liquidity provisioning. Nonetheless, borrowers should be aware of the following risks:

- Fixed repayment amounts: when a borrower takes out a ZLL they lock in a fixed repayment amount. This means that if the pool's borrow rates fall afterwards, the borrower will still have to pay the previously locked-in repayment amount. Moreover, borrowers should be aware that early repayment doesn't lead to a lower APR, i.e., the repayment amount is constant and independent of how long the borrow position was open.
- Opportunity costs: when a borrower pledges collateral into a pool, any associated rewards aren't automatically harvested for them (e.g., airdrops or the like).
- Rolling loans: if a borrower wants to roll a loan there's no guarantee that this will be possible (future pool liquidity might be insufficient) or that the new loan terms will be favorable (see section 2.5.4).

here.

[25] This allows to save on potential transfer fees because the collateral asset doesn't need to be transferred out of the pool and back in.

[26] This means that rolling loans always causes a reduction in the borrower's loan currency balance.

[27] Essentially, the repayment proceeds are 'pre-funded' by the pool's available liquidity.

2.6 Approvals & Peripheral Contracts

The MYSO v1 core allows users to approve other addresses to call certain functions on their behalf. This enables functional extensibility through peripheral contracts while keeping the MYSO v1 core lean and maintaining a good *separation of concerns*. For example, by virtue of approvals one can move functionalities related to wrapping and unwrapping of tokens into a peripheral router contract instead of having to incorporate this into the core.[28] Another example is automating liquidity provisioning through peripheral vault contracts, e.g., removing liquidity depending on collateral prices change (see fig. 3) or claiming and reinvesting loan proceeds, etc.

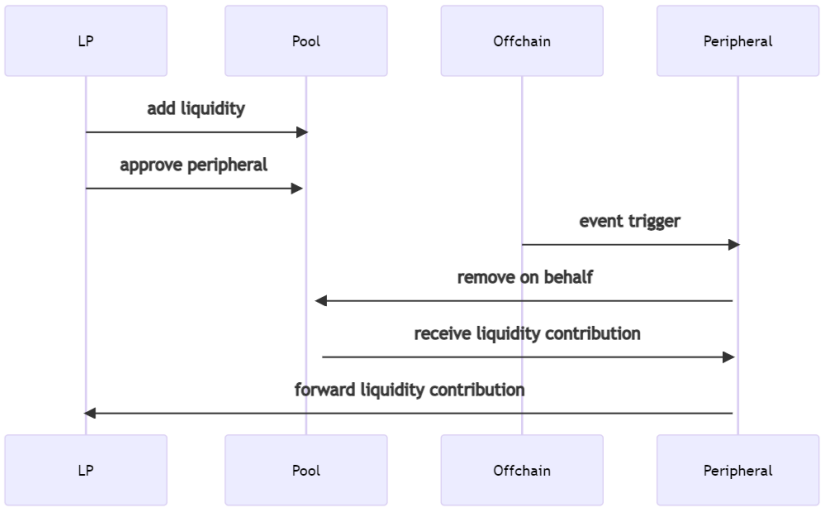


Figure 3: Example of how a peripheral contract could be constructed to automatically remove liquidity on behalf of a LP depending on some off-chain price data (which can be verified through an on-chain oracle).

To approve a third party address (i.e, an EOA or contract) users need to call `setApprovals()` and specify which functions the given third party shall be allowed to call on their behalf. Users can approve the following functions:

- Add liquidity: if approved, a third party can add liquidity on behalf of the given LP. This means the third party pays the liquidity contribution amount but the LP is credited with a corresponding amount of LP shares. Naturally, this approval is rather low risk because in the worst case, one can receive LP shares for free. Nonetheless, LPs should carefully assess the to-be-approved address because adding liquidity on their behalf can prolong their liquidity

[28] Note that in addition to wrapping and unwrapping a peripheral router contract can be used to minimize transaction overhead associated with individual pool approvals, or to incorporate flashswaps for leverage.

provisioning lock-up (see section 2.4.2).[29]

- Claiming: the approved third party can claim loan proceeds on behalf of a LP. LPs should carefully assess the to-be-approved address as it will have control over all unclaimed loan proceeds the LP is entitled to. Even if the third party is a non-custodial vault contract, it nonetheless is vital to assess the correctness of the corresponding contract to prevent any damage from potential ill claiming (e.g., skipping loan indices, see section 2.4.4).
- Remove liquidity: this allows a third party to remove liquidity on behalf of a LP. Users should be very cautious to provide this approval as the third party will essentially have full control over the LP's removable liquidity contribution.
- Borrowing: the approved third party can pledge collateral on behalf of the approving address. To be more specific, the approved party can then pay the collateral amount and receive the loan amount but the approving address will be registered as the loan holder and be able to repay and reclaim the pledged collateral.[30] Generally speaking, this approval is rather low risk because worst case the third party pays the collateral amount and receives the loan, while the approving address obtains a free option on the underlying collateral.
- Repaying: the approved third party can repay on behalf of the approving address. In this case, the third party pays the repayment amount and the collateral is sent to a `recipient` address that must either be the approved third party or the original loan owner.
- Rolling Over: if approved, a third party can roll over a loan on behalf of the approving address. In this case the third party pays the rollover cost (see section 2.5.4) and a new loan is taken out on behalf of the original loan owner. This means that the original loan owner retains their option to reclaim their previously pledged collateral; however, at a new repayment amount.

[29] I.e., the LP's liquidity provisioning lock-up period is reset to 120 seconds.
[30] This allows wrapping and unwrapping of tokens through a peripheral contract.

2.7 Pricing ZLLs

What are *fair loan terms* at which rational market participants would be willing to take out ZLLs and fund them? To answer this question we need a way to price ZLLs. We can do this by viewing them as swaps and pricing the embedded call option leg to derive a *fair APR*. To be more precise, every ZLL can be seen as two swaps (see fig. 4): (i) a swap between the borrower and the liquidity pool, and (ii) a swap between the liquidity pool and the LPs. Section 2.7.1 and section 2.7.2 describe these swaps in more detail.

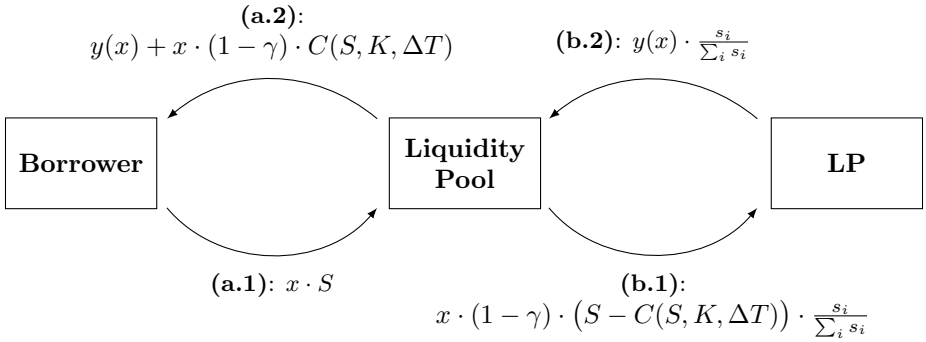


Figure 4: Zero-Liquidation Loan decomposed into two swaps, (a) and (b). (a.1) The borrower pledges collateral. (b.1) The LPs become entitled to the pledged collateral on a pro-rata basis and write an equal number of calls on it. (b.2) The LPs provide funding for the loan amount on a pro-rata basis. (a.2) The borrower receives the total loan amount as well as a call option to reclaim their collateral (net of fees).

2.7.1 Payoff from a Borrower's Perspective

From a borrower's perspective, a ZLL can be seen as a swap in which the borrower:

- pledges x collateral tokens;
- receives $y(x)$ loan tokens (see section 2.5.2);
- is allowed to reclaim their previously pledged collateral (net of any fees) if they pay the locked-in repayment amount prior expiry of the loan. Essentially, this represents a call option to buy $x \cdot (1-\gamma)$ collateral tokens from the pool for the *RepaymentAmount* (see section 2.5.3).[31]

In other words, taking out a ZLL as a borrower is equivalent to doing the following swap:

$$x \cdot S \xmapsto{borrow} y(x) + x \cdot (1-\gamma) \cdot C(S, K, \Delta T) \tag{12}$$

where

[31] Note that the call options are fully backed by the collateral the borrower pledges. No rehypothecation takes place, ensuring that borrowers can at all times exercise their calls.

x = amount of collateral tokens sent to the pool
S = price of the collateral
$y(x)$ = loan amount
γ = pool fees
$C(...)$ = value of call option
K = strike price of the call option, where $K = \frac{RepaymentAmount}{x \cdot (1-\gamma)}$
ΔT = tenor of the loan (or time to expiry of the call)

Figure 5 illustrates the borrower's payoff function before and after the loan is taken out. Initially, they hold the collateral asset and participate 1:1 in the value change of the asset. After the loan is taken out, the borrower holds call options on their collateral, and the loan amount.[32] This means the borrower retains their upside participation in the collateral while being protected from any downside risk, i.e., if the collateral price falls below the strike price K then the borrower doesn't have any incentive to repay and will leave the collateral in the pool.

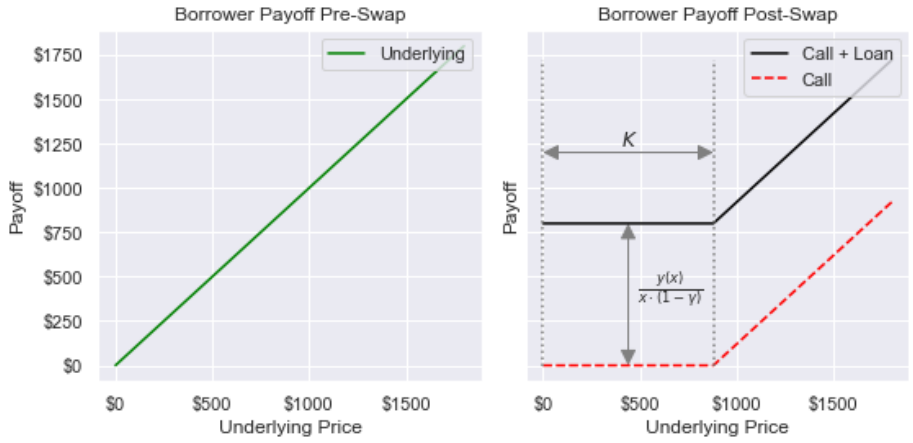


Figure 5: Illustration of a borrower's payoff before and after taking out a ZLL.

2.7.2 Payoff from a LP's Perspective

From a LP's perspective, funding a ZLL can be seen as a swap in which the LP:

- provides $y \cdot \frac{s_i}{\sum_i s_i}$ in pro-rata funding for the incoming loan;
- receives $x \cdot (1-\gamma) \cdot \frac{s_i}{\sum_i s_i}$ in pledged collateral;
- writes $x \cdot (1-\gamma) \cdot \frac{s_i}{\sum_i s_i}$ call options.

In other words, a LP who funds a ZLL does the following swap:

$$\alpha \cdot y \xrightarrow{LP} \alpha \cdot x \cdot (1-\gamma) \cdot \big(S - C(S, K, \Delta T)\big) \tag{13}$$

[32] Depending on the strike price, we can construct a swap where the borrower becomes indifferent between holding the underlying or a combination of calls and loan amount (see section 2.7.3).

where $\alpha = \frac{s_i}{\sum_i s_i}$ denotes the pro-rata share of the given LP i (see section 2.4.1).

Figure 6 illustrates the payoff diagram of a LP before and after funding a ZLL. Before, the LP simply holds their liquidity contribution amount, which is independent of the underlying price. After the LP funded a ZLL, their position turns into an in-the-money covered call, i.e., they're long in the underlying collateral but also short an in-the-money call option.[33]

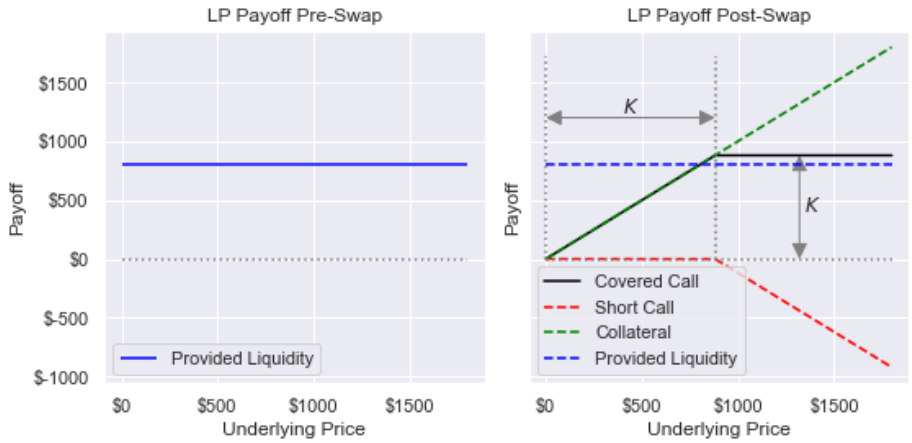


Figure 6: Illustration of a LP's payoff before and after funding a ZLL (assuming only one LP in pool and 100% utilization).

If at expiry of the loan the underlying price is above the strike price (where $K = \frac{RepaymentAmount}{x \cdot (1-\gamma)}$), then the borrower is naturally incentivized to repay and the LP will be able to claim a pro-rata share of the repayment amount. Otherwise, if the underlying price falls below the strike price then the borrower will not repay and the LP will be able to claim a pro-rata share of the collateral. Note that fig. 6 is a simplified illustration that assumes that there's only one LP and the pool utilization is 100%. In practice, however, there will be other LPs in the pool as well, meaning that LPs fund incoming loans only on a pro-rata basis, and similarly, LPs only gain exposure to a pro-rata share of an in-the-money covered call position.

2.7.3 Fair APRs

In a fair ZLL, both sides of eq. (12) have the same value such that neither borrowers nor LPs are better or worse off after the ZLL.[34] We can find a *fair strike* that makes the swap in eq. (12) have zero value by solving the following minimization problem:

$$K^* = \arg\min_K \Big(x \cdot S - y(x) - x \cdot (1-\gamma) \cdot C(S, K, \Delta T) \Big)^2 \tag{14}$$

For simplicity, we'll assume that ZLL borrowers holds a European style option instead of an American one and use the well-known Black-Scholes model as an approximation to price the call

[33] Or, equivalently, the covered call can be seen as a short synthetic put.
[34] Or, equivalently one could also use eq. (13) here.

[2]:[35]

$$C(S, K, \Delta T) = S\Phi(d_1) - Ke^{-r(\Delta T)}\Phi(d_2) \tag{15}$$

where

$$\begin{aligned} d_1 &= \frac{ln(S/K) + (r + \sigma^2/2)(\Delta T)}{\sigma\sqrt{\Delta T}} \\ d_2 &= d_1 - \sigma\sqrt{\Delta T} \end{aligned} \tag{16}$$

and

r = risk free rate
σ = price volatility of the collateral
Φ = Gaussian cumulative distribution function

Based on eq. (14) it is easy to infer the corresponding *fair APR*:

$$APR = \left(\frac{x \cdot (1 - \gamma) \cdot K}{y(x)} - 1 \right) \cdot \frac{1}{\Delta T} \tag{17}$$

LPs have positive expected value if they fund ZLLs at a higher APR than the *fair APR*, and conversely, borrowers have positive expected value if they borrow at a lower APR than the *fair APR*.

Figure 7 provides a numerical example for *fair APRs* for different LTV and tenor combinations. One can see that the APR increases for higher LTVs as well as longer tenors, which fits the intuition that LPs would demand a higher compensation when underwriting loans with higher (and longer) collateral price risk [17]. Similarly, underwriting loans on more volatile collateral assets should entail a higher APR compensation. By comparing fig. 7 and fig. 10 we indeed can see that this is the case, i.e., the *fair APRs* are overall higher in the higher volatility case and that the differences between the individual LTV and APR combinations become more pronounced.

[35] If one wants to take into account the American style exercise feature of ZLLs one can, for example, use the Longstaff-Schwartz model [10] instead of Black-Scholes.

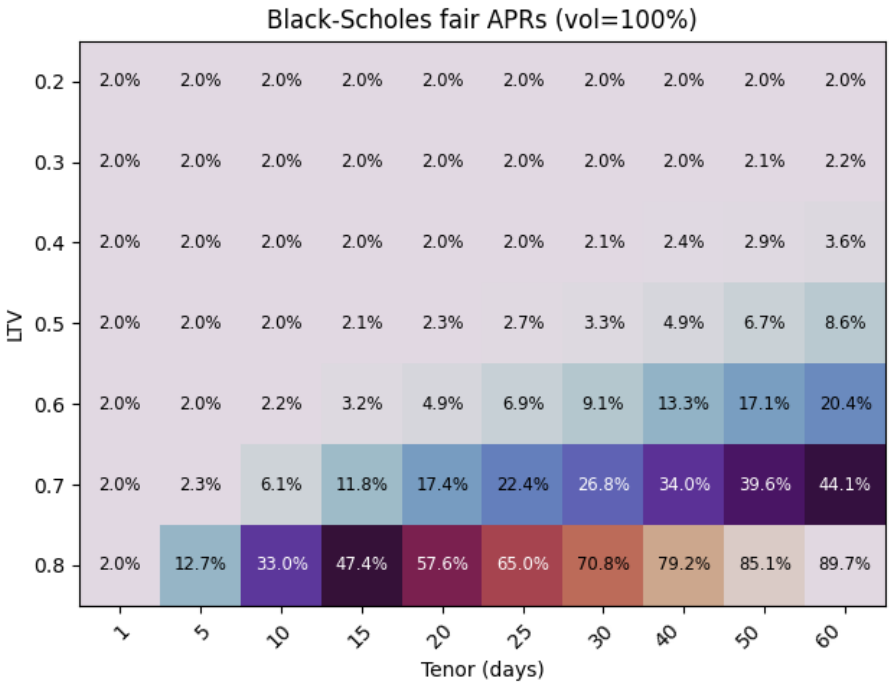


Figure 7: Numerical example of fair APRs across different LTV and loan tenor combinations.

3 Closing Remarks

By introducing ZLLs, MYSO v1 allows to radically simplify the user experience for borrowers and provide LPs with a conservative yield enhancement strategy in the form of physically settled in-the-money covered calls[36]. From a systemic point of view, MYSO v1 can help mitigate some of the key risks associated with liquidation-centric credit markets, such as cascading liquidations, externalities stemming from liquidation-related MEV, and oracle manipulations. What makes MYSO v1 unique is that its pools are open-ended, allowing borrowers to take out loans with a constant tenor[37] and that liquidity provisioning is continuously possible. In addition, the protocol is trust-minimized and operates without relying on any trusted third parties. Moreover, liquidity pools are isolated from one another such that a bad collateral asset in one pool cannot compromise the integrity of other pools. Despite the aforementioned strengths of the MYSO v1 core, it also comes with limitations that require further work, in particular w.r.t. potential liquidity fragmentation for certain asset pairs, possible claiming overhead and the need for active liquidity provisioning. Nonetheless, we hope that the MYSO v1 protocol can meaningfully contribute to the further development of DeFi.

[36] Or, equivalently, a short synthetic put.

[37] This stands in contrast to alternative liquidation-less borrowing solutions in which pools typically have a fixed expiry date such that the offered loan tenors get shorter over time.

References

[1] L. Baker. Paradigms for on-chain credit. https://jumpcrypto.com/paradigms-for-on-chain-credit, Aug 2022.

[2] F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 1973.

[3] DefiLlama. Dashboards - DeFi overview. https://defillama.com, October 2022.

[4] S. Deshmukh, S. Warren, and K. Werbach. Decentralized finance (defi) policy-maker toolkit. https://www3.weforum.org/docs/WEF_DeFi_Policy_Maker_Toolkit_2021.pdf, June 2021.

[5] Eurex. Credit, concentration wrong way risk. https://www.eurex.com/ec-en/services/risk-management/credit-concentration-wrong-way-risk. visited on 2022-10-19.

[6] Galaxy Digital Research. Mev: How flashboys became flashbots. https://assets.ctfassets.net/h62aj7eo1csj/NYw2dyWSNeD4KEHP0R4c1/c997bba68ecdc845306eb1ff8bdf5032/GLXY_-_MEV_Whitepaper.pdf, Jan 2021.

[7] O. Godbole. Ethereum money markets see record liquidations as ether tanks; makerdao revenue surges. https://www.coindesk.com/markets/2022/01/25/ethereum-money-markets-see-record-liquidations-as-ether-tanks-makerdao-revenue-surges, Jan 2022.

[8] S. Haig. Solend to not freeze whale's $216m account after move decried as 'opposite of defi'. https://thedefiant.io/solend-whale-proposals-freeze-account, June 2021.

[9] F.-S. Lhabitant. Derivatives in portfolio management: Why beating the market is easy. https://risk.edhec.edu/sites/risk/files/EDHEC_WhyBeatingTheMarketIsEasy.pdf, Nov 2000.

[10] F. Longstaff and E. Schwartz. Valuing american options by simulation: A simple least-squares approach. *The Review of Financial Studies*, 2001.

[11] T. Mackinga, T. Nadahalli, and R. Wattenhofer. Twap oracle attacks: Easier done than said? https://eprint.iacr.org/2022/445, Apr 2022.

[12] L. Odier. Lombard loans. https://www.lombardodier.com/home/private-clients/lombard-loans.html.

[13] K. Qin, L. Zhou, P. Gamito, P. Jovanovic, and A. Gervais. An empirical study of defi liquidations: Incentives, risks, and instabilities. https://arxiv.org/pdf/2106.06389.pdf, Jun 2021.

[14] Ruler Protocol. A market driven lending platform that provides non-liquidatable and fungible loans. http://rulerprotocol.com.

[15] A. Sardon. Zero-liquidation loans: A structured product approach to defi lending. https://arxiv.org/abs/2110.13533, October 2021.

[16] A. Sardon. Claiming intervals. https://github.com/mysofinance/notebooks/blob/main/claiming-intervals.ipynb, Oct 2022.

[17] A. Sardon. Thinking about fair APRs for profit and fun. https://github.com/mysofinance/notebooks/blob/main/thinking-about-fair-aprs-for-profit-and-fun.ipynb, September 2022.

[18] The Options Industry Council. An income strategy: Let's sell some options. https://www.fidelity.com/bin-public/060_www_fidelity_com/documents/learning-center/lets-sell-some-options.pdf, 2017.

[19] Timeswap. Like uniswap, but for lending & borrowing. https://timeswap.io.

[20] Vendor Finance. Non-liquidatable, fixed-rate, perpetual loans. https://vendor.finance.

Appendices

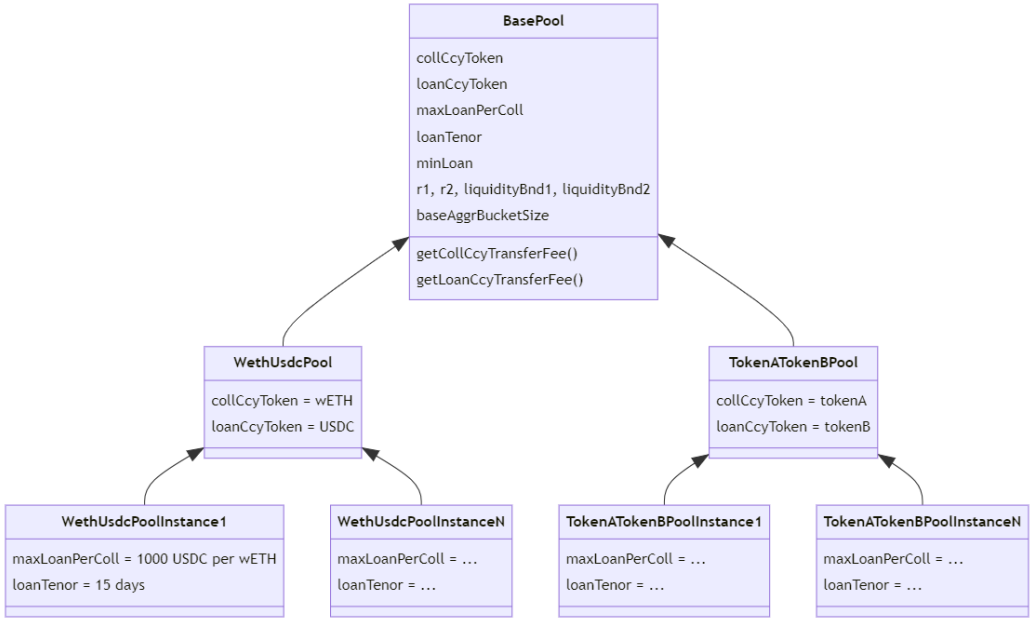


Figure 8: Base pool contract, derived asset-pair specific implementation contracts and pool instances.

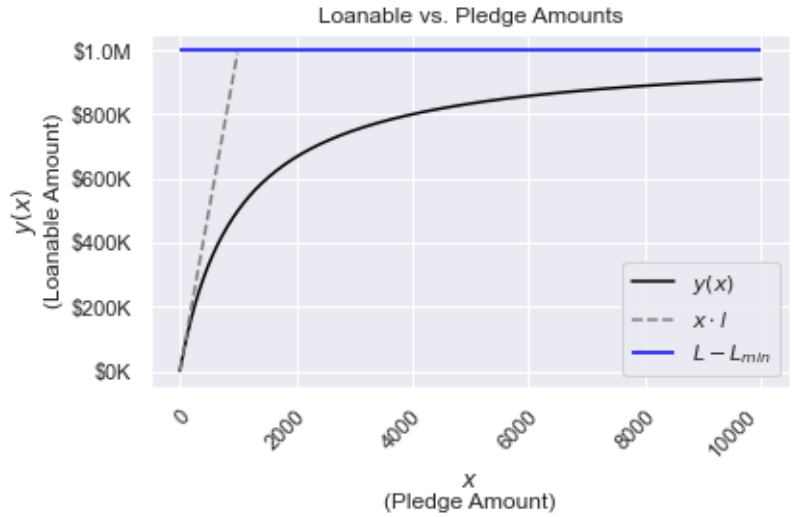


Figure 9: Example of how the loan amount $y(x)$ changes for different collateral amounts x, assuming $l = 1000$, $L = 10^6$, and $L_{min} = 1$.

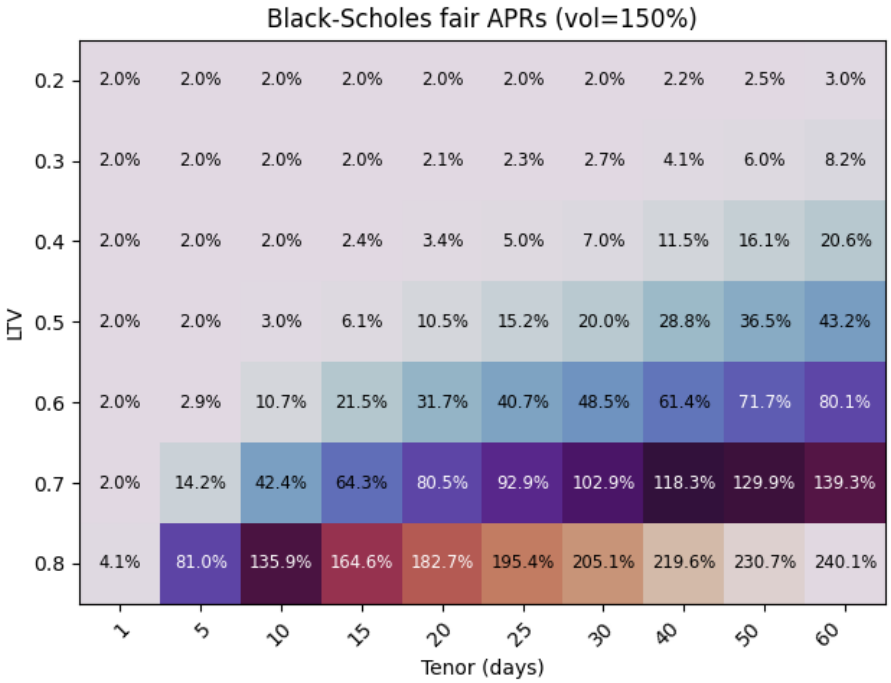


Figure 10: Similar to fig. 10, but with a higher assumed collateral price volatility.