Aave V3 Risk Parameter Methodology

Yonatan Haimovich
haimo.yonatan@chaoslabs.xyz

Omer Goldberg
omer@chaoslabs.xyz

February 2023

Abstract

The Aave protocol enables overcollateralized borrowing, and it relies on *liquidators* to pay back loans on behalf of accounts that are overleveraged. To limit the negative externalities of lending meltdowns, the protocol has risk parameters that define account leverage and liquidation bonuses. Here, we put forward a methodology for optimally selecting these risk parameters. At the heart of this methodology is Chaos Labs' high-fidelity agent-based simulation platform, which we use to simulate on-chain debt and liquidation behaviors. We then use the agent-based simulation platform to estimate the protocol's value at risk and recommend risk parameters that yield an acceptably low value at risk.

1 Introduction

AAVE is a decentralized finance (DeFi) protocol built on top of the Ethereum blockchain that utilizes smart contracts to facilitate financial transactions in a trustless manner. The protocol allows users to lend and borrow cryptocurrencies.

On November 10th, 2022, Chaos Labs was engaged by the Aave community to provide Risk Management services for it's V3 deployments across the following networks: Ethereum, Arbitrum, Optimism, Fantom, Polygon, and Avalanche.

The Chaos Labs' Risk Management platform and services are centered around market risk and economic security. The protocol faces additional risk vectors (smart contract risk, regulatory risk, etc.), as we will elaborate on below, but these are out of scope for this assessment and Chaos engagement. Chaos Lab's utilizes a novel, proprietary cloud-based simulation platform to assess market risk and to produce optimized risk parameter recommendations for the Aave community.

1.1 Paper Goals

In this paper, we provide an agent-based simulation risk framework that Chaos Labs will utilize to determine risk parameters for the Aave protocol. We specify our methodology for finding optimal risk parameters that juggle user experience and protocol risk, and we use Chaos Labs' simulation platform to estimate protocol losses. This paper is a technical deep dive into the process that we will use to recommend Aave protocol parameters.

1.2 Aave Overview

Aave is based on the concept of money markets, which are financial markets that facilitate the borrowing and lending of funds. In traditional money markets, borrowers and lenders interact directly through intermediaries, such as banks or financial institutions. In contrast, Aave allows borrowers and lenders to interact directly through smart contracts without intermediaries.

Aave allows users to earn interest on their digital assets by lending it to borrowers, who can use the borrowed funds for various purposes, such as margin trading or making purchases. On the borrowing side, AAVE allows users to borrow digital assets against collateral comprised of other digital assets. When a borrower takes out a loan, they must pay interest on the loan to the protocol. The interest rate is determined by the supply and demand for different assets on the AAVE platform, and it may be fixed or variable. For the duration of the paper, we will focus on a high-level understanding of the Aave protocol, focusing on the areas that pertain to the presented statistical framework presented, as there are ample resources for understanding the granular level details of Aave V3.

Aave Balance Sheet. Aave maintains a balance sheet of assets and liabilities. Assets represent deposited collateral and collected fees, while liabilities represent loans. Liabilities are liens, giving the protocol the right to the user's deposited collateral. Liens in Aave are on-call, meaning they have no fixed term or length. Borrowers can attempt redemption anytime, granted they meet protocol requirements. Liens are closed when a borrower returns the borrowed asset in addition to any fees accrued throughout the loan period.

All Aave loans are over-collateralized at the time of origination, with a buffer, attempting to secure the protocol against market fluctuations and high volatility. Borrow positions on AAVE are overcollateralized using a dynamic collateralization system, which means that the value of the collateral is regularly monitored and adjusted based on the changing market conditions. We aim to ensure that Aave's assets are always more significant than its liabilities. Aave v3 provides numerous risk parameters that dictate the economic security of the protocol. This paper will focus solely on the parameters essential for ensuring an optimized liquidation mechanism.

1.3 Aave Liquidations Overview

The Aave liquidation mechanism enables the protocol to provide lending between assets whose relative prices vary over time. When a borrower's collateral assets goes down in price relative to the assets they are borrowing, the protocol must rebalance their positions to ensure that they remain over-collateralized.

This is done via liquidations. For these liquidations to operate smoothly, the protocol must set values for a number of parameters, which we detail below.

1.4 Aave's Liquidation Risk Parameters

Loan to Value (LTV). LTV is a percentage that represenets a ratio defining the maximum amount of assets that can be borrowed with a specific asset as collateral. This ratio dictates the collateral value needed to open a borrow position. For example, if LTV for ETH is 0.7, then for every 1 ETH worth of collateral, borrowers can borrow 0.7 ETH worth of digital assets.

Liquidation Threshold (LT). The liquidiation threshold is a parameter that the protocol sets for every asset that can be used as collateral. The LT value can be between 0 and 1; a larger value means that each unit of the the collateral counts higher toward the account's health (see 'Health Score' section below).

Health Score. Each position's risk is monitored using a health factor. The health factor determines the risk of a particular lending position. A higher position health factor indicates that a lending position is relatively safe, while a lower position health factor indicates that a lending position is at risk of being liquidated. Once a position's health factor is lower than 1, that position is eligible for liquidation. The AAVE protocol may automatically sell a portion of the collateral to repay the loan and protect the lender's interests. Health factor is calculated as:

$$H_f = \frac{\sum_{i \in A}(\text{Collateral}_i \text{ in ETH}) \cdot (\text{LT}_i)}{\text{Total Borrows in ETH}},$$

where A is the set of all Aave assets.

Liquidation Penalty (LP) The Liquidation Penalty (LP) is a fee involved in purchasing a former loan, taken on any given on-chain collateral, which is now liquidated by the protocol. The fee goes to the protocol as a means of buying into a former loan. The reason why loans get liquidated is they pass the Liquidation Threshold (LT), thus no longer being in the hands of the original asset holder.

Aave Liquidation Example. Suppose an Aave user, John, has in his possession asset `A` and wants to use it as collateral to borrow asset `B`. John goes to open a position on Aave, and at the time his position is opened, the following is true:

1. Asset `A` is worth \$1000 USD, has a loan to value (LTV) ratio of 80%, and liquidation threshold (LT) of 85%, and liquidation bonus (LB) of 5%.
2. Asset `B` is worth \$1 USD.
3. John deposits 1 `A` as collateral into Aave, and he borrows 500 `B`. This is an initial LTV of 50%, which is acceptable, since it is below the 80% maximum LTV.

Suppose that some time after John opens his position, the USD value of `A` begins to drop, while the USD value of `B` remains stable. Once the value of `A` drops enough, such that (Collateral Value) $\cdot$ (LT) $<$ (Borrow Value), i.e. once John's account has a health less than 1, then John's account is eligible for a liquidation. Suppose that this is the case, and that asset `A` drops to a price of \$560, rendering John's position undercollateralized. Then one of the following two scenarios may occur.

1. John's account is liquidated. Suppose, for example, that the liquidation seizes half of John's collateral. Then they would be seizing 0.5 `A` at a value of \$280, and they would be obligated to pay back $(1 - \text{LB})$ times that amount of value in the borrowed asset, i.e. \$266 of borrowed asset value. Since `B` is valued at \$1, we would have that John's new position Aave position is 0.5 `A` of collateral and $500 - 266 = 234$ `B` of borrows. John's account's new health score is $\$280 \cdot 0.85/\$234 = 1.017$, which is greater than one, and thus no longer liquidatable. This is a standard liquidation.
2. John's account is not liquidated. Then the protocol may incur bad debt if asset `A` moves even

lower in price, relative to asset B. That is, if there are no liquidations, then the protocol may incur debt equal to (Borrow Value) − (Collateral Value).

1.5 Aave's Protocol Risk Vectors

Aave Companies has created a risk scale to help the Aave community evaluate the risk of different assets. The scale uses a grading system ranging from A+ (least risky) to D- (risky). This scale is considered when listing new assets and their respective configurations; see table 1 for details.

	Days	Transactions	Holders	Permission	Market Cap	Average Volume	Normalized Volatility
	>	>	If permissionless + holders > 3 down for some control 4 for centralized		>	1M & 3M>	1M, 3M & 6M < with 1 outlier < then next
A+	730	500,000,000	100,000,000	ETH	14,900,000,000	20,000,000,000	0.005
A	730	100,000,000	1,000,000	Centralized but regulated	10,000,000,000	10,000,000,000	0.015
A-	730	10,000,000	500,000		5,000,000,000	2,500,000,000	0.025
B+	365	1,000,000	100,000	Permissionless	500,000,000	700,000,000	0.038
B	365	350,000	50,000		250,000,000	500,000,000	0.051
B-		350,000	25,000	Centralized real time PoF	100,000,000	100,000,000	0.064
C+	365	100,000	10,000		50,000,000	20,000,000	0.079
C				Some controlling functions		5,000,000	0.094
C-		100,000	2,500		30,000,000	1,000,000	0.109
C-	365	25,000		Some control over minting	15,000,000		
D+		25,000		Centralized with audit	10,000,000	500,000	0.124
D	365			Centralized	5,000,000	250,000	0.139
D-				Opaque			

Table 1: The risk matrix utilized when listing new assets. This table provides a qualitative assessment framework that Aave community can utilize when deciding if an asset should be listed; for assets that are listed, it can be utilized to help to qualitatively determine the risk parameters for the asset. Credit: Aave Companies.

Smart Contract Risk. Smart contract risk refers to the technical security of the underlying code of an asset. It is meant to reflect the maturity and resilience of a piece of code. This is hard to measure objectively; therefore we use proxies such as: the number of days and transactions the smart contract has been in use, community engagement, and development stats.

All assets listed on Aave should have undergone thorough audits by reputable auditors. Assets with a high level of smart contract risk are considered extremely risky as collateral and should only be onboarded with strict risk mitigation measures in place. [1]

Counter-Party Risk. Counter-party risk pertains to the governance of a given asset and degree of centralization. It is assessed based on factors such as: the level of decentralization of the asset's governance, the number of parties that control the asset's protocol, the number of holders of the asset, and the level of trust in the entity, project, community, or processes associated with the asset. [2]

Market Risk. Market risks in the protocol are dynamic and impacted by the pool's size and supply and demand oscillations. Constant assessments of average daily volume, volatility, and market capitalization aim to mitigate market risk.

Liquidity Risk. Primarily based on on-chain liquidity and trading volume, liquidity is critical for the liquidation process. Liquidation risks are mitigated via liquidation parameters (i.e., the lower the liquidity, the higher the liquidation incentives). This article focuses on presenting a framework for liquidation parameter optimizations. Supply and borrow caps are also critical for risk mitigation, but they are out of the scope of this paper.

[1] For more information, please visit Aave's formal defintion here.
[2] For more information, please visit Aave's formal defintion here.

Volatility Risk. As seen in our example above, price volatility can negatively affect collateral value. Asset volatility is a crucial factor in determining its respective risk parameter configuration. The least volatile currencies are stablecoins, followed by ETH and BTC. High volatility will yield low LTs and LTVs, while low volatility will allow us to be more risk-on.

Economic Security Risk. Economic security is a new vertical with increasing relevance and importance. We loosely define economic security as exploits or attacks that manipulate prices.

1.6 Simulations

We build simulations that interface directly with blockchain applications or emulate on-chain protocols. These simulations utilize mathematical assumptions based on relationships between protocol parameters and exogenous variable. A collection of these assumptions comprises a model used to understand better how the corresponding protocol functions.

Why simulations? If the relationships that compose the model are straightforward, it may be feasible to use closed-form mathematical methods or equations to acquire *precise* information; this is called an *analytical* solution. However, most real-world systems' vast complexity renders analytical models non-realistic; we turn to simulations for these models. In a *simulation*, we use a computer to evaluate a model *numerically* to estimate the model's desired actual characteristics.

Simulation Challenges. Many of the most worthwhile systems of the model need to be simplified and require simulations. Since this is the case, why aren't simulations a more popular tool? Building high-fidelity simulations is an arduous task. When modeling DeFi protocols, the job is even more difficult as this is an emerging field with a short history and relatively small data sets. While the challenges vary, we can bucket the challenges broadly into two verticals; precise modeling and scale. We will describe the system we have architected to handle the unique scale challenges below. Simulation modeling and the statistical framework used to produce recommendations are the paper's focus, and we'll leave that discussion for the next part of the paper.

Parallel Simulations and High Level System Architecture. Chaos supports two types of simulations:

1. Chaos EVM Simulations - These execute on a proprietary, python-based Chaos-constructed EVM.
2. "On-chain" Simulations - These execute on the rEVM on an optimized version of Anvil.

Chaos has developed a novel, hybrid approach that leverages the advantages of both simulation solutions while providing accurate risk analysis and parameter recommendations.

Chaos EVM Simulation. The Chaos EVM is a python-based agent-based simulation environment. Every simulation initializes with a data-sync phase where new or historical mainnet data is extracted and loaded. This data includes but is not limited to the following: account portfolios and balances, agent elasticity, protocol liquidity, and risk parameters. Chaos EVM simulations are highly performant, showing 250x latency and CPU improvements over their on-chain (below) counterparts. Chaos EVM simulations are compelling, allowing us to narrow our search space significantly and quickly. Once we discover an optimal risk parameter configuration, we can backtest them with the on-chain simulations.

On-Chain Simulation. DeFi applications execute on blockchains that offer data-rich, transparent environments. The complete visibility and transparency of DeFi applications significantly dwarf their traditional finance counterparts, where financial applications and instruments execute in opaque, private settings.

The ability to fork blockchains gives us transparency and data readability out of the box. On-chain simulations are initiated and orchestrated by a simulation executor that utilizes a novel agent and scenario model to interact with a dedicated blockchain fork. Two primary components comprise on-chain simulations: (1) an optimized, customized Anvil fork that supports our intensive high throughput requirements and simulation length beyond what is ordinarily possible; and (2) a simulation engine that orchestrates the different actors and models interacting with the Fork can collect data points and metrics for diagnosis.

With an rEVM environment and executor orchestrator toolkit, we can granularly control the state of any blockchain fork. To summarize, blockchain simulations are unique and compelling for several reasons. Forks allow us to obtain a complete snapshot of the runtime environment at a given time (block height). On-chain agents can interface with native blockchain protocols identically to how they will be interfaced in

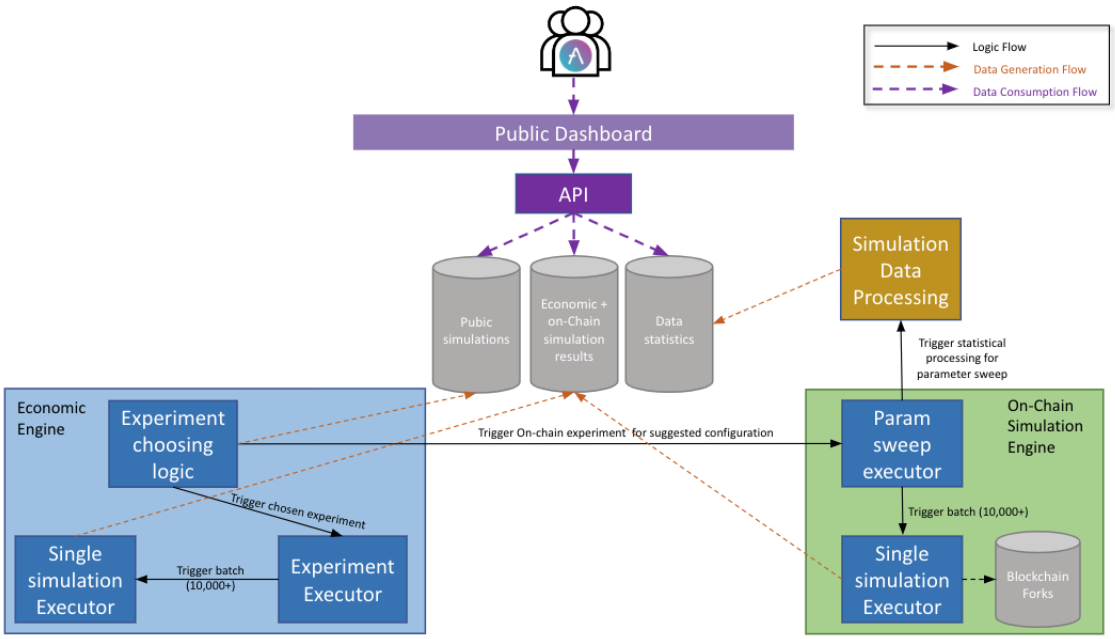


Figure 1: An architecture diagram of the simulation engines utilized to search for and provide bounds on Aave's risk parameters.

production. Creation of tools for simulation analysis with similar interfaces to tools used in production, such that users can explore data in familiar ways. For example, imagine a block explorer for a private blockchain instance used to run a simulation. The advantages of on-chain simulations are clear. However, they come with a cost. On-chain simulations must execute all EVM operations. Even in an optimized environment, on-chain simulations are comparatively resource-heavy and increase latency by orders of magnitude.

Chaos Cloud Architecture. Estimating value at risk requires a massive scale. Our proprietary cloud solution can parallelize thousands of executors and forks across thousands of machines with real-time data processing to yield a statistical analysis. Putting this all together, we have the following:

1. The initial parameter exploration executes on the proprietary Chaos EVM. This significantly narrows the search space and ultimately yields a parameter recommendation.

2. The parameter recommendation is backtested on an on-chain simulation. The results of on-chain simulations are shared with the community for verifiability and transparency.

See figure 1 for an illustration of the simulations architecture.

Agent-Based Simulation. An *agent* is an autonomous "entity" that can sense its environment, including other agents, and utilize this data in decision-making. Agents can learn and adapt their behaviors over time.

In the context of DeFi simulations, agents emulate users and can be traders, arbitrageurs, liquidators, borrowers, liquidity providers, and more. An *agent-based simulation* is a simulation where the entities (agents) interact with other entities and adapt to their environment. In the Aave example, agents - borrowers, liquidators, and arbitrageurs - interact with Aave and DEXes as prices change, for example.

Monte Carlo Simulations. A Monte Carlo simulation takes the uncertain variable and assigns it a random value. The model is then run, and a result is computed. Simulations are repeated while assigning many different values to the variable. Once the simulation is complete, the results are aggregated and averaged to arrive at an estimate. In the case of Aave, each simulation utilizes a randomly-generated price trajectory and we use an agent-based model to simulate the response that this invokes from borrowers and liquidators of the protocol.

Why Agent-Based Monte Carlo Simulations? In many financial applications, such as measuring the distribution of a static portfolio's value at a time in the future, the price dynamics fully describe the

portfolio's value. In contrast, our problem is to simulate the protocol losses that the protocol would incur, and this quantity is dependent not only on price dynamics, but also on the interactions of borrowers and liquidators. Thus, for a given simulation, we update the actions that borrowers and liquidators would do at that timestep. So long as our simulations are not statistically biased, we can use these simulation results to estimate the expected value of protocol losses. We now provide a more in-depth description of our methodology used to construct these Monte Carlo agent-based simulations, as well as the statistical testing framework we use to determine the value at risk for particular parameter configurations.

2 Risk Parameter Testing Methodology

To test whether a set of risk parameters, we determine the protocol's value at risk in a scenario where they have those risk parameters.

Definition 2.1 (Value at Risk)**.** *Value at Risk (VaR) is the 99th percentile of the protocol losses that accrue due to below-water accounts over the course of 24 hours.*

We run simulations to estimate the VaR, and we only recommend parameters that will keep VaR below some pre-specified upper bound K.

2.1 Estimating Value at Risk

To estimate the value at risk, we take the following steps.

1. Run a set of parameters, s, for 10,000 iterations of our simulation. Calculate the 99th percentile loss across those 10,000 iterations.
2. Run another 10,000 simulations with the same parameters, and calculate the 99th percentile loss of the 20,000 total iterations. If the difference between the 99th percentiles is less than some pre-specified noise bound, ε, then we say that the value-at-risk is the 99th percentile over all of the simulations run. Otherwise, we repeat this step until we hit ε convergence.

This approach can be understood as running simulations until we reach a convergence on a value for the 99th percentile of losses. We base this off of the approach taken by risk desks at top banks, whereby they run simulations until a particular convergence bound is met. Alternatively, one could utilize a statistical testing framework to determine the probability of passing a particular VaR, which we provide in Appendix B.

VaR Calculation Example Suppose we are calculating VaR using 10000 simulations of bad debt accrual over a 24 hour period for a certain parameter set s. Denote each batch of simulations as a round. The result of each simulation is the total bad debt accrued over the specified period, and these results are ranked in ascending order. The final VaR estimate for round i is then the 9900th result in the ordered list, denote this R_i

We compute R_i starting at $i = 0$. For each subsequent round we compute R_i and check the convergence condition $|R_{i-1} - R_i| < \varepsilon$. If the convergence condition is passed, we return R_i as our final VaR estimate for the parameter set s.

We now provide more depth on the methodology used to simulate protocol losses.

2.2 Simulating Protocol Losses

When estimating the VaR above, we abstracted a "simulation" as some stochastic function $L : S \to 0 \cup \mathbb{R}^+$ that simulates the protocol losses. We now describe in greater detail how we compute L.

To compute L, we first compute a realization of price trajectories for each of the assets that can be supplied on Aave. We then utilize these price trajectories to estimate the on-chain DEX liquidity that would be present, given the price trajectories. Once the price and liquidity trajectories are determined, we utilize an agent-based model, with lender-borrower agents and liquidator agents.

2.2.1 Assumptions

1. Look-forward period. We only look forward by a single day worth of blocks.

2. Price correlations. We assume that asset prices are correlated, and we use a historical window of 30 days to find this correlation.

3. Price process. We assume that the log-prices obey a (1,1) GARCH process on short timescales (i.e. minute-by-minute), and that they obey a normal distribution on longer timescales (i.e. hourly and longer). We have validated these assumptions visually by analyzing (partial) autocorrelation plots.

4. Exogenous prices. Price trajectories are not affected by liquidations nor other actions taken by agents in the simulation. With that said, the Aave protocol existed at the time that our historical price trajectories were sampled, and so our price trajectory distributions should account for price-liquidation feedback loops, even though we do not explicitly model this relationship.

5. Static liquidity. We assume that over the next day, decentralized exchange liquidity will remain relatively constant.

6. At most one liquidation per account per block. We assume that liquidators do not execute more than one liquidation on each account. Any liquidation that occurs on an account makes it impossible for other accounts to liquidate.

7. Only DEX liquidity. We assume that liquidators only utilize on-chain DEX liquidity to perform liquidations. We have seen empirical evidence that this is correct. This assumption is supported by empirical on-chain evidence, where we see that many liquidations utilize on-chain spot liquidity to execute their trades. Although this assumption will become less valid as on-chain lending markets become more developed, this assumption is currently accurate, and this assumption errs on the side of lower-risk parameter recommendations.

8. One-of-n non-collusion. We assume that there is *at least* one liquidator that does not collude with other liquidators and that maximizes their profits from liquidations. Specifically, there would not be scenarios where all liquidators decide to wait to liquidate an account's collateral, in hopes that the reward will be larger if they wait until the next block. Due to the accessibility and low profit margins that are arise empirically with on-chain lending markets, we believe this assumption is fair.

9. Responsive liquidations. If an account does not have a liquidation within c blocks of its health going below zero, then its loans should be marked as bad debt. We set $c = 2$ blocks in our analysis. This assumption errs on the side of lower risk parameter recommendations.

10. Finite time-to-liquidation. An account's collateral value will be liquidated by the protocol at a time $c_{protocol}$ blocks after an account's loans are marked as bad debt. This is a total of $c + c_{protocol}$ blocks after the account's health goes below zero. This assumption comes from the idea that the protocol will get rid of its bad debt at some point in the future [3].

11. Not statistically testing black swan events. We do not attempt to reach statistical significance on the probability of black-swan events, since black-swan events are categorically immune to statistical testing. Instead, we perform a separate "stressed VaR" methodology – which we will describe in greater detail in a subsequent paper – that utilizes the Chaos EVM to model example black-swan events. Modeling events like these is useful for qualitatively determining how bad the fallout would be for Aave in the case of spectacular events, such as a large stablecoin depeg, or a mass exodus of on-chain liquidity.

[3] Note: as Ori pointed out, a mechanism for this does not exist in the protocol today, and thus this assumption isn't particularly fair.

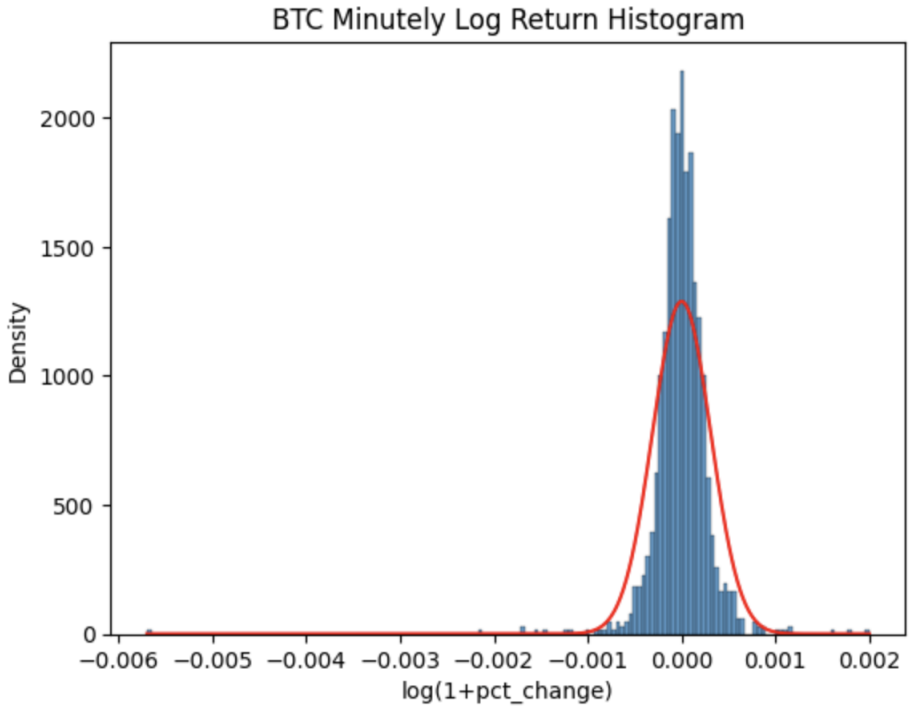


Figure 2: A histogram of BTC minutely log returns from 11/19/2022 to 11/20/2022. The red line is a normal distribution with the same mean and standard deviation as the log returns. That is, BTC minutely returns are lognormal.

2.2.2 Computing price trajectories

We use a discrete-time random walk stochastic processes to generate price trajectories of assets.

Geometric Brownian Motion. The canonical model for generating price trajectories is geometric brownian motion (GBM). In a simple GBM model, the price (S) changes according to a stochastic differential equation: $dS_t = S_t(1 + \mu dt + \sigma dW_t)$, where W_t is a Brownian motion, σ is a volatility scaling parameter, and μ is a forcing parameter. This stochastic DE can be integrated with Itô's formula to get that $S_t = S_0 \cdot e^{(\mu - \frac{\sigma^2}{2})t + \sigma W_t}$. This analytic solution makes it possible to sample a price trajectory's outcome without running each timestep.

Issues with GBM. Although GBM is a popular model in finance literature, it does not hold true in practice at short time intervals. We often see erratic price jump behavior at short timescales, along with short bursts of high or low volatility, which are not reflected by the constant-σ GBM model. We also see times of new information that lead to a short-term jump or drop in returns, which is also not reflected in the constant-σ GBM model. To correct for these, we forego the GBM modeling and instead utilize variable volatility price trajectories.

For a clear evidence of the non-normality of this data, see the distribution of price trajectories in figure 2. We see that the distribution of returns contains outliers that are many standard deviation events. To model returns as following a GBM process, we would either need to entirely ignore long-tail events by using the real σ, or we would need to make these long-tail events possible by increasing the σ, which would overestimate the volatility of returns in the majority of cases. Clearly, a GBM with constant σ is not the right model for returns behavior, and this is of particular importance due to the fact that long-tail returns have a major impact on the health of Aave accounts.

Not only is the returns distribution fat-tailed, but it is also autoregressive. Figure 3 shows a plot of the minutely log returns for a period of over 1 month, during which there were a number of high-volatility events. In some of these periods, the median across 100 minutes elevates as well. This demonstrates that a minute with a large absolute return is likely to be close in proximity to other high absolute return minutes. If we

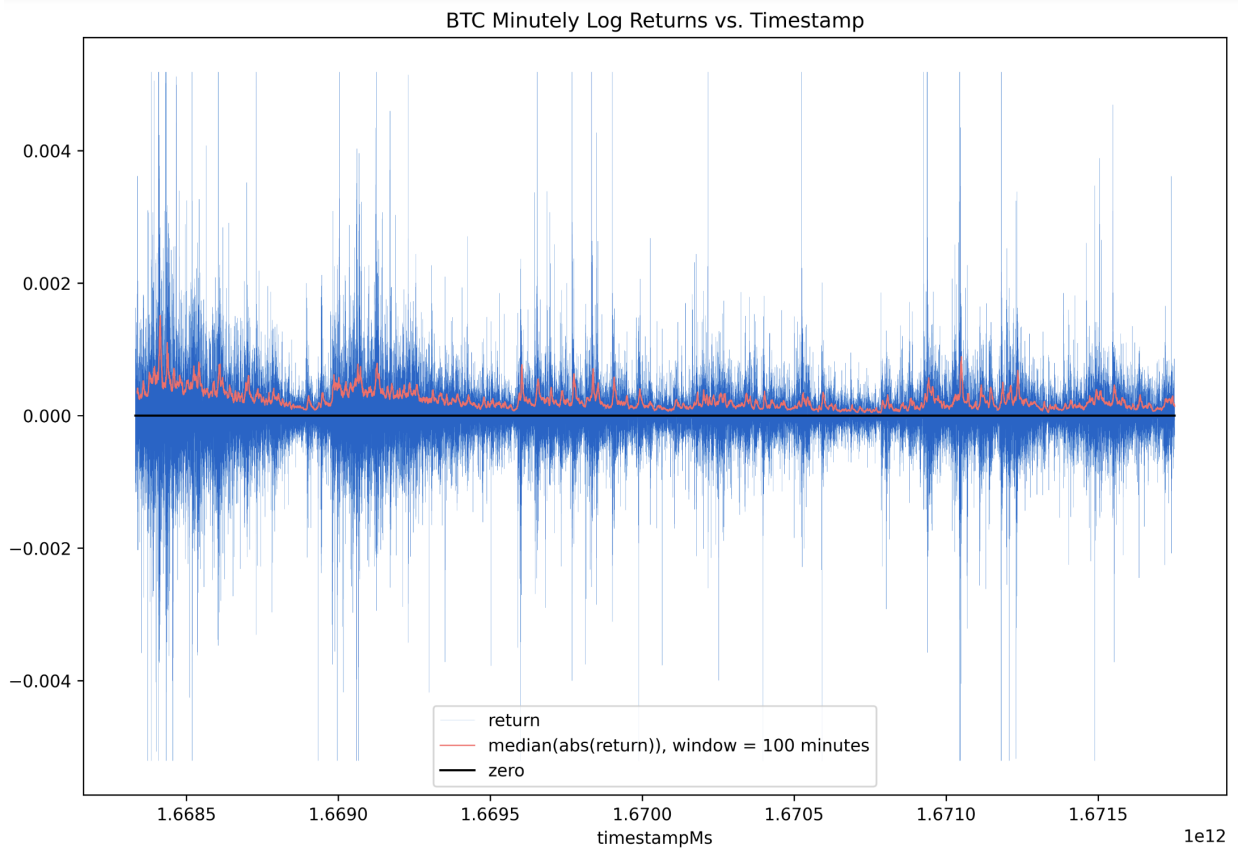


Figure 3: BTC log returns vs. time (blue) and BTC median of absolute log returns (red), from 11/14/2022 to 12/22/2022. The top and bottom 0.05% of values are removed, for ease of viewing.

assume that the mean return is approximately zero, then the returns are simply the residuals of a timeseries process with mean 0, and the variance of these residuals demonstrate autoregressive tendencies.

Modelling Volatility with GARCH. We can extend our GBM model to track a more realistic volatility by modeling the autoregressive tendency of the variance of returns. By looking at historical data, we are able to see that a GARCH(1,1) model of the volatilities is the most fitting. From here, we can fit a GARCH model to historical data to find the baseline volatility (ω), the ARCH parameter (α), and the GARCH parameter (β). We then compute the following (assuming that the drift, μ, is equal to zero):

$$S_t = S_{t-1}\left(1 + \varepsilon_t\right), \tag{1}$$

$$\varepsilon_t = \sigma_t \cdot z, \text{ and} \tag{2}$$

$$\sigma_t = \sqrt{\omega + \alpha\varepsilon_{t-1}^2 + \beta\sigma_{t-1}^2}. \tag{3}$$

The z_t term here is a white noise term with mean of 0, and it is commonly set to z_t $\mathcal{N}(0,1)$. This new process is quite similar to the GBM model, except it is discrete, it assumes zero drift, and it has a time-varying volatility parameter σ. With this process, we are able to generate price paths for assets in the same way that we could for the GBM model.

Correlated GARCH. Although our GARCH model improves upon the GBM model, we do not yet capture the fact that returns are correlated. For GBM models, this is typically performed by requiring the white noise term for any two assets, i and j, their Wiener process terms, dW_t^i and dW_t^j, must satisfy $\mathbb{E}[dW_t^i \cdot dW_t^j] = \rho_{i,j}$, where $\rho_{i,j}$ is the Pearson correlation coefficient of the log returns. Instead of using the independent returns white noise distribution $z_t^i \sim \mathcal{N}(0,1)$, we sample all of the white noises from a multivariate normal distribution with mean $\mathbf{0}$ and covariance matrix $\Sigma = [\rho_{i,j}]$:

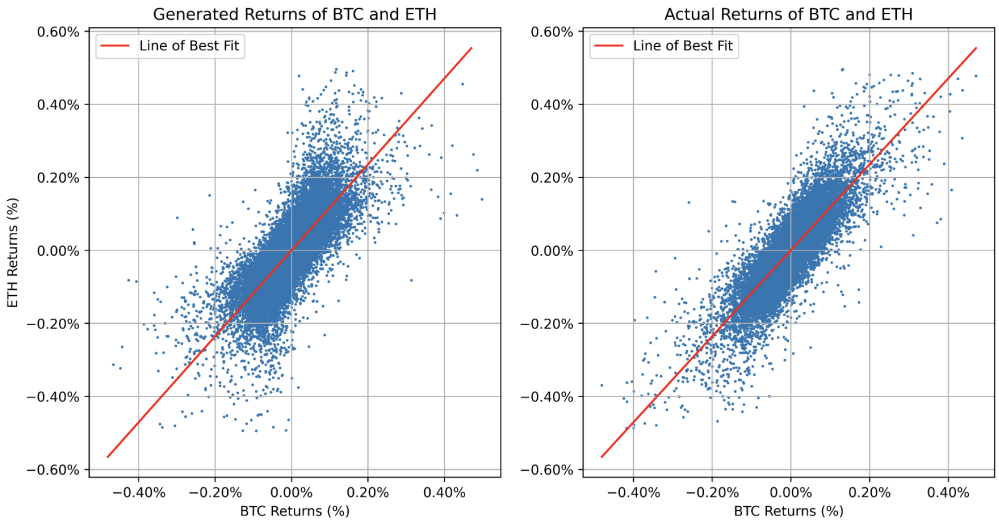


Figure 4: Left: returns generated by our multivariate white noise distribution for BTC and ETH. Right: historical returns returns for BTC and ETH. Each use the same line of best fit, which was generated via historical data, is plotted on each.

$$\mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, [\rho_{i,j}]). \tag{4}$$

In the case where assets' returns have no linear correlation, our formulation is identical to the white noise distribution for independent returns. In practice, many cryptoassets are correlated. See figure 4 for a comparison of the multivariate normal distribution that we sample for ETH and BTC returns, as compared to the returns that have arisen historically. See also figure 5 for an example of a single day's change in asset prices.

2.2.3 Estimating on-chain liquidity

Calculating the profitability of a liquidation over a randomly generated price trajectory requires estimating the slippage on selling the liquidated collateral. One could attempt to use an analytical model, based on historical swaps. However utilizing assumption (4), and the capability of the Chaos platform to interact with on-chain liquidity pools, we wish to rely on agents' operation as well as the AMMs design to achieve a better estimation of the on-chain liquidity over the course of the simulation, which would allow calculating slippage more accurately. Liquidator agents will close the trade in the liquidity pools over the liquidation path as they would in a real-world scenario, arbitrageurs will rebalance the pools back to market price, and liquidity providers will pull liquidity at times of high volatility.

Ideally, we could initialize each pool with the current on-chain liquidity, and simply simulate the agents. However, since optimal arbitrage is computationally intensive, we use DEX aggregators, as well as historical liquidations data to pre-map the possible liquidation routes for each pair of Aave listed assets. That allows us to avoid the routing problem, and limit the number of pools we have to manage. We then initialize each pool with its time-weighted average of last 14 daily on-chain liquidity snapshots. Once the route to liquidate a pair of assets is known and pools are initialized, liquidator agents will execute swaps across the pools using the relevant AMM swap functions.

2.2.4 Borrower agents

We initialize borrowers' portfolios based on historical on-chain data. When examining new risk parameters, such as increasing an asset's liquidation threshold by 5%, we adjust borrower agents to their original health, by increasing their borrows against that asset by 5%. For a reduction of the asset's liquidation threshold, we would simply withdraw 5% of the collateral asset. Borrower agents in our model are passive,

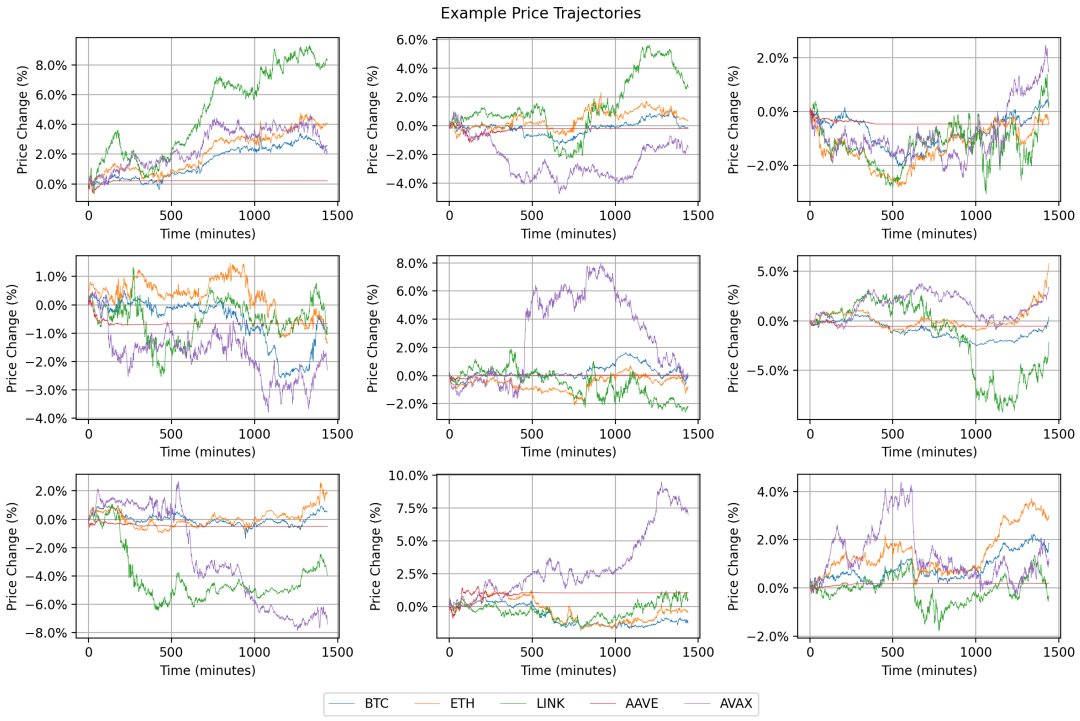


Figure 5: Examples of correlated GARCH-generated price paths for a single day.

once adjusted to the examined parameter configuration, they will not be adjusted throughout the course of the simulation. Borrowers' portfolios can only change due to a liquidation event.

We are examining only significant borrowers who are at risk - their health factor is less than 2, and eliminate those with portfolio value under $1000.

2.2.5 Liquidator agents

Liquidator agents are simulated as rational agents that interact with exclusively on-chain liquidity sources. At the beginning of each block, each liquidator agent examines each below-1-health account, finds the most profitable liquidation on that account, and executes that liquidation by selling the collateral on-chain and paying back the loan with the proceeds of the on-chain sale. The liquidator will only conduct the liquidation if it is profitable to execute via on-chain liquidity sources [4].

For the liquidator agents, we compute the optimal liquidations via a single-peak search over the space of liquidation sizes. The proportion of liquidated collateral is in $(0, 0.5]$, and we can search this space on each borrowed asset to find the optimal proportion of collateral that is liquidated. For m collateral assets and n borrowed assets, our algorithm takes $O(m \cdot n)$ time to find the collateral and borrowed assets, along with the optimal proportion of liquidated collateral asset.

We assume that only one liquidation happens on an account in each block. This simplifies and speeds up the simulation, since it allows us to ignore intra-block competitive liquidator behaviors. Furthermore, this assumption leads to a systematic overestimate in our simulation's estimates of protocol losses, which leads to more conservative parameter estimates.

[4]The assumption that only on-chain liquidity sources are used leads to a systematic underestimate of the amount of liquidations that would occur in real life, and thus leads our simulations to overestimate the true value at risk. However, we have verified with on-chain data that this assumption is quite accurate for historical liquidations, and any error introduced by this assumption will only lead to more conservative risk parameter recommendations.

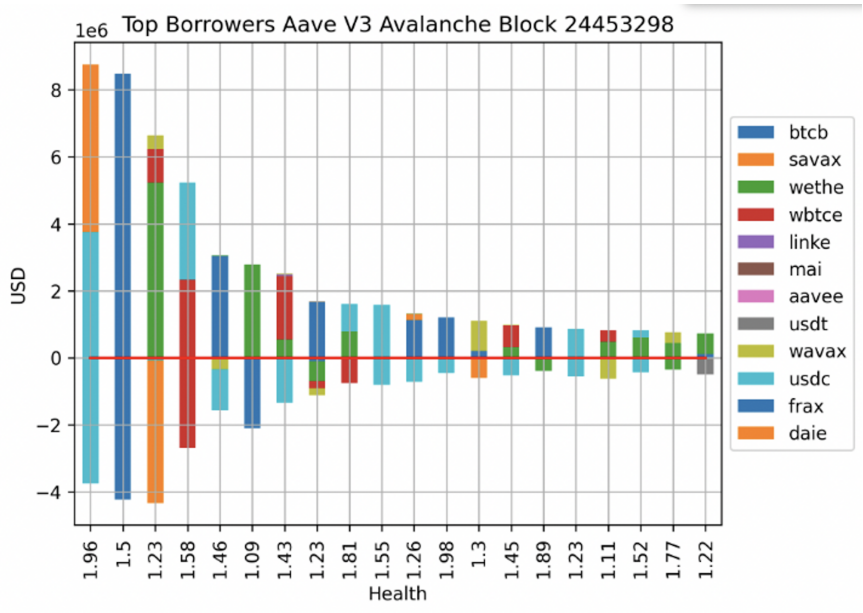


Figure 6: Top borrowers on the Aave protocol. Each bar represents a borrower. For each borrower, the quantity *above* the red line is the supplied amount for each asset; the quantity *below* the red line is the borrowed amount for each asset.

2.2.6 Protocol losses

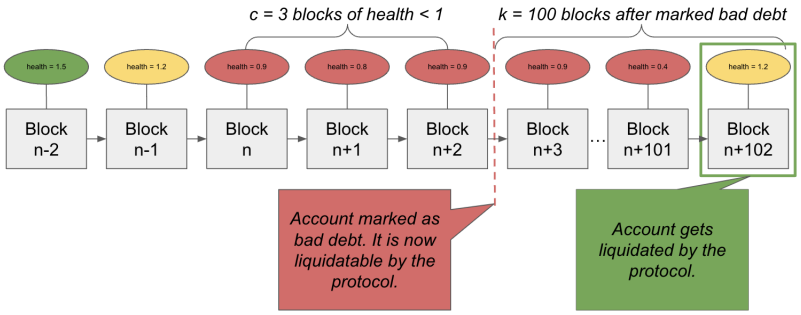
Calculating an appropriate metric to represent protocol losses is as much of an art as it is a science. There does not exist a failsafe mechanism that caps the downside that the protocol may face on an underwater position that is not liquidatable. It requires a governance vote to clear out bad debt from the protocol, and there is no guarantee that these governance votes reach an actionable conclusion in a finite period of time. Thus, there is no protocol-enforced upper bound on the losses that may be incurred by a bad position, other than marking that position's collateral to zero.

With that said, we model this uncertainty of flushing out bad debt by using a time delay parameter k. When a position goes underwater and is not liquidated for 2 consecutive blocks, we say that the protocol recognizes the position as bad debt. We say that the protocol has a delay of k blocks following the time they recognize the position as bad debt, at which point they liquidate the position at market using on-chain liquidity. Specifically, the protocol sells all of the position's collateral, pays it all toward the loan, and then pays the additional loan that is not covered by the sold collateral. This amount – the paid back loan value minus the sold collateral value – is the protocol's loss on the position.

To find the protocol's total loss in a period, we sum up the loss that it incurs from each of its positions in the period. The great majority of positions will not dip below 1 health, and thus will not lead to a loss for the protocol.

3 Results

We have run a simulation to measure the Value at Risk of Aave v3 deployment on Avalanche using the setup described in this paper. We have generated 3 Million price trajectories using the GARCH model, Lend-Borrow position and liquidity snapshots that were taken from January 3, 2023.



Demo of how we assume bad debt liquidations clear. The parameters c and k are manually determined. c = amount of time before something is marked as bad debt. k = amount of time before protocol liquidates bad debt.

When the account collateral is liquidated by the protocol, we calculate *(collateral sale value) - (amount of loan paid back)*, and this is the protocol's profit from the liquidation. This profit will almost certainly be a negative number.

Figure 7: The methodology we use to compute protocol losses.

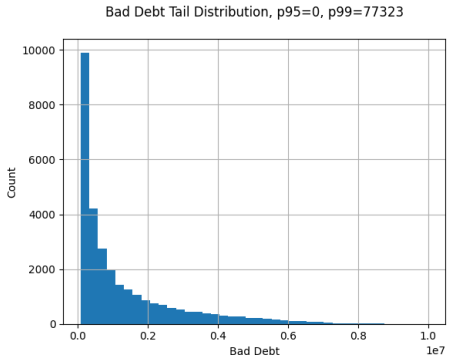


Figure 8: The tail of bad debt across 3 million samples. For this experiment, we saw that the p99 was \$77323. Figure shows the distribution of the top 1% of bad debt.

A Risk Parameter Search

In section 2, we discuss our methodology for determining if a risk parameter $s \in S$ leads to a lower VaR than our upper bound K. Although this is important, it is trivial to find risk parameters that solve this, e.g. by setting the liquidation threshold to 0. Clearly we would not want to set the liquidation threshold so low, because that would render the protocol extremely capital inefficient. We must balance the protocol's objectives of a high liquidation threshold and low liquidation bonus with the VaR constraint that pushes for a low liquidation threshold and high liquidation bonus. Naturally, we can represent this as a mathematical program:

$$\begin{aligned} &\max U(s) \\ &\text{s.t. } \mathrm{VaR}(s) \leq K \\ &\text{s.t. } s \in S, \end{aligned}$$

where $U(s)$ is the amount that the protocol values parameter configuration s, and $\mathrm{VaR}(s)$ is the value at risk under parameter configuration s. We use definition 2.1 for value at risk, and we define here the utility function U.

Definition A.1 (Protocol Utility Function)**.** *The protocol utility function, $U : S \to \mathbb{R}$, is given by*

$$U(s) = \frac{\mathbb{E}[\textit{protocol profit} \mid s]}{\textit{VaR}(s)},$$

where $\mathrm{VaR}(s) = \mathrm{p99}[\text{losses} \mid s]$ = the 99th percentile largest amount of losses for the protocol. This value is strictly positive.

This utility function is inspired by the Sharpe ratio, which is a metric used in finance to report a portfolio's risk-adjusted returns. Here, our utility function aims to grow expected returns while keeping p99 losses as small as possible. Unlike the Sharpe ratio, which uses the standard deviation of returns, we use the p99 of protocol losses. Our reasoning is that the distribution of protocol losses is not known to us except through our simulations, and the p99 of protocol losses is a more risk-sensitive value for fat-tailed distributions than standard deviation.

Given the utility function U, we can now utilize numerical optimization techniques to optimize our risk parameter configuration s. In particular, since the parameter space S is only two-dimensional, we utilize a simple grid search on parameters in S, compute VaR, and move on to other parameters with higher utility if the current parameters' VaR is less than K. We repeat this process to create a small list of parameters, and we then pass these parameters to the on-chain simulation test.

Our current parameter search optimization is far from perfect, but it is sufficiently fast for us to pass risk parameters into the on-chain simulation engine to check for statistical significance. There are a number of ways that we may improve the parameter search process in the future: further speeding up the surrogate function calculation, introducing Bayesian optimization techniques to reduce the number of VaR calculations, representing VaR as a component of the utility function rather than as a constraint, and utilizing previous simulations to warm-start our parameter search.

B Binomial Statistical Test of Protocol Losses

Here we provide a statistical testing framework that can be used to achieve a confidence level on the probability of losses exceeding a bound. In particular, we aim to make a statistical test of the statement of the form

$$\Pr(\text{protocol incurs losses greater than } K) \leq p^*,$$

for some pre-specified bound loss K and bound probability p^*. Furthermore, let α be the significance level for our statistical test, and let $p = \Pr(\text{protocol incurs losses greater than } K)$. We define our null hypothesis as follows, $H_0 : p \geq p^*$. The aim of our statistical test is to utilize simulations of protocol losses to reject the null hypothesis with α significance level. Since each time we generate protocol losses is independent and identically distributed (IID), we can perform our statistical test as a binomial experiment.

Binomial Experiment Testing Algorithm.

1. We run our simulation n times to generate a vector of protocol losses, $\mathbf{l} \in (\mathbb{R} \cup 0)$. We count the number of instances where the loss is greater than K; call this value m.

2. Suppose, to the contrary, that the null hypothesis is true, and that $p \geq p^*$. Then the probability that we observe m or fewer instances where the protocol losses are less than or equal to K is given by the binomial distribution CDF, F, with probability p:

$$F(m; n, p) = \sum_{k=0}^{m} \binom{n}{k} \cdot (p)^k \cdot (1-p)^{n-k}.$$

3. For two binomial probabilities, p_1 and p_2, we know that the binomial CDF $F(m; n, p_1)$ is pointwise less than the binomial CDF $F(m; n, p_2)$ if $p_1 > p_2$. We do not provide a proof for this statement here. Thus, we know that since $p \geq p^*$, then either (a) $p > p^*$ and $F(m; n, p) < F(m; n, p^*)$, or (b) $p = p^*$ and $F(m; n, p) = F(m; n, p^*)$. Thus, we know that $F(m; n, p) \leq F(m; n, p^*)$ pointwise.

4. Therefore, we know that the probability that we observe m of fewer instances where the protocol losses are less than or equal to K must be less than or equal to $F(m; n, p^*)$. We can calculate this value as

$$F(m; n, p^*) = \sum_{k=0}^{m} \binom{n}{k} \cdot (p^*)^k \cdot (1-p^*)^{n-k}.$$

5. Let $q = F(m; n, p^*)$. This is an upper bound on the probability that we observe m or fewer instances where the protocol losses are less than or equal to K. Thus, if $q < \alpha$, then we can reject the null hypothesis with α significance level. Otherwise, we fail to reject the null hypothesis.

Example Suppose we want to show with a 0.05 significance level (95% confidence level) that the Pr(losses greater than \$300,000) is less than 0.1%. Then we run $100,000$ simulations, and we observe that losses are greater than $\$300,000$ in $m = 80$ of the simulations. We calculate

$$q = \sum_{k=0}^{80} \binom{100,000}{k} \cdot (.001)^k \cdot (0.999)^{100,000-k} = 0.0226,$$

which is less than our significance level of 0.05. Thus, we reject the null hypothesis with 95% confidence.