FEDERAL UNIVERSITY OF SANTA CATARINA
TECHNOLOGY CENTER
AUTOMATION AND SYSTEMS DEPARTMENT
UNDERGRADUATE COURSE IN CONTROL AND AUTOMATION ENGINEERING

João Victor Assis da Silveira

Historical Loss-versus-Rebalancing (LVR) Data for Blockchain Automated Market Makers (AMMs)

Florianópolis
2024

João Victor Assis da Silveira

Historical Loss-versus-Rebalancing (LVR) Data for Blockchain Automated Market Makers (AMMs)

Final report of the subject DAS5511 (Course Final Project) as a Concluding Dissertation of the Undergraduate Course in Control and Automation Engineering of the Federal University of Santa Catarina.
Supervisor: Prof. Alex Sandro Roschildt Pinto, Dr.

Florianópolis
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Assis da Silveira, João Victor
Historical Loss-versus-Rebalancing (LVR) Data for Blockchain Automated Market Makers (AMMs) / João Victor Assis da Silveira ; orientador, Alex Sandro Roschildt Pinto, 2024.
66 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2024.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Loss-versus Rebalancing. 3. Automated Market Makers. 4. Blockchain. 5. Software. I. Roschildt Pinto, Alex Sandro. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

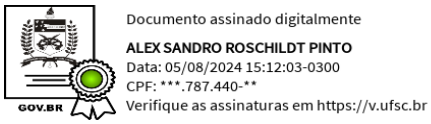
João Victor Assis da Silveira

Historical Loss-versus-Rebalancing (LVR) Data for Blockchain Automated Market Makers (AMMs)

This dissertation was evaluated in the context of the subject DAS5511 (Course Final Project) and approved in its final form by the Undergraduate Course in Control and Automation Engineering

Florianópolis, June 10th, 2024.

Prof. Marcelo De Lellis Costa de Oliveira, Dr.
Course Coordinator



Prof. Alex Sandro Roschildt Pinto, Dr.
Advisor
UFSC/CTC/INE

José Fernando Ribeiro Rosa, B.A
Supervisor
Bleu

Prof. Rodrigo Castelan Carlson, Dr.
Evaluator
UFSC/CTC/DAS

Prof. Hector Bessa Silveira, Dr.
Board President
UFSC/CTC/DAS

This work is dedicated to my classmates and my dear mother.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my friends for their support and encouragement over the past few years. Your partnership and companionship were invaluable.

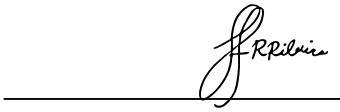
I would also like to thank the company Bleu Studio and for the Prof. Alex Sandro Roschildt Pinto for their support in implementing this project. His guidance was fundamental to the realization of this project.

And lastly, and most importantly, I wish to express my gratitude to Maria Luiza, my mother, whose strength and love have been the foundation for all the achievements I have reached up to this moment. For every sacrifice, every word of encouragement, and every act of kindness, I will always be grateful. In every obstacle overcome and every small step towards this accomplishment, your presence was essential. For being with me in moments of uncertainty and celebrating victories with me, my admiration for you only grows. Without your dedication, this journey would not have been possible. This project is more than an academic goal; it is the materialization of your faith in me and your love. With love and gratitude, João Victor, your son.

DISCLAIMER

Florianópolis, June 10th, 2024.

As representative of the Bleu Studio in which the present work was carried out, I declare this document to be exempt from any confidential or sensitive content regarding intellectual property, that may keep it from being published by the Federal University of Santa Catarina (UFSC) to the general public, including its online availability in the Institutional Repository of the University Library (BU). Furthermore, I attest knowledge of the obligation by the author, as a student of UFSC, to deposit this document in the said Institutional Repository, for being it a Final Program Dissertation (*"Trabalho de Conclusão de Curso"*), in accordance with the *Resolução Normativa n°126/2019/CUn*.



José Fernando Ribeiro Rosa
Bleu

ABSTRACT

Decentralized Finance (DeFi) has emerged as a rapidly growing sector within the blockchain ecosystem, enabling financial services without the need for centralized intermediaries. One of the key components of DeFi is Automated Market Makers (AMMs), which facilitate decentralized trading of cryptocurrencies and other digital assets.

Most AMMs, such as Uniswap, Balancer, and Cowswap, rely on so-called "liquidity pools" composed of two or more assets. These pools enable traders to swap tokens without the need for an order book or counterparty. However, as trading activity occurs, the pools can become unbalanced, leading to potential losses for liquidity providers.

The concept of Loss-versus-Rebalancing (LVR) seeks to analyze and quantify the trade-off between the potential losses due to arbitrage and the costs associated with frequent rebalancing operations. By understanding this relationship, liquidity providers and AMM developers can make informed decisions regarding pool management strategies, asset allocation, and potential optimizations.

The main aim of this thesis project is to apply methods for calculating historical LVR within a specific AMM protocol. The development of a backend and frontend application serves primarily as a tool to display these findings. This approach not only clarifies the core focus of the research—analyzing LVR—but also enhances the application's utility for liquidity providers and researchers interested in exploring the implications of LVR.

Keywords: LVR. AMM. Pool. Liquidy Pool, Liquidy Provider.

RESUMO

As Finanças Descentralizadas (DeFi) surgiram como um setor de rápido crescimento dentro do ecossistema blockchain, permitindo serviços financeiros sem a necessidade de intermediários centralizados. Um dos principais componentes do DeFi são os Formadores de Mercado Automatizados (AMMs), que facilitam a negociação descentralizada de criptomoedas e outros ativos digitais.

A maioria dos AMMs, como Uniswap, Balancer e Cowswap, depende dos chamados "*pools* de liquidez"compostos por dois ou mais ativos. Esses *pools* permitem que os negociadores troquem tokens sem a necessidade de um livro de ordens ou contraparte. No entanto, à medida que a atividade de negociação ocorre, os *pools* podem se desequilibrar, levando a possíveis perdas para os provedores de liquidez.

O conceito de Perda versus Rebalanceamento (LVR) busca analisar e quantificar a compensação entre as potenciais perdas devido à arbitragem e os custos associados às operações frequentes de rebalanceamento. Ao entender essa relação, os provedores de liquidez e os desenvolvedores de AMM podem tomar decisões informadas sobre estratégias de gerenciamento de *pools*, alocação de ativos e possíveis otimizações.

O principal objetivo deste projeto de tese é aplicar métodos de cálculo de LVR histórico dentro de um protocolo AMM específico. O desenvolvimento de uma aplicação backend e frontend serve principalmente como uma ferramenta para exibir essas descobertas. Esta abordagem não apenas esclarece o foco principal da pesquisa – a análise do LVR – mas também aumenta a utilidade da aplicação para provedores de liquidez e pesquisadores interessados em explorar as implicações do LVR.

Palavras-chave: LVR. AMM. Pool. Piscina de Liquidez, Provedor de Liquidez.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

CEX	Centralized Exchange
DeFi	Decentralized Finance
DEX	Decentralized Exchange
LPs	Liquidy providers
LVR	Loss-Versus-Rebalancing
UFSC	Federal University of Santa Catarina

CONTENTS

1 INTRODUCTION

The rapid rise of Decentralized Finance (DeFi) within the blockchain ecosystem has introduced a series of financial innovations that eliminate the need for centralized intermediaries. Among these innovations, Automated Market Makers (AMMs) stand out, allowing for decentralized trading of cryptocurrencies and other digital assets through liquidity pools composed of two or more assets. However, as trading activity occurs, these pools can become unbalanced, resulting in potential losses for liquidity providers.

The central issue addressed by this work is the concept of Loss-Versus-Rebalancing (LVR), which analyzes and quantifies the trade-off between potential losses due to arbitrage and the costs associated with frequent rebalancing operations within liquidity pools. The absence of comprehensive historical LVR data impedes liquidity providers and AMM developers from making informed decisions regarding pool management strategies and asset allocation.

The lack of detailed and historical understanding of LVR represents a significant challenge to the development and efficiency of AMMs. For companies and institutions involved in the DeFi space, such as decentralized exchanges and trading platforms, optimizing the management of liquidity pools is crucial. A deep understanding of LVR can lead to more effective loss mitigation strategies and the development of more robust and resilient AMM protocols, benefiting both liquidity providers and end-users.

To address this problem, this work proposes the development of an application that calculates and displays historical LVR values for Uniswap v2 liquidity pools. The solution involves creating a backend application for data aggregation and LVR calculations, and a frontend application for visualizing this data. The application will enable liquidity providers, developers, and researchers to explore and analyze LVR data, providing valuable insights into the dynamics of liquidity pools.

1.1 METHODOLOGY

1. **Defining LVR for a specific AMM**: Establishing a clear and precise definition of LVR tailored to the mechanics of Uniswap v2.
2. **Test methodologies for calculating historical LVR**: Apply methodologies and processes to calculate LVR values using historical trading and liquidity data.
3. **Implementing the backend application**: Building a backend system to automate data collection, processing, and LVR calculations.
4. **Creating the frontend application**: Developing an intuitive frontend interface to display the calculated LVR values, allowing users to explore and analyze the data.

The results obtained include the creation of a functional system that calculates and displays historical LVR values for Uniswap v2 liquidity pools. This system provides liquidity providers with a powerful tool to optimize their pool management strategies, minimizing losses and improving asset performance. For AMM developers, the insights derived from historical LVR data can inform the development of more efficient and robust protocols.

1.2 THE COMPANY

This section presents the company where the internship was conducted, Bleu Studio.

1.2.1 Bleu Studio

Bleu Studio specializes in web3 technology and user experience, serving both enterprises and DAOs. Our vision is rooted in the belief that there is significant potential to enhance and simplify blockchain interactions. Our mission is to bridge the gap between the advanced capabilities of blockchain and the current end-user experience in web3, which we believe can be greatly improved. Additionally, Bleu Studio is involved in projects across web2.0 and web2.5.

The foundation of Bleu Studio stems from the entrepreneurial spirit of alumni from the Control and Automation Engineering course at the Federal University of Santa Catarina (UFSC). Established on December 22, Bleu's primary goal is to contribute significantly to software projects in the crypto space. The founding members, who were already familiar with each other, shared a common vision: to combine their expertise and passion to drive advancements in the ecosystem by delivering exceptional solutions. Despite being a young company with a team of only six individuals, Bleu has already proven its capabilities across various sectors, including web3, web2, and web2.5.

A major part of our work has been in collaboration with the DAO Balancer DeFi Liquidity Protocol. The project highlighted in this report is also associated with Balancer DeFi, adding another achievement to our portfolio. To showcase our experience with Balancer, here are some projects that Bleu Studio has completed:

- **Vault Internal Balances Manager**: An application designed for the efficient management of tokens within the Vault's internal balance.
- **Stable Swap Simulator**: A dashboard that simulates the operations of Balancer Stable Pools, aimed at optimizing pool parameters.
- **Twitter/Discord bots**: Bots that provide real-time metrics, including veBAL details for Balancer's pools, active on Twitter and in the metrics channel of Balancer's Discord.

- **Balpy Vault feature completeness**: Enhancements to the Balancer Python package (balpy), including previously inaccessible Vault functions.

In addition to our work in the cryptographic landscape, Bleu has also achieved significant progress in web2 projects. One notable project in this domain is the development and maintenance of a comprehensive white-label engagement application. This platform allows users to actively participate in our partners' campaigns, driven by a gamification system.

This work was entirely conducted by the author, from exploring the problem to implementing the proposed solutions. This project is an original initiative and does not leverage previous work or contributions from other teams.

1.3 STRUCTURE OF THE DOCUMENT

The document is structured as follows:

1. **Introduction**: Provides an overview of the problem, its importance, the proposed solution, methodology, results obtained, and the author's contributions.
2. **Theoretical Background**: Discusses fundamental theories, models, and technologies necessary for understanding the proposed solution.
3. **Requirements and Conceptual Framework**: Outlines the general, functional, and non-functional requirements of the project, using diagrams and information flows to establish the conceptual framework.
4. **Development and Implementation**: Details the development and implementation of the project, including diagrams, use cases, graphical interfaces, and decisions made.
5. **Evaluation**: Analyzes the obtained results, comparing accuracy among methods and performance improvements with chosen architecture.
6. **Conclusion**: Summarizes the work done, motivation, proposed solution, main results, project limitations, and suggestions for future work.

2 THEORETICAL BACKGROUND

The current chapter aims to introduce the reader to the concepts behind the technologies used for the development of this thesis project. The chapter will explore each technology's unique capabilities, reasons for their selection, and how they were seamlessly integrated to build the final web application.

We will explore the fundamental principles underlying the thesis.

2.1 CONCEPTS AND TOOLS

2.1.1 Blockchain

Blockchain technology functions similarly to a digital ledger or record book but offers significantly more security and sophistication. Picture it as a sequence of interconnected notebooks, each documenting a series of transactions or exchanges, such as transferring digital currency. When one of these notebooks, referred to as "blocks," is filled with transactions, a new one is initiated.

These blocks are linked in a particular order, creating a chain, which is why it's called a "blockchain." The key feature is that each new block includes a distinct reference to the block before it, like a unique fingerprint, ensuring the blocks are correctly ordered and untampered.

A distinctive feature of blockchain is its decentralization. Instead of a single person or organization controlling this ledger, it is maintained by a network of computers distributed globally. This structure ensures that no single entity has authority over the entire chain.

This decentralization enhances security and transparency. Since every participant in the network holds a copy of the ledger, it remains highly open and public. Furthermore, once information is recorded in a block, altering it becomes exceedingly challenging. To modify any data, one would have to change all subsequent blocks in the chain, a feat that is virtually impossible.

While blockchain is best known for underpinning cryptocurrencies like Bitcoin, its applications extend far beyond digital currencies. It can be utilized for tracking product supply chains, securely storing medical records, facilitating transparent voting in elections, and much more.

The figure 1 illustrates the process of a blockchain transaction from start to finish. Initially, a new transaction is entered and transmitted to a network of peer-to-peer computers globally. This network solves equations to confirm the validity of the transaction. Once confirmed as legitimate, transactions are clustered into blocks. These blocks are then chained together, forming a permanent record of all transactions. The process concludes with the transaction being complete.

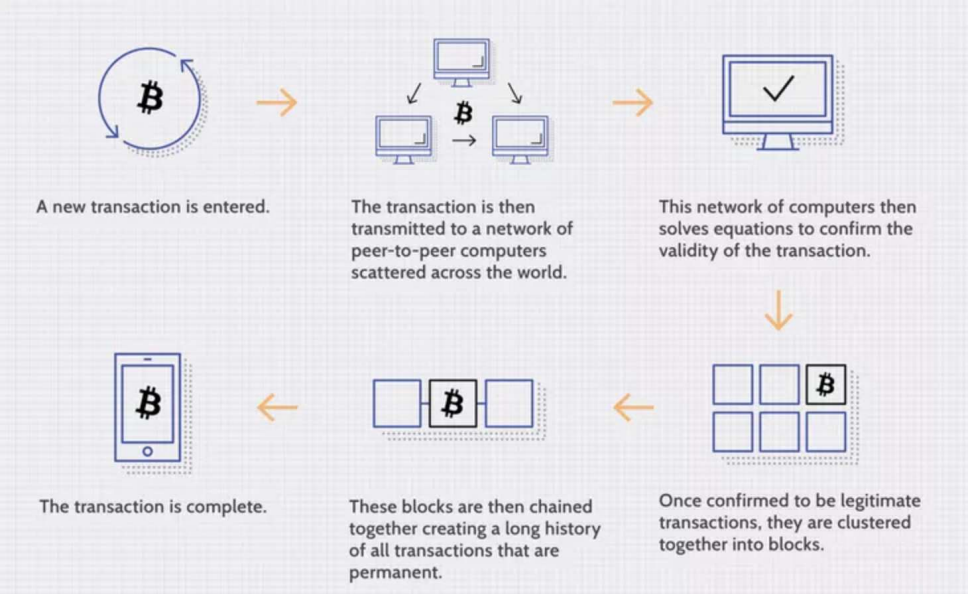


Figure 1 – How blockchain works.

Source: (HAYES, 2024)

In essence, blockchain is a revolutionary way of keeping records that are secure, transparent, and not under the control of any single entity, making it useful for a variety of applications beyond just financial transactions. (BLOCKCHAIN. . . , 2023)

2.1.2 Smart Contracts

Smart contracts are self-executing programs on a blockchain that perform specific actions when predefined conditions are met, eliminating intermediaries and minimizing delays. They automate agreements and workflows, providing immediate and assured outcomes for all parties involved.

These contracts function using "if/when...then..." logic embedded in blockchain code. A network of computers carries out the necessary actions, such as releasing funds or registering a vehicle, once the conditions are confirmed. Transactions are recorded on the blockchain permanently, ensuring immutability and controlled access for authorized parties.

Smart contracts can incorporate multiple conditions, requiring participants to agree on transaction representations, governing rules, potential exceptions, and dispute resolution methods. Initially programmed by developers, the growing adoption of blockchain technology in business has led to the creation of templates and online tools that simplify smart contract development (SMART. . . , 2023).

2.1.3 Subgraph

A Subgraph is a tool specifically designed for indexing and querying data from blockchains, primarily used to efficiently collect and process events generated by smart

contracts. When a smart contract on a blockchain like Ethereum emits an event, the Subgraph functions as an indexer, capturing and organizing this data to make it easily accessible and searchable.

The main purpose of a Subgraph is to transform the complex and often unstructured data from blockchain events into a format that is simple to query and understand. It defines which data from the blockchain will be indexed and how it will be stored, making the retrieval of specific information from the blockchain quick and straightforward, as opposed to manually sifting through the entire blockchain data.

In essence, a Subgraph is an essential tool for anyone dealing with blockchain data, as it provides a means to efficiently index, store, and query event data from smart contracts. This makes blockchain data more accessible and usable for a variety of applications. (SUBGRAPH. . . , 2023)

In the context of this project, Subgraph will be employed to capture and organize the data from AMM pools, specifically focusing on the events related to liquidity provision, swaps, and other relevant activities. This indexed data will then calculate LVR values, enabling more efficient and insightful analysis. By utilizing Subgraph, the project ensures that the required data is processed efficiently and made readily available for the backend application to perform LVR calculations and for the frontend application to display these results effectively.

2.2 CENTRALIZED EXCHANGES (CEX)

A centralized exchange (CEX) is a platform used to buy, sell, and trade cryptocurrencies and other assets, operated by a company or organization that maintains full control over all transactions. Unlike decentralized platforms, users of centralized exchanges do not have direct control over their wallet keys and must trust the exchange to manage their funds securely.

To trade tokens through a CEX, users must first create an account and go through a verification process, often including Know Your Customer (KYC) protocols, to begin trading. Centralized exchanges offer a wide range of trading pairs and usually provide advanced features such as margin trading, futures, and options. These platforms act as custodians of users' funds, meaning they hold and manage assets on behalf of their users.

CEXs, being unique control points, are attractive targets for hacking and cyberattacks, leading to notable security risks. There have been numerous high-profile breaches resulting in substantial losses of users' funds. These platforms demand personal information from their users, raising privacy concerns, especially as the data collected may be vulnerable to breaches or misuse.

Another significant disadvantage is counterparty risk, since the exchange controls the transactions and holds the assets, users must trust the exchange to act hon-

estly and efficiently. Additionally, since users do not own their private keys, they have limited control over their funds. Access to funds may be restricted by the exchange for various reasons, sometimes arbitrarily.

2.3 AUTOMATED MARKET MAKERS (AMMS)

Imagine that you have a USDC (a crypto token that is worth 1 dollar) token and want to trade it for Ether (another crypto token); AMMs make this possible. AMMs are smart contracts that enable market participants to trade one cryptocurrency for another directly on the blockchain. They are a class of decentralized exchange protocols that rely on mathematical algorithms to price assets and facilitate trades without needing a traditional order book. Unlike centralized exchanges (CEX) where buyers and sellers place orders to trade assets at specific prices, AMMs allow users to trade directly against a liquidity pool. (CHINTAI, 2024)

2.3.1 How AMMs Work

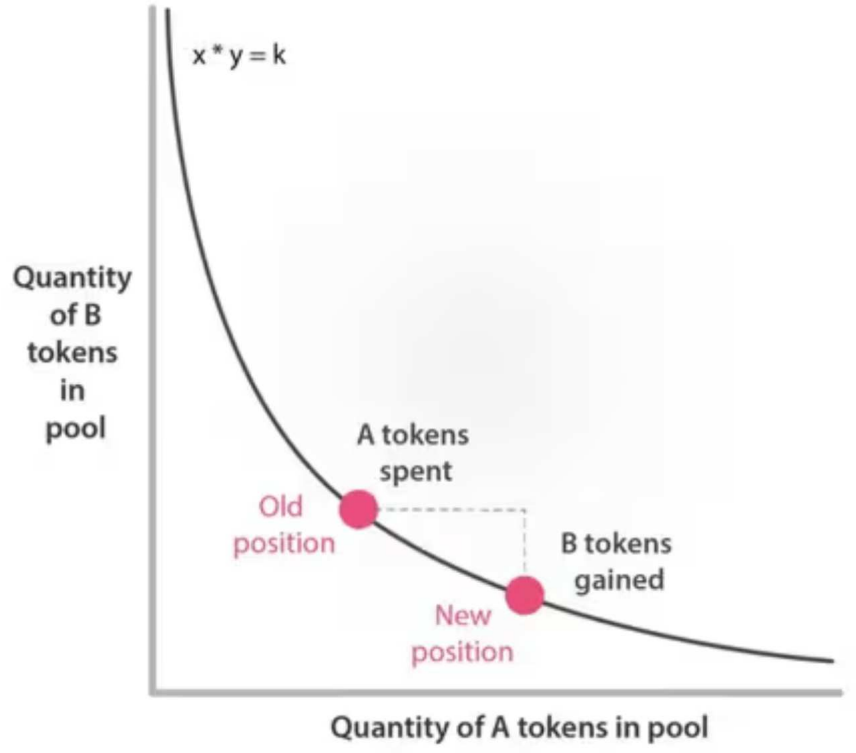
AMMs operate using liquidity pools. Each liquidity pool is governed by a specific algorithm that determines the price of the assets within the pool based on their relative supply. One of the most common algorithms used by AMMs is the constant product formula, represented as:

$$x \times y = k \tag{1}$$

where x and y are the quantities of the two assets in the pool, and k is a constant. This formula ensures that the product of the quantities of the two assets remains constant, regardless of trades that occur within the pool.

The figure 2 illustrates the constant product AMM equation. It shows how the quantities of two assets in the pool are inversely related, maintaining the constant product k. When a user spends token A, the quantity of token B they receive is determined by this relationship, shifting the pool's position along the curve.

Figure 2 – Constant product AMM Equation



Source: (BITDEGREE, 2021).

When a user wants to trade one asset for another, they add their asset to the pool and receive the other asset in return, with the prices automatically adjusted by the AMM's algorithm. This process eliminates the need for matching buyers with sellers, as the trade occurs directly with the liquidity pool.

The figure 3 illustrates the constant product AMM mechanism in Uniswap V2. When a user wants to trade one asset for another, they add their asset to the pool and receive the other asset in return. The prices are automatically adjusted by the AMM's algorithm, which removes the need for matching buyers and sellers since the trade occurs directly with the liquidity pool.

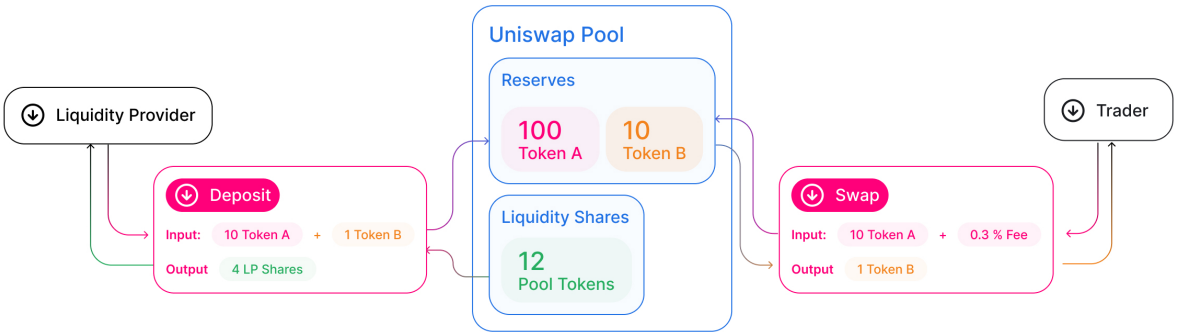


Figure 3 – Constant Product AMM - Uniswap V2.

Source: (UNISWAP, 2020)

2.3.1.1 Maintaining Token Prices in AMM Pools

AMM pools maintain token prices through the mathematical formula that automatically adjusts the asset prices based on their relative supply in the pool.

To align the token prices in an AMM pool with those in CEXs, arbitrageurs play a crucial role. Arbitrageurs monitor price discrepancies between the AMM pool and CEXs. When they detect a price difference, they execute trades that help balance the prices across the platforms. For example, if a token is cheaper in the AMM pool compared to a CEX, arbitrageurs will buy the token in the AMM pool and sell it on the CEX, thereby increasing the token's price in the AMM pool and decreasing it on the CEX until the prices converge.

2.3.2 Liquidity Pools

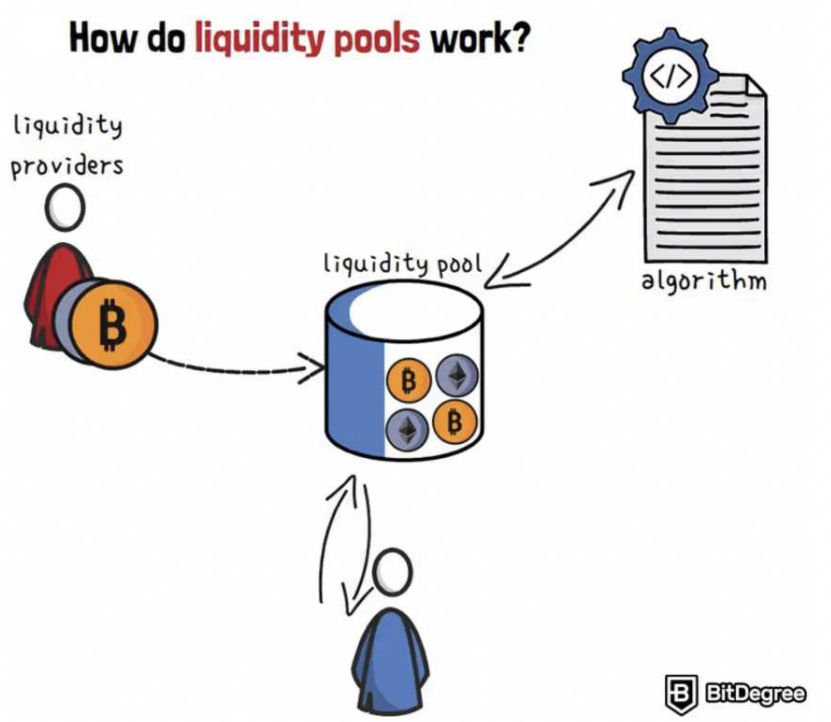
In the context of AMMs, a liquidity pool is a collection of funds locked in a smart contract that is used to facilitate trading on an AMM. Each pool typically contains pairs of tokens (e.g., ETH/USDC). Liquidy providers (LPs) deposit an equivalent value of each token into the pool and, in return, receive LP tokens that represent their share of the pool. These pools are used to execute trades without needing a direct counterparty, as the smart contract automatically manages the trades based on the predefined formula.

Liquidity pools are not only pivotal for trading but also serve as platforms for yield farming and staking. In yield farming, participants deposit assets into a pool to earn transaction fees or other incentives distributed by the protocol. Staking involves locking tokens into a pool to support the operation and security of a blockchain network, with rewards typically given in return.

The figure 6 illustrates how liquidity pools work. Liquidity providers deposit their tokens into the liquidity pool, which is governed by an algorithm. Traders can then swap

tokens within the pool, with the algorithm automatically adjusting prices based on the pool's composition.

Figure 4 – Liquidity Pools



Source: (ROUSE, 2022).

These pools provide transparency, security, and automated processing for DeFi transactions and interactions. Consequently, liquidity pools in decentralized finance are versatile, acting as essential components for trading, lending, staking, and insurance, all supported by smart contract technology. (**POOL**)

2.3.2.1 How They Work Together

- AMM Protocol: Manages the overall trading mechanism, applying the mathematical formula to ensure continuous liquidity and pricing.
- Liquidity Pool: Provides the actual tokens that are traded. When a trade occurs, the tokens in the pool are swapped, and the formula is used to adjust the price according to the relative amounts of tokens in the pool.

2.3.3 Liquidity Providers (LPs)

A Liquidity pool is a place where you can lock up your money or a specific asset, for a set amount of time. If you do so, you'll be called a Liquidity provider. Liquidity

providers are essential to the functioning of AMMs. They contribute assets to the liquidity pools and, in return, receive a share of the transaction fees generated by trades within the pool. LPs bear the risk of impermanent loss, which occurs when the price of the assets in the pool diverges from their price at the time of deposit, potentially leading to a lower value when they withdraw their assets.

2.3.4 Benefits of AMMs

For instance, if a trader wants to exchange ETH from their blockchain wallet for USDC, they could use a custodial CEX like Binance. This process involves depositing ETH to Binance, trading it for USDC within Binance's custody, and then withdrawing the USDC back to their wallet. This method entails several costs and risks, including giving up custody of assets to the exchange, facing exchange credit risk, needing a trading account that meets regulatory requirements, and potential fees and delays imposed by the CEX. Moreover, CEXs might not list all tokens, especially those with smaller market caps, and their trade rules and pricing mechanisms can lack transparency.

AMMs offer an alternative that mitigates these issues. Technically, AMMs are smart contract wallets, which can hold crypto assets and operate entirely based on blockchain code without human intervention. In the earlier example, the AMM wallet would hold both ETH and USDC. When a trader initiates a transaction to swap ETH for USDC, the ETH is transferred to the AMM's inventory, and USDC is simultaneously transferred from the AMM to the trader in a single atomic transaction. This process eliminates credit risk as trades are completed instantly, ensuring the trader receives the purchased assets upon selling the original assets. The pricing mechanisms in AMMs are transparent; for example, the Uniswap v2 smart contract is composed of less than a thousand lines of publicly available code. Once deployed on the blockchain, the code is immutable, providing a reliable and transparent trading environment for those who avoid exchange credit risk, face barriers to CEX access, or value the transparency of AMM operations.

The asset inventory for AMMs is supplied in a decentralized manner. Any market participant can become an LP by contributing ETH and USDC to the AMM's inventory for an ETH-USDC pair or any other token pair. The AMM pool uses these contributions for trading, and trading fees are proportionately distributed to liquidity providers based on their share of the inventory. Liquidity providers can withdraw their share of the assets at any time, although the amounts of ETH and USDC may differ from their initial contributions due to market trades and fee accumulation. The primary focus of our paper is to analyze the costs and benefits for liquidity providers in AMMs.

2.3.5 Earning from Arbitrage

Arbitrage is a trading strategy that involves buying and selling the same asset in different markets to profit from differences in the asset's price in those markets. Arbitrageurs earn profits by exploiting price differences between AMM pools and CEXs. Here's how it works:

- **Identifying Price Discrepancies**: Arbitrageurs continuously monitor the prices of tokens in AMM pools and CEXs. They use sophisticated algorithms and tools to detect when a token is priced lower on one platform compared to the other.
- **Executing Trades**: Once a price discrepancy is identified, arbitrageurs quickly buy the underpriced token from the cheaper platform and sell it on the more expensive platform. This can involve moving tokens from an AMM pool to a CEX or vice versa.
- **Profit Realization**: The profit from arbitrage is realized in the difference between the buying and selling prices, minus any transaction fees. Since these opportunities often exist for only a brief moment, arbitrageurs need to act swiftly to capitalize on them.

The figure 5 illustrates an arbitrage flow example. It shows the step-by-step process of how arbitrageurs monitor prices on both centralized exchanges (CEX) and decentralized exchanges (DEX), prepare funds, buy the asset on the platform where it is cheaper, transfer the asset, and sell it on the platform where it is more expensive, finally calculating the net profit after deducting fees.

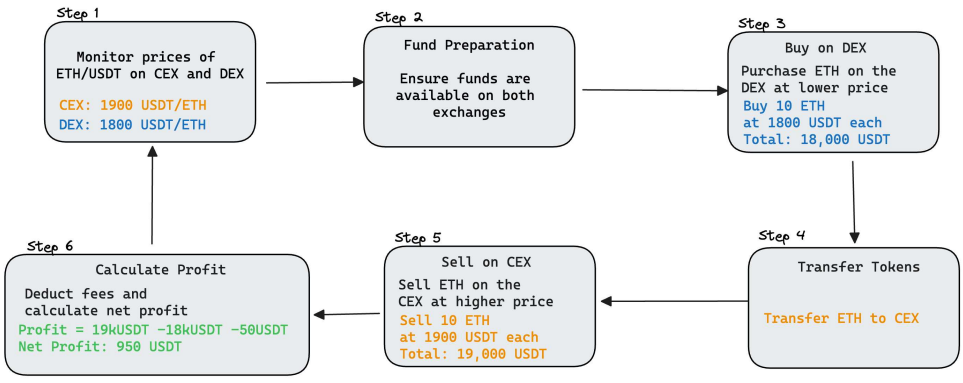


Figure 5 – Arbitrage flow example.

Source: Author

Through this process, arbitrageurs help stabilize prices across different platforms. Their activities ensure that the prices of tokens in AMM pools remain close to those in CEXs, contributing to the overall efficiency and liquidity of the market.

2.3.6 Rebalancing

In the context of constant function AMMs, a rebalancing portfolio refers to the ongoing adjustments made to the asset ratios within the liquidity pool to maintain a constant product formula. In those models, liquidity providers deposit an equal value of two assets into a pool, and the pool's asset ratios adjust automatically with each trade to ensure the product of the asset quantities remains constant. (MILIONIS et al., 2022)

For example, Uniswap v2 operates on the principle $x \cdot y = k$, where x and y are the quantities of two different assets in the pool, and k is a constant. This formula ensures that the product of the quantities of the two assets always equals k. Each trade within the pool alters the quantities of the two assets. For example, if a trader swaps one asset for another, the pool automatically adjusts the quantities to maintain the constant product. This process inherently rebalances the pool after each transaction. When LPs provide liquidity to a pool, they add equal value to both assets. As trades occur, the relative amounts of these assets change, necessitating continuous rebalancing to maintain the constant product formula. The LPs' positions are effectively "rebalanced" with each trade to reflect the current pool composition. (HAYDEN ADAMS NOAH ZINSMEISTER, 2020)

Consider a Uniswap v2 pool with ETH and USDC. Initially, the pool might have 100 ETH and 200,000 USDC, making $k = 100 \times 200,000 = 20,000,000$. If a trader swaps 10 ETH for USDC, the pool rebalances by increasing the amount of USDC and reducing the amount of ETH to maintain the product k. The new asset quantities adjust according to the constant product formula, ensuring the pool remains balanced.

2.3.6.1 Mathematical Representation

The rebalancing can be mathematically represented by the formula:

$$(x + \Delta x) \cdot (y - \Delta y) = k \tag{2}$$

where x and y are the initial quantities of the two assets, Δx and Δy are the changes in quantities due to a trade.

This equation ensures that the product k remains constant, reflecting the continuous rebalancing of the portfolio within the pool.

2.3.6.2 Impermanent Loss

Rebalancing in Uniswap v2 also leads to a phenomenon known as impermanent loss, where the value of LPs' assets in the pool may be less than holding the assets outside the pool due to the rebalancing mechanism. This occurs because the pool continuously adjusts the asset ratios, potentially leading to less favorable rates for LPs compared to static holdings. (IMPERMANENT. . . , 2020)

2.4 LOSS-VERSUS-REBALANCING

LVR in AMMs is a critical concept that quantifies the adverse selection cost faced by liquidity providers due to the rebalancing activities in the AMM pools. This concept is crucial for understanding how LPs' returns are affected by the inherent mechanisms of AMMs, such as Uniswap and Balancer.

In an AMM, LPs provide liquidity by depositing assets into the pool, which are then used to facilitate trades. The pool automatically rebalances itself by adjusting the asset ratios according to a predefined mathematical formula, such as the constant product formula used in Uniswap v2. This rebalancing, while essential for the functioning of the AMM, can lead to what is termed "impermanent loss" — a temporary loss in value compared to simply holding the assets outside the pool.

LVR specifically measures the difference in performance between a liquidity position in an AMM and a hypothetical rebalancing strategy that continuously adjusts asset holdings in response to price changes. This strategy effectively hedges against market movements, mitigating the impact of volatility on the LP's returns. The LVR metric captures the additional costs incurred by LPs due to their inability to dynamically rebalance their positions as effectively as an ideal rebalancing strategy. (MILIONIS et al., 2022)

Mathematically, LVR can be understood as the sum of the slippage cost (the loss due to trading against the AMM's pricing curve) and the opportunity cost of not being able to rebalance positions continuously at market prices. This cost increases with the volatility of the underlying assets and the frequency of trades in the AMM pool.

Understanding LVR is crucial for designing better AMM protocols and setting appropriate fee structures to compensate LPs for the risks they bear. By minimizing LVR, AMM designers can improve the attractiveness of providing liquidity, thus enhancing the overall efficiency and liquidity of the market.

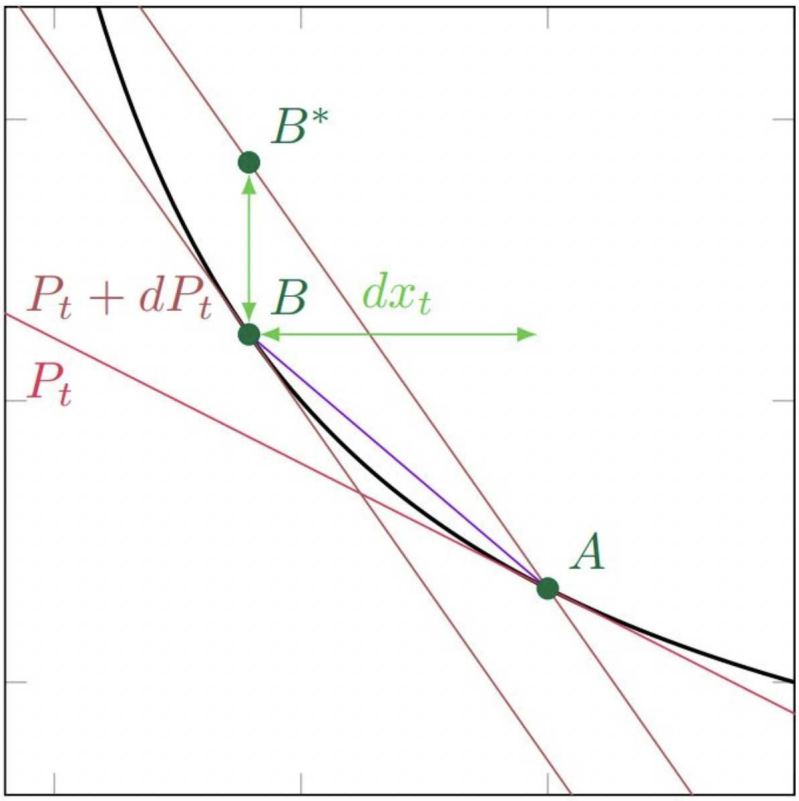
This concept is thoroughly examined in a paper published in August 2022 by Jason Milionis, Ciamac Moallemi, Tim Roughgarden, and Anthony Lee Zhang (MILIONIS et al., 2022). The paper delves into sophisticated mathematical theories, including stochastic processes and the Black-Scholes model, to rigorously define and analyze LVR.

AMMs generally offer less favorable prices compared to CEXs. Imagine you are providing liquidity for the USDC-wETH pair. If the price of ETH on a CEX like Binance goes up, the AMM (like Uniswap V2) will not immediately recognize this price change. It only adjusts its price when numerous buy orders come in. During this process, your liquidity pool sells ETH at a lower price than the new Binance price, thus reducing its ETH holdings. By the time Uniswap's price aligns with Binance, your LP has incurred a loss because it could have executed those trades at the higher Binance price.

Imagine that the price of ETH rises from P_t to $P_t + \Delta P_t$. The AMM responds

by trading from point A to point B, resulting in the sale of Δx_t ETH. However, if these trades had taken place at the new price $P_t + \Delta P_t$, the AMM would have transitioned to point B^* instead. The vertical distance between B^* and B signifies the LVR. (ZHANG, 2023b)

Figure 6 – AMM loss.



Source: (ZHANG, 2023a).

In summary, LVR also corresponds to the profits made by arbitrageurs exploiting the price difference between AMM and a CEX like Binance. Arbitrageurs can buy ETH from the Decentralized Exchange (DEX) at the price corresponding to the A to B trade and sell it on Binance at the price corresponding to the A to B^* trade, thereby making a profit equal to the gap $B^* - B$. The concept of LVR represents a transfer of value out of the application, driven by the information asymmetry between the informed trader and the liquidity provider. Conversely, it acts as a valuable metric for comparing the opportunity losses between the Rebalancing Portfolio and the LPs portfolio.

It is also extremely important to note that by definition LVR does not take into account transaction fees, therefore it takes into account that it is always an arbitrage opportunity since the price of the token in an AMM will always be different about the price of tokens on a CEX. Works that calculate liquidity losses for arbitration taking into

account transaction fees are being developed, one of them is *Automated Market Making and Arbitrage Profits in the Presence of Fees* (JASON MILIONIS; ROUGHGARDEN, 2023), which defines a probability model to estimate whether a certain block would have an arbitration opportunity.

Equations and calculation implementations from the BOOK will be discussed in chapter 4.

3 REQUIREMENTS AND CONCEPTUAL FRAMEWORK

This chapter describes the requirements and conceptual framework for application development. The framework includes general requirements that guide the scope of the project and specific functional and non-functional requirements that shape the development of the backend and front-end applications.

3.1 GENERAL REQUIREMENTS

The general requirements establish the fundamental objectives and constraints of the project:

- **Accuracy of Calculations**: The system must provide precise and reliable calculations of LVR to ensure that the insights derived from the application are actionable and trustworthy.
- **Historical Data Integrity**: The application must handle extensive historical data with integrity, ensuring data is consistently formatted, stored, and retrieved without loss or corruption.
- **Scalability**: The backend must be capable of scaling to accommodate large datasets that grow over time as more trading data becomes available and support multiple blockchains and Exchanges.
- **User Accessibility**: The frontend interface must be user-friendly, allowing users with various levels of technical expertise to navigate and utilize the application effectively.
- **Privacy**: The user does not need to show login data. No need to execute transactions or connect to user wallets.

3.2 FUNCTIONAL REQUIREMENTS

Functional requirements specify the behaviors the system should exhibit:

- **Data Collection**: The system must automatically fetch historical trading and liquidity data from Uniswap v2 pools.
- **LVR Calculation**: It should calculate LVR by analyzing price fluctuations, trading volumes, and liquidity changes over specified time intervals.
- **Data Visualization**: The front end should present LVR calculations in a clear and interpretable format, such as graphs, facilitating easy analysis and decision-making.

3.3 NON-FUNCTIONAL REQUIREMENTS

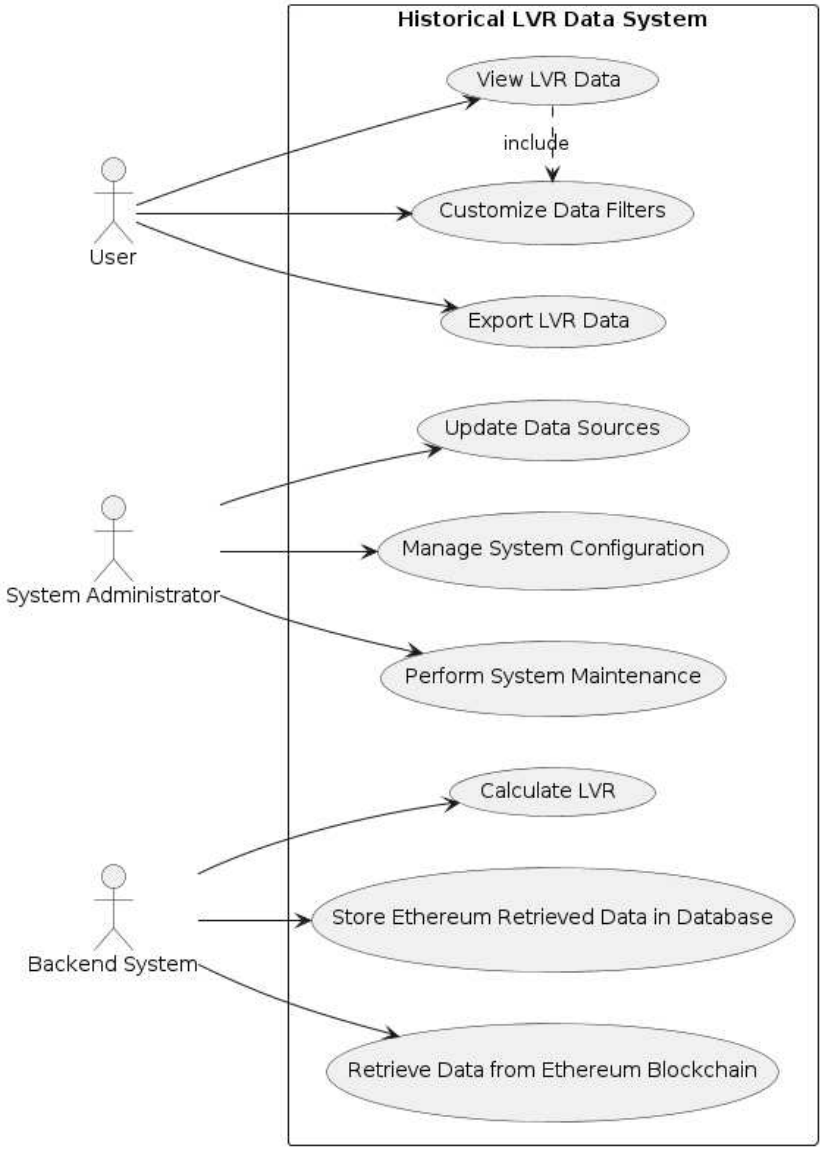
Non-functional requirements ensure that the system operates according to defined standards without specifically describing functionalities:

- **Performance**: The system should perform LVR calculations and generate reports swiftly, ideally within seconds of user requests.
- **Usability**: The interface should be clear and simple, minimizing the learning curve and enhancing user engagement.
- **Maintainability**: The codebase and system architecture should be well-documented and structured to facilitate easy updates and maintenance.
- **Compatibility**: Ensure compatibility with major operating systems and devices to reach a broader user base.

The diagram in figure 7 illustrates how different actors such as Users, System Administrators, and the Backend System engage with the system to perform various tasks essential for the operation and utilization of the system.

The use case diagram 7 depicts the interactions and functionalities of the Historical LVR Data System. Users can view LVR data, customize data filters, and export LVR data. System Administrators are responsible for managing system configuration, performing system maintenance, and updating data sources. The Backend System retrieves data from the Ethereum blockchain, stores the retrieved data in the database, and calculates LVR.

Figure 7 – Use Case Diagram for the Historical LVR Data System: Interactions and Functionalities



Source: Author

4 DEVELOPMENT AND IMPLEMENTATION

This chapter delves into the comprehensive process of developing a solution for analyzing Historical LVR data in AMMs, with a particular focus on Uniswap V2.

First, it will be discussed preliminary research into how to calculate LVR. The implementation involved extracting Uniswap V2 data using Dune Analytics, encompassing reserves, swaps, mints, and burns for a specific pool, to calculate LVR through 2 methodologies. Subsequently, a streamlined approach to data utilization was investigated.

With a clear understanding of the necessary resources, we determined the appropriate tools and designed the development strategy for a web application. The backend of the application is built with Ruby on Rails, providing a JSON API, while the frontend utilizes Next.js for an interactive user interface and display of the historical LVR data. The backend incorporates extensive functionalities such as background jobs and the execution of Python code inside Ruby, the implementation of TimescaleDB for leading with large time-series databases.

This chapter will cover the detailed steps and decisions taken throughout the development process, from initial research to the implementation of the web application, providing insights into both the challenges faced and the solutions devised.

4.1 PRELIMINARY RESEARCH

Given the novelty of the LVR concept, there is a significant lack of comprehensive data available on the internet. However, the majority of the existing LVR calculations pertain to Uniswap V2. Therefore, we decided to focus our initial efforts on Uniswap V2. This decision was driven by several factors:

- Uniswap V2's extensive adoption and robust data availability.
- The well-documented nature of Uniswap V2, facilitates understanding and data extraction.
- The relevance of Uniswap V2 in the current DeFi ecosystem, making our research applicable to a broad audience.

Our research began with a thorough analysis of Uniswap V2's mechanics, including its liquidity pools, swaps, mints, and burns. We examined how these operations impact the overall LVR calculation and identified the specific data points needed for accurate measurement.

To facilitate data extraction and processing, we initially utilized Dune Analytics, a powerful tool for querying blockchain data. Through Dune, we were able to retrieve detailed records of reserves, swaps, mints, and burns for a specific pool within Uniswap

V2. This data served as the foundation for our LVR calculations, enabling us to test various methodologies and refine our approach.

However, during this phase, we encountered several challenges with Dune Analytics. The querying process proved to be slow and expensive, which limited our ability to collect extensive data for comprehensive LVR calculations. Consequently, we decided to use Dune Analytics only for initial calculations and proof of concept.

4.2 METHODOLOGIES FOR LVR CALCULATION

In this phase, research on different methods to calculate LVR in AMMs was conducted. As seen in Chapter 3, LVR can be defined as the proportion of value lost by liquidity providers in an AMM due to arbitrage activities or the proportion of value lost by liquidity providers that occurs whenever an AMM has an outdated (stale) price compared to some other trading platform. In the context of DeFi, arbitrageurs exploit price discrepancies between the AMM's asset prices and the broader market prices. As they do this, they rebalance the asset prices within the AMM, ensuring they align with the market. And said that, this rebalancing often results in a loss for the liquidity providers because they end up trading their assets at less favorable prices. Each price movement LVR (empirical version) can be expressed as:

$$\mathrm{LVR}_n = \sum_{i=1}^{n} a_i \cdot (p_i - q_i), \tag{3}$$

where a_i amount of the token traded, p_i external market price (CEX), q_i average AMM price for the trade.

The Automated Market Making and Loss-Versus-Rebalancing paper (MILIONIS et al., 2022), derives an expression for calculating LVR in AMMs with highly advanced mathematical content, including stochastic process theory and the Black-Scholes model. The following equation, from the the paper, defines the profits and loss ($P\&L$) in a pool considering a Rebalancing Portfolio:

$$\underbrace{x_t P_t + y_t - (x_0 P_0 + y_0)}_{\mathrm{P\&L}_t} = \int_0^t x_s \, dP_s, \quad \forall\, t \geq 0. \tag{4}$$

As an example, imagine a pool composed of ETH and USDC that at an instant $t = 0$ has $x = 1$ (1 ETH) and $y = 1000$ (1000USDC), and the value of ETH is 1000\$ (P), as 1 UDSC = 1\$, on the instant $t = 0$ the pool will have 1 * 1000\$ + 1000\$ = 2000\$. Now imagine that in the instant $t = 1$ the ETH value is 4000\$, so the value, the pool value in this instant will be 1 * 4000\$ + 1000\$ = 5000\$. So the $P\&L$ in this case would be 3000\$. It is important to understand that this formula itself is only focusing on the P&L of the Rebalancing Portfolio from time 0 to time t.

$$\text{P\&L}_t = V_t - V_0 + \text{FEE}_t \tag{5}$$

And the pool profits and losses can be also defined by the equation 5, being the $V_t - V_0$ being the pool value in the instant t and 0, and FEE_t the eaned fees.

$$R_t = V_0 + \int_0^t x^*(P_s)\, dP_s, \quad \forall\, t \geq 0. \tag{6}$$

Being V_0 the initial amount of assets provided as liquidity, we can construct a Rebalancing Portfolio, as indicated in the equation above. The equation expresses the value of the rebalancing strategy R_t at any time t as the sum of V_0 and the cumulative gains or losses from following the optimal trading strategy $x^*(P_s)$ as the asset price changes over time. This strategy aims to replicate the performance of an optimally managed portfolio by dynamically adjusting the holdings in response to market price movements. LVR is defined as the difference between this Rebalancing Portfolio and the LP portfolio provided to the DEX.

From the theorem 1 in the Automated Market Making and Loss-Versus-Rebalancing paper, we have:

$$\text{LVR}_t = \int_0^t \ell(\sigma_s, P_s)\, ds, \quad \text{where} \quad \ell(\sigma_s, P_s) = \frac{\sigma_s^2 P_s}{2} y'(P_s) \geq 0,$$

And for Uniswap V2:

$$\frac{\ell(\sigma, P)}{V(P)} = \frac{\sigma^2}{2}\theta(1-\theta). \tag{7}$$

And as the tokens in Uniswap V2 has the same weight, we can take $\theta = 1/2$, so:

$$V(P) = 2L\sqrt{P}, \quad \ell(\sigma, P) = \frac{L\sigma^2}{4}\sqrt{P}, \quad \frac{\ell(\sigma, P)}{V(P)} = \frac{\sigma^2}{8}.$$

$$\text{LVR} = \frac{\sigma^2}{8}. \tag{8}$$

This shows that the constant product market maker allows for straightforward expressions for LVR. Specifically, $\frac{\ell(\sigma,P)}{V(P)}$, which represents the loss per unit time as a fraction of the mark-to-market pool value, is simply $\frac{1}{8}$ times the instantaneous variance.

LVR_t can also be interpreted as the losses from a delta-hedged LP position, excluding fees. In other words, a strategy that takes a long position in the CFMM LP position and a short position in the rebalancing strategy incurs a cost of $V_t - R_t$ at time t, ignoring any fees collected.

$$V_t - V_0 = (R_t - V_0) - \text{LVR}_t \tag{9}$$

$$\text{LVR}_t = R_t - V_t \tag{10}$$

$$\text{delta-hedged LP P\&L}_t = \text{LP P\&L}_t - R_t = \underbrace{\text{FEE}_t - \text{LVR}_t}_{\text{fees minus LVR}}. \tag{11}$$

The LVR definitions seen in equations 3 and 8 will be used to calculate the LVR, the empirical equation seen in 3, to calculate empirically we need data from all trades in a certain pool and also the value of the pool's tokens in a CEX, while the 8 equation defines that the LVR is just 1/8 of the square of the variance of the token's value in the pool. Taking this into account, the 8 equation will be used in the app for the final calculation of the LVR while the 3 and 11 equations will be used mainly to analyze results and system accuracy.

We can exemplify on how to apply the method from the equation 8. Consider a USDC-ETH pool pair with a daily volatility of 5% ($\sigma = 0.05$) and a liquidity of \$100,000,000. The LVR can be calculated as follows:

$$\text{LVR} = \frac{0.05^2}{8} = 3.125\,\text{bp}\,(0.0003125\%)$$

To determine the loss in dollars, multiply the LVR by the pool's liquidity at that moment:

$$0.0003125 \times 100{,}000{,}000 = 31{,}250\,\text{USD}$$

This would be a daily LVR considering the $\sigma = 0.05$, and for the historical LVR, we could calculate the daily LVR for our hourly LVR and do an accumulated sum.

4.3 ANALYSIS OF UNISWAP V2 DATA

Given the limitations of Dune Analytics, we explored alternative data sources and methods to enhance our data collection process. We identified the Uniswap V2 subgraph as a more efficient and cost-effective solution. The Uniswap V2 subgraph provides access to a wealth of data through a GraphQL API, enabling us to query and retrieve the necessary information more efficiently.

The Uniswap V2 subgraph offered several advantages:

- **Efficiency:** The GraphQL API allowed us to execute complex queries and retrieve large datasets quickly and efficiently.
- **Cost-effectiveness:** Using the subgraph significantly reduced our data collection costs compared to Dune Analytics.
- **Comprehensive data:** The subgraph provided detailed information on liquidity pools, swaps, mints, burns, and other relevant events within Uniswap V2.

While the Uniswap V2 subgraph addressed many of our data collection needs, there were still some gaps and limitations. To fill these gaps, we decided to supplement the subgraph data with information directly from the Ethereum blockchain using Infura's RPC (Remote Procedure Call) service. Infura's RPC allowed us to access specific blocks and transactions, ensuring we had complete and accurate data for our LVR calculations.

By combining data from the Uniswap V2 subgraph and Infura's RPC, we were able to build a comprehensive dataset for Uniswap V2 swap and reserves. This approach provided us with the necessary data to accurately calculate LVR and enabled us to overcome the challenges faced during our preliminary research.

In summary, Uniswap V2 data analysis involved an analysis of the best tool for data collection. Initially relying on Dune Analytics, we switched to the Uniswap V2 subchart for its efficiency and cost-effectiveness. Supplementing this with data from Infura's RPC ensured we had a complete data set for our LVR calculations.

4.4 TECHNOLOGY STACK AND TOOL SELECTION

During the development of the project, three programming languages were an integral part of our strategy: Python, Ruby, and JavaScript. Python was used because of its easy data analysis capabilities, allowing us to analyze and validate multiple data sources and decide which source is best for the project – whether through a GraphQL subgraph, direct queries to Dune Analytics, or access to blockchain data via RPC. This validation process ensured the choice of the most appropriate data sources. Ruby was used to power the back-end of the application, with its effective data management. JavaScript, with its broad support for web technologies, was chosen to develop the dynamic and responsive front end, enhancing user interaction and experience.

4.4.1 Validation and Data Analysis

For the initial validation and data analysis, Python was employed due to its robust libraries and tools that excel in data manipulation and analysis. Jupyter Notebooks were used extensively to explore and prototype data-related operations. The Pandas library facilitated data cleaning, transformation, and analysis, which were critical in the early stages of the project to ascertain the integrity and usability of the data extracted from blockchain sources. This setup provided a flexible environment for testing different data sources and methodologies effectively.

4.4.2 Backend

In the backend, the Ruby on Rails framework was chosen primarily due to the developers' familiarity and expertise with the framework. However, the choice was also

strategic, as Ruby on Rails is renowned for its robustness and is utilized by technology giants such as GitHub, Airbnb, and Shopify. It offers a comprehensive suite of tools that are ideal for backend development, including:

- **Convention over Configuration:** Rails accelerates development by providing a standard structure for web applications.
- **Integrated Testing Framework:** Rails includes a built-in framework for writing and running tests, which is essential for maintaining code quality and reliability.
- **Active Record:** This ORM framework simplifies data manipulation and querying in the database, making it easier to work with complex data structures necessary for LVR calculations.

Rails extensive library of gems and plugins also allowed for easy integration of additional functionalities, such as background job processing with SolidQueue, which was crucial for handling asynchronous tasks like data fetching and processing.

4.4.3 Frontend

For the frontend, the Next.js framework was selected, which enhances the React library to produce high-quality web applications. Next.js was chosen for several compelling reasons:

- **Server-Side Rendering (SSR):** Next.js provides SSR capabilities, which improve the performance and SEO of web applications by rendering JavaScript content on the server before sending it to the client.
- **Static Site Generation (SSG):** Next.js supports SSG, allowing pages to be rendered at build time and served as static HTML, which speeds up load times and enhances user experience.
- **API Routes:** Next.js facilitates the creation of API endpoints within the same application, simplifying the architecture and reducing the development overhead.

In short, the technology stack selected for the project was chosen thinking mainly about the amount of data extracted and how we would extract this data. More details about the data acquisition and processing will be explained in section the next section.

4.5 BACKEND DEVELOPMENT

The backend of the project serves as the backbone for data processing and API services. This section details the implementation strategies and technologies deployed to manage the complex data flows characteristic of our blockchain analytics application.

4.5.1 Database Configuration and Management

At the heart of our backend is PostgreSQL, a relational database management system chosen for its reliability and performance. To handle time series data management needs for over 2,000 pools and growing, we use PostgreSQL with TimescaleDB.

TimescaleDB is a PostgreSQL extension designed specifically for time series data. It enables automatic partitioning over time through hypertables, which significantly improves the performance of data insertions and complex queries (OLES, 2019). TimescaleDB is very well integrated with PostgreSQL, allowing us to leverage SQL capabilities while benefiting from performance improvements tailored to time series LVR.

PostgreSQL uses SQL, a standardized programming language used to manage and manipulate relational databases. It allows for querying, updating, inserting, and deleting data, as well as creating and modifying the structure of database systems. (SHIELDS, 2020)

Below we have an example of creating a SQL table and a hypertable.

```
1
2 CREATE TABLE uniswap_v2_reserves (
3     time timestamptz NOT NULL,
4     address varchar NOT NULL,
5     reserve0 numeric(20,10) DEFAULT 0,
6     reserve1 numeric(20,10) DEFAULT 0,
7     reserve_usd numeric(20,10) DEFAULT 0
8     ...
9 );
10
11 CREATE INDEX index_uniswap_v2_reserves_on_address ON uniswap_v2_reserves
     (address);
12 CREATE INDEX index_uniswap_v2_reserves_on_time ON uniswap_v2_reserves(
     time);
13 CREATE UNIQUE INDEX index_uniswap_v2_reserves_on_address_and_time ON
     uniswap_v2_reserves(address, time);
14
15 SELECT create_hypertable('uniswap_v2_reserves', 'time',
     chunk_time_interval => INTERVAL '1 month');
```

Listing 4.1 – SQL query to create uniswap v2 reserves table and timescale hypertable.

4.5.2 Integration of Ruby with Python Tools

As it was decided that the subgraph would be the best source for data collection, when querying it was necessary to paginate the uniswap v2 subgraph. There are some tools for this pagination that work very well, such as Subground (Python), but none of

them are for Ruby. To overcome this impasse, PayCall was used, a Ruby library that enables the execution of Python code within a Ruby environment.

- **PyCall:** PyCall is a library that allows Ruby applications to interactively call Python functions. We used PyCall to incorporate Python-based tools like Subground, which enabled efficient pagination and querying of the Uniswap V2 subgraph data directly from our Ruby application. This cross-language integration was vital for optimizing our data collection workflows.

```
require "pycall/import"
require "pycall"

class SubgroundsBase < BaseService
  attr_reader :sg, :subgraph_url

  def initialize(subgraph_url)
    @sg = PyCall.import_module("subgrounds").Subgrounds.new
    @subgraph_url = subgraph_url
  end

  memoize def subgrounds
    sg.load_subgraph(subgraph_url)
  end
end
}
```

Listing 4.2 – Subgrounds setup in rails (executing Python code inside Ruby environment)

4.5.3 Optimizing Query Performance

To efficiently manage the dynamic and voluminous data from DEXs, we implemented materialized views within PostgreSQL. These views store precomputed LVR calculations, enabling rapid query responses and reducing database load during high-traffic periods. In other words, instead of calculating the pool LVR every time we receive a frontend request, we calculate the LVR one time per day and create a table with the results, so we a kind of cached LVR.

4.5.4 Asynchronous Processing

To handle the high throughput of data updates and maintain responsiveness, we implemented asynchronous jobs using SolidQuee. With a async tasks job we can manage tasks that are resource-intensive and time-consuming without blocking the main thread, thus keeping the application performant and responsive.

```
class SyncUniswapV2ReservesJob < ApplicationJob
  queue_as :default

  def perform
    Pool.where(protocol: "uniswap_v2", blockchain: "ethereum").find_each
    do |pool|
      SyncUniswapV2PoolReservesJob.perform_later(pool.address)
    end
  end
end
}
```

Listing 4.3 – Sync Pools Job

- **Background Jobs:** Routine tasks such as updating pool information, pool reserves, and refreshing materialized views with new LVR calculations are handled asynchronously. These jobs ensure that the database is regularly updated with the latest data from all active Uniswap V2 pools on the Ethereum network.

4.5.5 RESTful API Design

The backend communicates with the frontend through a RESTful API, structured to facilitate clear and effective data exchange. The API endpoints are designed to:

- Provide comprehensive listings of pools and DEXs.
- Perform on-demand LVR calculations for each pool.

Data exchanged through the API is formatted in JSON, performing a lightweight data transfer that can be easily consumed by web applications. This API design supports modularity and ease of integration with front-end services.

In short, the backend development of our system focused on creating a robust infrastructure and thinking about the scalability of the application, capable of handling large-scale data operations efficiently.

4.5.6 Calculating LVR

The LVR calculation was performed using SQL. A materialized view of the query was built, and a simplified version of the query can be seen in the 8 equation. Note that LVR can be calculated multiple times in the same pool when there is more than one risk asset. A complete version of the calculation will be included in the attachments.

```
WITH ranked_reserves AS (
    SELECT
      address,
      time,
```

```
5       reserve_usd,
6       reserve0 / NULLIF(reserve1, 0) AS token1_price,
7       LAG(reserve0 / NULLIF(reserve1, 0)) OVER (PARTITION BY address
    ORDER BY time) AS previous_token1_price
8     FROM
9       uniswap_v2_reserves
10    WHERE
11      address = '#{pool_address}'
12  ),
13  volatility_and_lvr AS (
14    SELECT
15      time,
16      address,
17      ((token1_price - previous_token1_price) / NULLIF(
    previous_token1_price, 0)) * 100 AS token1_volatility_percentage,
18      reserve_usd * (POWER(((token1_price - previous_token1_price) /
    NULLIF(previous_token1_price, 0)) * 100, 2) / 8) / 10000 AS
    LVR_in_dollars
19    FROM
20      ranked_reserves
21  ),
22 SELECT
23  address,
24  token1_volatility_percentage IS NOT NULL,
25  time,
26  SUM(LVR_in_dollars) OVER (PARTITION BY address ORDER BY time) AS
    LVR_cumsum
27 FROM
28  volatility_and_lvr
29 WHERE
30 token1_volatility_percentage IS NOT NULL
```

Listing 4.4 – SQL query to calculate LVR in Uniswap V2 pool.

4.6 FRONTEND DEVELOPMENT

The initial phase of front-end development involved creating a design and user flow diagram to ensure a seamless user experience. The main goal was to provide users with an easy-to-navigate interface that allows them to effortlessly browse a list of DEX pools and view detailed LVR analytics for each pool.

- **User Flow:** The user starts at a dashboard that lists all available DEX pools. Each pool entry provides a brief overview, including the pool's name and a snapshot of its current LVR status. Users can select any pool to view a detailed graph representing the historical LVR data.

- **Design Considerations:** The interface is designed to be clean and minimalistic, reducing visual clutter to focus user attention on the data. Important actions and data points are highlighted through the use of contrast and color, ensuring that users can quickly identify the most pertinent information.

4.7 SYSTEM ARCHITECTURE AND DESIGN

The system architecture for the application designed to calculate LVR comprises a backend integrated with a dynamic and responsive frontend. The architecture ensures efficient data processing, real-time analytics, and seamless user interactions. Figure 8 illustrates the overall system design.

The system consists of three main components: the Frontend System, the Backend System, and the Workers.

The **Frontend System** is built using a React web client, providing an intuitive and interactive user interface. Users access the frontend through the internet, allowing them to interact with the application seamlessly. The frontend communicates with the backend via JSON over HTTP, sending requests and receiving responses.

The **Backend System** is the core of the application, implemented with Ruby on Rails and Docker for containerization. It includes several key subcomponents:

- An application server that handles incoming requests from the frontend.
- Python code executed within the Ruby environment, facilitating complex data analysis and calculations.
- Integration with external data sources such as the Subgraph to retrieve necessary data for LVR calculations.
- A PostgreSQL database with TimescaleDB extension for efficient storage and querying of time-series data.
- A caching layer to optimize performance by temporarily storing frequently accessed data.
- Background jobs managed by a worker system to handle asynchronous tasks, such as data fetching and processing, without blocking the main application flow.

The flow of data and interactions is as follows:

1. **User Interaction**: The user accesses the web client through the internet, performing actions such as viewing LVR data or customizing filters.
2. **Frontend Requests**: The frontend sends requests to the backend in JSON format.

3. **Backend Processing**: The application server in the backend processes these requests, possibly involving the execution of Python code for complex computations.

4. **Data Retrieval and Storage**: The backend retrieves data from the Subgraph or the PostgreSQL database (enhanced with TimescaleDB for time-series data). If necessary, it stores new data or updates existing data in the database.

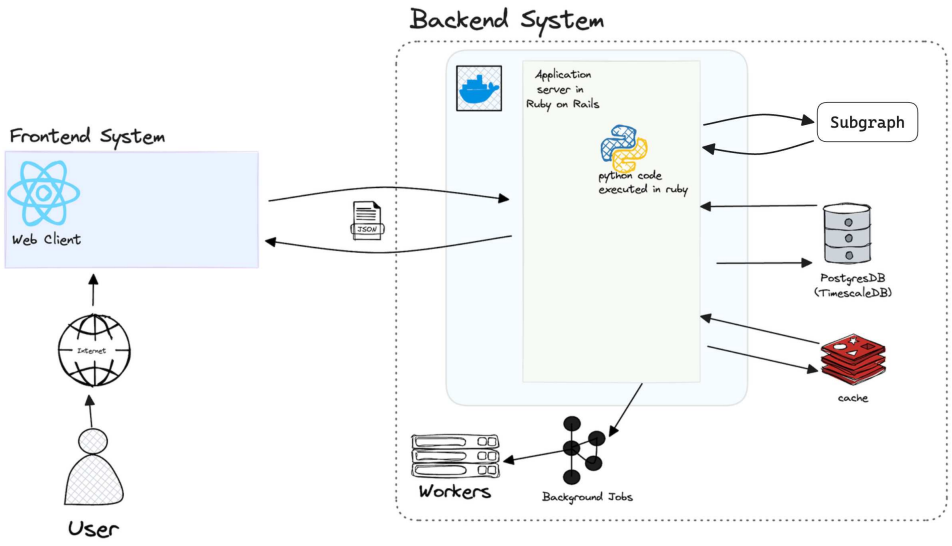
5. **Caching**: Frequently accessed data is cached to improve response times for future requests.

6. **Background Jobs**: Tasks that require heavy computation or data fetching from external sources are handled by background jobs, managed by workers, ensuring that the main application remains responsive.

7. **Response to Frontend**: The backend sends the processed data back to the frontend in JSON format, which the web client uses to update the user interface.

This architecture provides a robust and scalable solution for calculating and presenting LVR data, ensuring that users have access to real-time, accurate information through a responsive interface.

Figure 8 – System Design



Source: Author

The system combines the strengths of Ruby on Rails and Python, integrated into a backend infrastructure. Using TimescaleDB to manage time series data, along with a dynamic React-based frontend, gives users access to LVR analytics with little downtime.

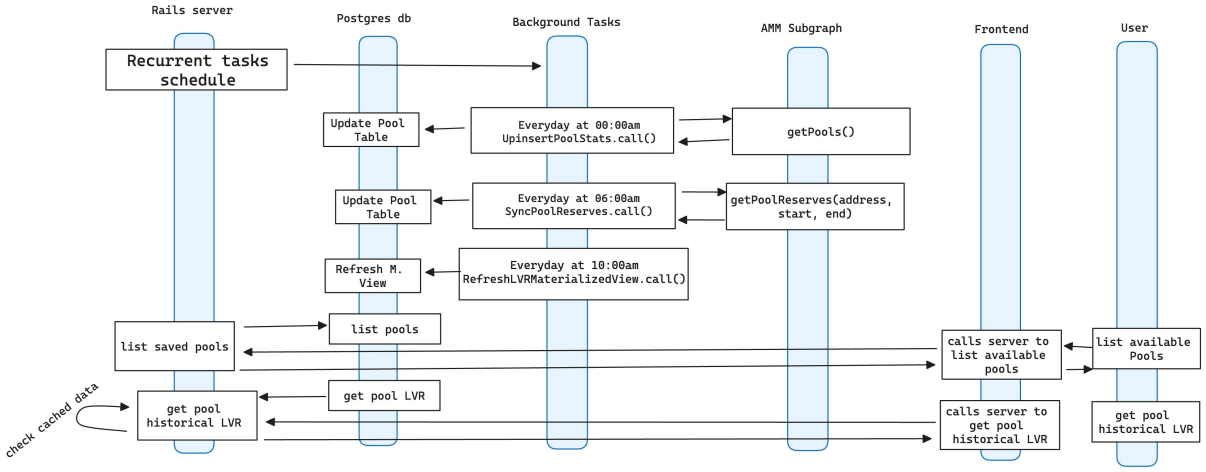


Figure 9 – System flow

Source: Author

The web application flow can be viewed in figure 9 and summarized as follows: Recurring tasks are defined on the server, with SolidQueue responsible for identifying and scheduling them. These tasks include updating the pools, synchronizing the pool reserves data (used to calculate the LVR), and updating the materialized view that stores the calculated LVR for all pools. Background tasks execute functions to query data from AMM pools and save the updated/synchronized data in the database.

Users can list, filter, and paginate through the available pools, prompting the frontend to request the available pools from the backend. When a user selects a pool, the frontend makes another request to the backend to retrieve the historical LVR data for that pool. These historical LVR requests are cached with Redis, with an expiration of 1 day. The frontend then displays a graph and summary of the pool data to the user.

5 EVALUATION AND RESULTS

This chapter evaluates the system developed for analyzing the historical LVR, focusing on Uniswap V2. The evaluation is divided into two main sections: accuracy and performance, providing an assessment of the system's effectiveness and efficiency in meeting the project's objectives.

5.1 ACCURACY

Accuracy is important in calculating LVR as it directly affects the reliability of insights provided to liquidity providers. This section reviews the measures taken to ensure the accuracy of LVR calculations, including data integrity, validation processes, and error mitigation. To analyze accuracy, we will compare the LVR calculation with the classical method and the method using the black scholes model. The classic method requires all transactions in the pool. The model that uses black scholes is very similar to the classic one, even using hourly data. Let's also compare with using data from the daily pools. For the analysis, the Uniswap V2 pool 0xb4e16d0168e52d35cacd2c6185b44281ec28c9dc (USDC-ETH pair) was used. This pool was chosen due to the number of transactions and liquidity, one of the top 4 pools with the most liquidity on Uniswap V2.

5.1.1 Per Hour Reserves Data

Figure 10 shows the LVR in the USDC-ETH pool by applying the equation 3 that calculates the LVR empirically and uses every trade (swap) data in Uniswap V2. Figure 10 also illustrates the LVR calculation using the equation 8, using hourly reserves data.

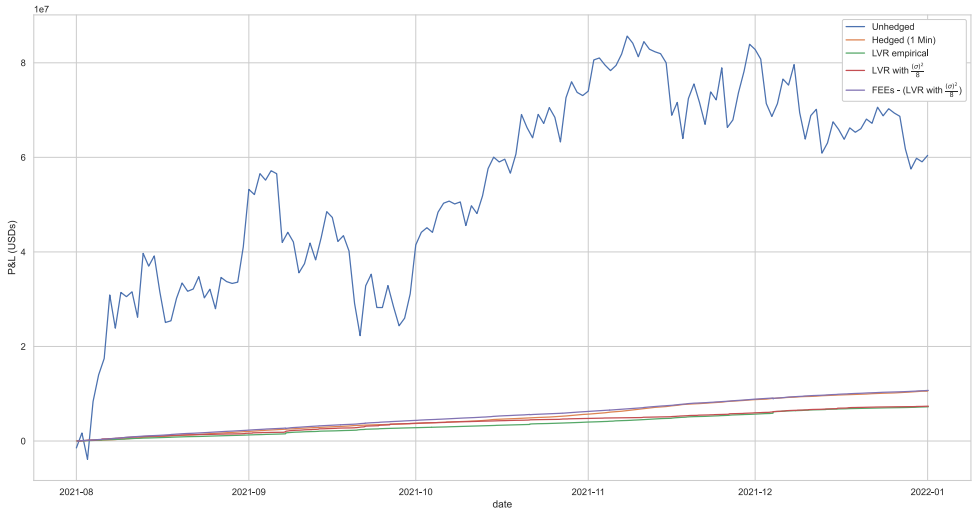


Figure 10 – Accuracy comparison: hourly data.

Source: Author

5.1.2 Per Day Reserves Data

In figure 11 we also illustrate the LVR calculated empirically but for applying the constant function market maker LVR formula from 8 we use the daily pool reserves from Uniswap V2 subgraph.

Figure 11 – Accuracy comparison: per day data.

Source: Author

The figures 10 and 11 also calculate the fees generated by the trades by taking each transaction and multiplying it by the Uniswap V2 fee rate. In the figures, it is possible to notice that we calculate the 1-minute delta hedge (application of equation 11). The delta hedge, defined in the section 5.1.2, represents the profits and losses of the pool excluding market risk, while the unhedged value represents the accumulated profits and losses (application of equation 4). As discussed in the section , the delta hedge of a pool can be defined as the difference between the earned fees and the LVR. This also allows us to arrive at another calculation of the LVR by subtracting the delta hedge from the fees. The figures also serve as proof of accuracy in the calculations. Since the empirical LVR is the most accurate (taking all transactions into account), we can calculate the error of the LVR calculated with hourly and daily data:

The Mean Absolute Percentage Error using hourly data proved to be reasonable testing on the USDC-ETH pool which has approximately 106,021,078 USD of liquidity.

In short, the method used to implement the app is with hourly reserves, for hourly data, the percentage error is reasonable, and considering that the LVR is an accumulated value, for this same pool, if we consider the first 5 months of the pool, the final percentage error was only 1.71%, while the error for daily data the error of the total LVR value after 5 months was 11.30%. The fact of using hourly data instead of

Metric	Data Per Hour	Data Per Day
Mean Absolute Error (MAE)	509,877.90 USD	1,199,968.77 USD
Root Mean Squared Error (RMSE)	598,044.88 USD	1,228,356.50 USD
Mean Absolute Percentage Error (MAPE)	16.22%	34.98%

Table 1 – Error metrics for LVR calculations considering daily data and hourly data using the equation 8 compared to LVR calculated empirically

Source: Author.

Method	LVR	Error Percentage
Empirically	7,241,142.00 USD	-
$R\dot{\sigma}^2/8$ (Data Per Hour)	7,365,471.00 USD	1.71%
$R\dot{\sigma}^2/8$ (Data Per Day)	8,059,682.00 USD	11.30%

Table 2 – LVR after 5 months for USDC-ETH pool.

Source: Author.

data from all transactions makes development easier and makes the app more scalable since we do not need to duplicate all the blockchain data to have a good result.

5.2 PERFORMANCE

System performance will be measured based on calculation time and the time a user waits to view a pool's LVR. This section evaluates system performance in terms of data processing speed, query optimization, and overall system scalability. Comparisons between different approaches are provided to highlight the improvements achieved.

5.2.1 TimescaleDB vs. Non-TimescaleDB

This performance analysis compares TimescaleDB with a standard PostgreSQL database in terms of efficiency in data insertion and processing times for calculating LVR values. It's focused on the impact of using TimescaleDB's hypertable feature with a 3-month chunk size. The tests were conducted using 1 million records for data insertion. For LVR calculations, we selected the top 10 pools in Uniswap V2 by Total Value Locked (TVL) and computed the median LVR.

Figure 12 summarizes the performance differences observed during the tests, we use PostgreSQL without the timescale extension as a reference. It can be seen that the timescale takes less time to insert temporal data and also to execute the query that calculates the LVR.

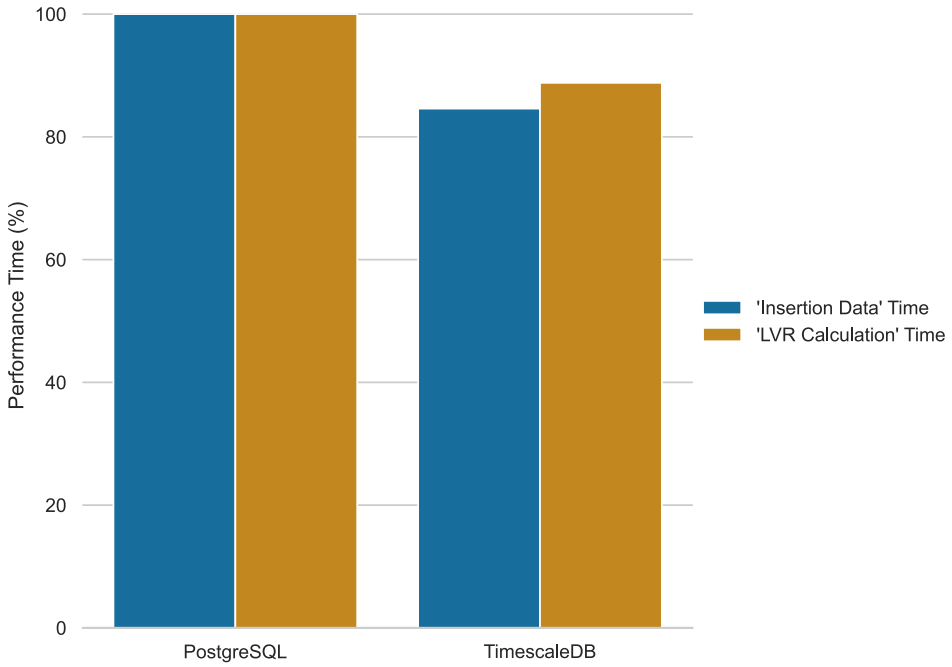


Figure 12 – Performance comparison between PostgreSQL and TimescaleDB.

Source: Author

While TimescaleDB showed a reduction in LVR calculation time by 11.22% for larger pools and a 15.42% improvement in data insertion time, its performance lag in scenarios involving smaller pools is notable but not critical. For pools with smaller TVL and fewer transactions per month, TimescaleDB underperformed due to increased planning time for queries and the time required to load each chunk from the hypertable. Although TimescaleDB's performance was approximately 30% worse in these scenarios, it is important to note that the overall time taken for LVR calculations for pools with few transactions was relatively short due to the small size of the data involved, but for pools with a huge amount of transactions, the TimescaleDB stood out. Most users typically focus on larger pools with higher TVL due to investment preferences, making this a minor concern in usage scenarios.

5.3 WEB APPLICATION OVERVIEW

The web application was designed to provide users with an easy-to-navigate interface allowing users to filter pools and also choose between dark or light themes.

5.3.1 LVR Interface

Figure 13 shows the initial page of the application. The clean and user-friendly interface allows users to quickly select a pool and explore its LVR result.

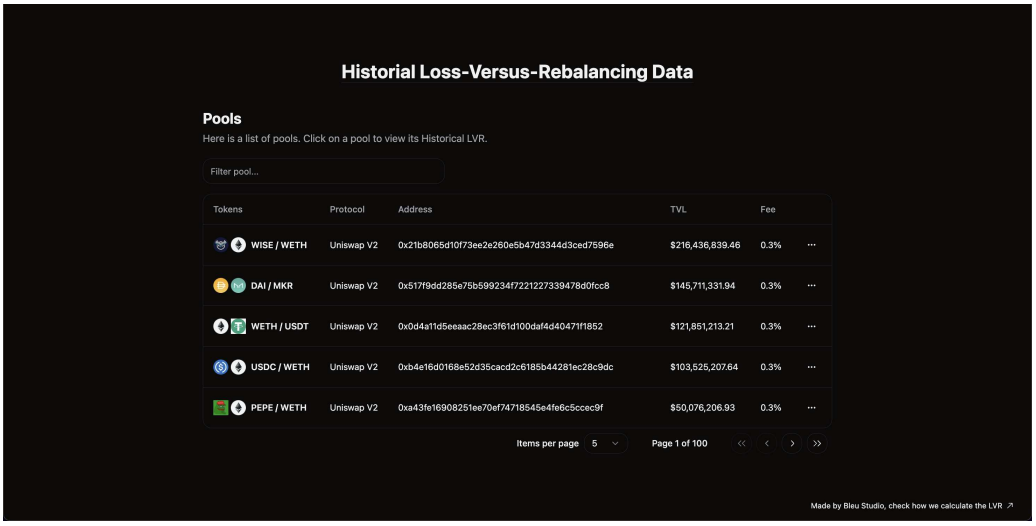


Figure 13 – Initial App page. Users can filter the pool and select one to check the LVR

Source: Author.

5.3.2 Historical Pool LVR

Figure 14 is a screenshot with the historical LVR provided for the selected USDC/WETH pool. This page allows the users to hover the mouse over the graph to see exact LVR values on certain dates.

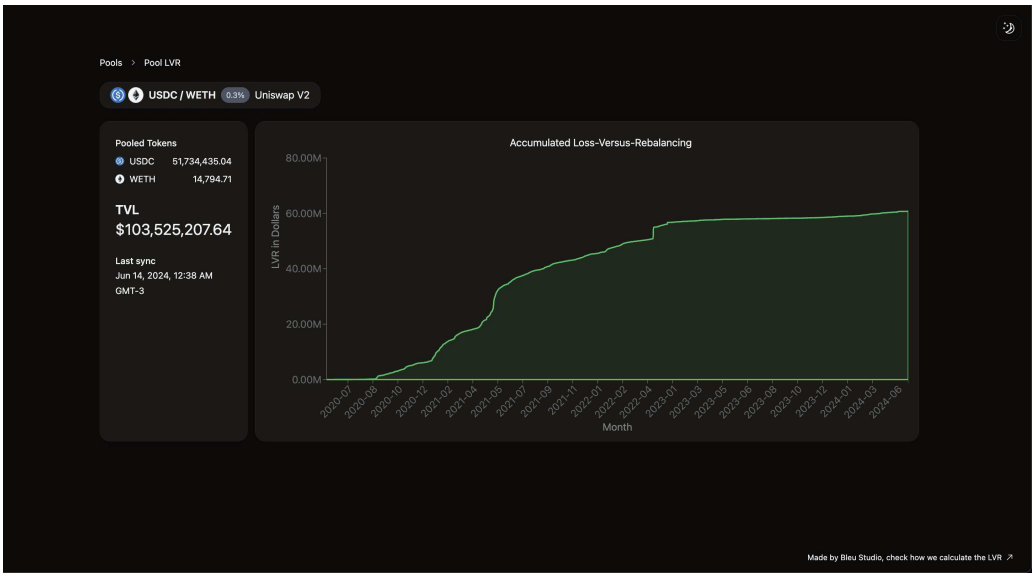


Figure 14 – LVR Page. Historical LVR for Uniswap V2 wETH-USDT pair.

Source: Author.

A feature under development is also to show how the token's volatility in the external market affects LVR, compiling LVR + token volatility in the same chart view of the historical LVR data, depicting how LVR has evolved over time.

6 CONCLUSION

The main objective of this project was to build a web application capable of showing historical LVR in AMMs. The construction of the web application went through several phases such as analysis of calculation methods, and analysis of data sources until reaching the implementation of a backend and frontend that provided the LVR's historical data. This thesis project has successfully elucidated the concept of LVR within AMM protocols, particularly focusing on the Uniswap V2 protocol. By developing the app that interfaces the stored reserves events and consumes data directly from the Ethereum blockchain, this project has achieved its primary objective. The application not only demonstrates the potential losses due to arbitrage but also serves as a practical tool for liquidity providers and researchers interested in the dynamics of AMMs.

The methodologies employed in this thesis were effective in achieving the expected results. The integration of subgraph technology proved integral in retrieving and handling blockchain data efficiently, while the use of a temporal database allowed for effective management and querying of time-series pool reserves data. These technologies combined to form an effective system that provides clear and actionable insights into the trade-offs between arbitrage-induced losses and the benefits of pool rebalancing. The use of materialized cache view techniques reduced the user's waiting time to view historical LVR data, increasing the app's performance. Background tasks cause new pools to be added automatically to the app, without the need for a third-party author to intervene.

6.1 FUTURE WORK

Looking ahead, there are several avenues for further research and development where if continuing to develop and refine these tools, we can contribute more significantly to the evolving landscape of DeFi, offering stakeholders better tools for decision-making and strategy formulation in managing their investments in liquidity pools.

6.1.1 Integrate with other AMMs and Blockchains

Future work could expand the scope of the current system to include additional AMMs across different blockchains, enhancing the comparative analysis of LVR across a broader spectrum of decentralized finance environments. One example is to integrate with Balancer V2 Weighted Geometric Mean Market Maker, one of the best AMMs on the market.

6.1.2 Combined LVR with earned fees

Furthermore, integrating a feature to calculate and display net profits from liquidity provision – accounting for fees received and LVR losses would significantly increase the usefulness of the application. This implementation would provide liquidity providers with a clearer view of their true gains and potential losses, taking into account both fee rewards and costs associated with arbitrage losses. It would also be a great addition to implement a system that takes into account the transactions and block-time of each blockchain, implementing methods such as the methods developed in Automated Market Making and Arbitrage Profits in the Presence of Fees (JASON MILIONIS; ROUGHGARDEN, 2023).

6.1.3 Add tool to estimate future LVR

the LVR can be predicted if we estimate the volatility of a risk asset, a tool in which the user can enter parameters and visualize future losses would help the LP in deciding which pool to invest in.

6.2 SUGGESTIONS FOR LIQUIDY PROVIDERS AND PROTOCOLS

To mitigate LRV, we have some approaches that AMMs can take into consideration, and the liquidity provider can also use this when deciding which protocol to invest in:

- **Dynamic Fee Structures**: Implement fee mechanisms that scale with market volatility. Higher volatility could justify higher fees, helping to compensate LPs for increased risks. Conversely, fees could be reduced during periods of low volatility to attract more liquidity.
- **Price Oracles**: Utilize reliable and high-frequency price oracles to ensure AMM prices closely match external market prices, reducing arbitrage opportunities and thereby mitigating LVR.
- **Reducing block times**: Decreasing the block times on blockchains would lead to a smaller variation, leading to more frequent but smaller arbitrage opportunities. The idea is that this would create a higher volume of trades, allowing LPs to generate larger overall returns that could potentially surpass the losses incurred from LVR.

REFERENCES

BITDEGREE. **What Is an Automated Market Maker?** [S.l.: s.n.], 2021. https://www.bitdegree.org/crypto/learn/what-is-liquidity-pool-in-crypto. Accessed: 2024-05-10.

BLOCKCHAIN. [S.l.: s.n.], 2023. https://www.ibm.com/topics/blockchain. Accessed: 2024-05-10.

CHINTAI. **Automated Market Making: A Novel Design**. [S.l.: s.n.], 2024. White Paper. Accessed: 2024-06-10. Available from: https://chintai.io/group/amm-whitepaper.pdf.

HAYDEN ADAMS NOAH ZINSMEISTER, Dan Robinson. **Uniswap v2 Core**. [S.l.: s.n.], 2020. Accessed: 2024-06-10. Available from: https://uniswap.org/whitepaper.pdf.

HAYES, Adam. **Blockchain Facts: What Is It, How Ot Works, And How It Can Be Used**. [S.l.: s.n.], 2024. https://www.investopedia.com/terms/b/blockchain.asp. Accessed: 2024-04-10.

IMPERMANENT Loss Explained. [S.l.: s.n.], 2020. https://academy.binance.com/en/articles/impermanent-loss-explained. Accessed: 2024-05-10.

JASON MILIONIS, Ciamac C. Moallemi; ROUGHGARDEN, Tim. **Automated Market Making and Arbitrage Profits in the Presence of Fees**. [S.l.: s.n.], 2023. Accessed: 2024-03-10. Available from: https://arxiv.org/abs/2305.14604.

MILIONIS, Jason; MOALLEMI, Ciamac C.; ROUGHGARDEN, Tim; ZHANG, Anthony Lee. **Automated Market Making and Loss-Versus-Rebalancing**. [S.l.: s.n.], 2022. Accessed: 2024-03-10. Available from: https://arxiv.org/abs/2208.06046.

OLES, Bart. **An Introduction to TimescaleDB**. [S.l.]: Severalnines, 2019. Available from: https://severalnines.com/database-blog/introduction-timescaledb.

ROUSE, Margaret. **What is a Liquidity Pool and How Does It Work?** [S.l.: s.n.], 2022. https://www.bitdegree.org/crypto/learn/what-is-liquidity-pool-in-crypto. Accessed: 2024-05-10.

SHIELDS, Walter. **SQL QuickStart Guide: The Simplified Beginner's Guide to Managing, Analyzing, and Manipulating Data with SQL**. Chicago, IL: ClydeBank Media LLC, 2020. ISBN 978-1642011101.

SMART Contracts. [S.l.: s.n.], 2023. `https://www.ibm.com/topics/smart-contracts`. Accessed: 2024-05-10.

SUBGRAPH. [S.l.: s.n.], 2023. `https://www.alchemy.com/overviews/what-is-a-subgraph`. Accessed: 2024-05-10.

UNISWAP. **How Uniswap works**. [S.l.: s.n.], 2020. `https://docs.uniswap.org/contracts/v2/concepts/protocol-overview/how-uniswap-works`. Accessed: 2024-04-10.

ZHANG, ANTHONY LEE. **AMM loss**. [S.l.: s.n.], 2023a. `https://anthonyleezhang.substack.com/p/automated-market-making-and-loss`. Accessed: 2024-05-10.

ZHANG, ANTHONY LEE. **Automated Market Making and Loss-Versus-Rebalancing**. [S.l.: s.n.], 2023b. `https://anthonyleezhang.substack.com/p/automated-market-making-and-loss`. Accessed: 2024-05-10.

ANNEX A – SUBGRAPH QUERIES ON UNISWAP V2 SUBGRAPH

```ruby
require "pycall/import"
require "pycall"

class SubgroundsBase < BaseService
  attr_reader :sg, :subgraph_url

  def initialize(subgraph_url)
    @sg = PyCall.import_module("subgrounds").Subgrounds.new
    @subgraph_url = subgraph_url
  end

  memoize def subgrounds
    sg.load_subgraph(subgraph_url)
  end
end
```

Listing A.1 – Base service for run subgrounds on Ruby

```ruby
module UniswapV2
  module Subgraph
    class Base < ::SubgroundsBase
      SUBGRAPH_URL = "https://api.thegraph.com/subgraphs/id/
    QmZzsQGDmQFbzYkv2qx4pVnD6aVnuhKbD3t1ea7SAvV7zE"

      attr_reader :pool_address, :start_ts, :end_ts

      def initialize(start_ts: nil, end_ts: nil, pool_address: nil)
        super(SUBGRAPH_URL)
        @pool_address = pool_address
        @start_ts = start_ts
        @end_ts = end_ts
      end
    end
  end
end
```

Listing A.2 – Base service to query Uniswap V2 data through subgraph

```ruby
module UniswapV2
  module Subgraph
    class PoolInfo < Base
      def call!
        info_query = subgrounds.Query.pair(
          id: pool_address
        )

```

```
        info_data = sg.query_df([
          info_query.reserve0,
          info_query.reserve1,
          info_query.volumeUSD,
          info_query.createdAtTimestamp
        ])

        info_data.to_dict(orient: "records").first
      end
    end
  end
end
```

Listing A.3 – Service to get a Uniswap V2 pool information

```
module UniswapV2
  module Subgraph
    class PoolReserves < Base
      def call!
        reserves_query = subgrounds.Query.pairHourDatas(
          first: 10_000_000,
          where: {
            pair: pool_address,
            hourStartUnix_gte: start_ts,
            hourStartUnix_lt: end_ts
          }
        )

        reserves_data = sg.query_df([
          reserves_query.reserve1,
          reserves_query.reserve0,
          reserves_query.reserveUSD,
          reserves_query.hourStartUnix
        ])

        reserves_data.to_dict(orient: "records")
      end
    end
  end
end
```

Listing A.4 – Service to get a Uniswap V2 pool reserves

```
module UniswapV2
  module Subgraph
    class TrendingPools < Base
      def call!
        trending_query = subgrounds.Query.pairs(
          first: 5,
```

```
          orderBy: "reserveUSD",
          orderDirection: "desc",
          where: {reserveUSD_gt: 10000, volumeUSD_gt: 1000}
        )

        trending_data = sg.query_df([
          trending_query.id,
          trending_query.token0.symbol,
          trending_query.token0.id,
          trending_query.token0.decimals,
          trending_query.token1.symbol,
          trending_query.token1.id,
          trending_query.token1.decimals,
          trending_query.volumeUSD,
          trending_query.reserve0,
          trending_query.reserve1,
          trending_query.reserveUSD,
          trending_query.createdAtTimestamp
        ])

        trending_data.to_dict(orient: "records")
      end
    end
  end
end
```

Listing A.5 – Service to get a Uniswap V2 top 5 pools

```
import pandas as pd
from datetime import datetime
from subgrounds import Subgrounds

deployment_date = datetime(2021, 8, 1)
end_date = datetime(2022, 1, 1)


# Initialize Subgrounds
sg = Subgrounds()
pool = '...'

# Load the Uniswap V2 subgraph
uniswap_v2 = sg.load_subgraph('https://api.thegraph.com/subgraphs/id/
    QmZzsQGDmQFbzYkv2qx4pVnD6aVnuhKbD3t1ea7SAvV7zE')

burns_query = uniswap_v2.Query.burns(
    first=10000000,
    where={'pair': pool,
           'timestamp_gte': int(deployment_date.timestamp()),
```

```
            'timestamp_lte': int(end_date.timestamp()),},
)

burns_data = sg.query_df([
    burns_query.amount0,
    burns_query.amount1,
    burns_query.timestamp,
])
burns_data.to_csv("burns_data_2021_08-2022_01.csv", index=False)
```

Listing A.6 – Python query to get Uniswap v2 pool burns

```
deployment_date = datetime(2021, 8, 1)
end_date = datetime(2022, 1, 1)

sg = Subgrounds()
pool = '...'

# Load the Uniswap V2 subgraph
uniswap_v2 = sg.load_subgraph('https://api.thegraph.com/subgraphs/id/
    QmZzsQGDmQFbzYkv2qx4pVnD6aVnuhKbD3t1ea7SAvV7zE')

mints_query = uniswap_v2.Query.mints(
    first=10000000,
    where={'pair': pool,
            'timestamp_gte': int(deployment_date.timestamp()),
            'timestamp_lte': int(end_date.timestamp()),},
)

mints_data = sg.query_df([
    mints_query.amount0,
    mints_query.amount1,
    mints_query.timestamp,
])
mints_data.to_csv("mints_data_2021_08-2022_01.csv", index=False)
```

Listing A.7 – Python query to get Uniswap v2 pool mints

```
sg = Subgrounds()

# Load the Uniswap V2 subgraph
uniswap_v2 = sg.load_subgraph('https://api.thegraph.com/subgraphs/id/
    QmZzsQGDmQFbzYkv2qx4pVnD6aVnuhKbD3t1ea7SAvV7zE')
pool = '...'
swaps_query = uniswap_v2.Query.swaps(
    first=10000000,
    where={'pair': pool,
            'timestamp_gte': int(deployment_date.timestamp()),
            'timestamp_lte': int(end_date.timestamp()),},
```

```
)

swaps_data = sg.query_df([
    swaps_query.amount0In,
    swaps_query.amount0Out,
    swaps_query.amount1In,
    swaps_query.amount1Out,
    swaps_query.timestamp,
    swaps_query.pair.token1Price,
    swaps_query.pair.token0Price
])
swaps_data.to_csv("swaps_data_2021_12-2022_01.csv", index=False)
```

Listing A.8 – Python query to get Uniswap v2 pool reserves

ANNEX B – CALCULATING EMPIRIC LVR, LVR WITH EQUATION 8, FEES AND DELTA-HEDGED P&L.

```
# !pip install seaborn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import matplotlib.dates as mdates
from IPython.display import display, Latex,Math

# Set the seaborn style
sns.set_theme(style="whitegrid")
plt.rcParams.update({'font.size': 10})
plt.rc('text', usetex=False)

binance_data_full = pd.read_csv('...binance_ETH.csv')
binance_data_full['time'] = pd.to_datetime(binance_data_full['close_time
    '], format='ISO8601').dt.floor('T')

subgraph_merged = pd.read_csv("data/uniswap_v2_2021-08_2022-01.csv")
subgraph_merged['time'] = pd.to_datetime(subgraph_merged['time'], format
    ='ISO8601', utc=True).dt.floor('T')
subgraph_merged


dune_data = pd.read_csv('dune_data.csv')
dune_data['time'] = pd.to_datetime(dune_data['time'], format='ISO8601').
    dt.floor('T')


data = pd.merge(subgraph_merged,binance_data,on='time', how="inner")
data = pd.merge(data,dune_merged,on='time', how="inner")
data.sort_values('time', inplace=True)

data['eth_price'] = data['USDC_holding'] / data['ETH_holding']
data['mint_value'] = data['mints_amount0'] + data['mints_amount1'] *
    data['close_price']
data['burn_value'] = data['burns_amount0'] + data['burns_amount1'] *
    data['close_price']
data['pool_pnl'] = data['pool_value'].diff() + data['burn_value'] - data
    ['mint_value']

# Minutely rebalancing strategy just holds lagged ETH
data['rebal_minute_ETH'] = data['ETH_holding'].shift(1)
# Calculate PnL
data['rebal_minute_pnl'] = data['rebal_minute_ETH'] * data['close_price'
```

```
    ].diff()
39
40
41 # Construct a strategy which holds, at each time, the ETH holdings at
    the start of the hour
42 data['min5_rounded'] = data['time'].dt.floor('5T')
43 data['rebal_5min_ETH'] = data.groupby('min5_rounded')['ETH_holding'].
    transform('first')
44 # Calculate PnL
45 data['rebal_5minute_pnl'] = data['rebal_5min_ETH'] * data['close_price'
    ].diff()
46
47
48 data['hour_rounded'] = data['time'].dt.floor('H')
49 data['rebal_hour_ETH'] = data.groupby('hour_rounded')['ETH_holding'].
    transform('first')
50 # Calculate PnL
51 data['rebal_hour_pnl'] = data['rebal_hour_ETH'] * data['close_price'].
    diff()
52
53
54 data['hour4_rounded'] = data['time'].dt.floor('4H')
55 data['rebal_4hour_ETH'] = data.groupby('hour4_rounded')['ETH_holding'].
    transform('first')
56 # Calculate PnL
57 data['rebal_4hour_pnl'] = data['rebal_4hour_ETH'] * data['close_price'].
    diff()
58
59 data['date'] = data['time'].dt.date
60 data['rebal_day_ETH'] = data.groupby('date')['ETH_holding'].transform('
    first')
61 # Calculate PnL
62 data['rebal_day_pnl'] = data['rebal_day_ETH'] * data['close_price'].diff
    ()
63
64 aggdata = data.groupby('date').agg({
65     'pool_value': 'mean',
66     'pool_pnl': 'sum',
67     'rebal_minute_pnl': 'sum',
68     'rebal_5minute_pnl': 'sum',
69     'rebal_hour_pnl': 'sum',
70     'rebal_4hour_pnl': 'sum',
71     'rebal_day_pnl': 'sum'
72 }).reset_index()
73
74 aggdata['hedged_pnl_minute'] = aggdata['pool_pnl'] - aggdata['
    rebal_minute_pnl']
```

```
aggdata['hedged_pnl_5minute'] = aggdata['pool_pnl'] - aggdata['rebal_5minute_pnl']
aggdata['hedged_pnl_hour'] = aggdata['pool_pnl'] - aggdata['rebal_hour_pnl']
aggdata['hedged_pnl_4hour'] = aggdata['pool_pnl'] - aggdata['rebal_4hour_pnl']
aggdata['hedged_pnl_day'] = aggdata['pool_pnl'] - aggdata['rebal_day_pnl']

# Cumulative PnL
aggdata['cum_pool_pnl'] = aggdata['pool_pnl'].cumsum()
aggdata['cum_hedged_pnl_minute'] = aggdata['hedged_pnl_minute'].cumsum()
aggdata['cum_hedged_pnl_5minute'] = aggdata['hedged_pnl_5minute'].cumsum()
aggdata['cum_hedged_pnl_hour'] = aggdata['hedged_pnl_hour'].cumsum()
aggdata['cum_hedged_pnl_4hour'] = aggdata['hedged_pnl_4hour'].cumsum()

def format_plot(ax, ylabel):
    ax.xaxis.set_major_locator(mdates.YearLocator())
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    ax.xaxis.set_minor_locator(mdates.MonthLocator())
    ax.figure.autofmt_xdate()
    ax.set_ylabel(ylabel)
    ax.legend()

pool_fee = 0.003

data["LVR1"] = abs(data["swaps_amount1In"]*(data["close_price"] - (data['eth_price']))) # empiric lvr

lvrblackscholes = pd.read_csv("data/lvr_per_hour.csv")
lvrblackscholes['time'] = pd.to_datetime(lvrblackscholes['time'], format='ISO8601', utc=True)

data["cum_lvr"] = data["LVR1"].cumsum()
data["FEE1"] = ((data["swaps_amount0In"]) * pool_fee) + (((data["swaps_amount1In"]) * pool_fee * data["close_price"]))
data['feeminuslvr'] = (data["FEE1"] - data["LVR1"]).cumsum()
# Plot for Cumulative P&L, hedging frequency
fig, ax = plt.subplots(figsize=(20, 10))
sns.lineplot(data=aggdata, x='date', y='cum_pool_pnl', ax=ax, label='Unhedged')
sns.lineplot(data=aggdata, x='date', y='cum_hedged_pnl_minute', ax=ax, label='Hedged (1 Min)')
sns.lineplot(data=data, x='time', y='cum_lvr', ax=ax, label='LVR empirical')
sns.lineplot(data=lvrblackscholes, x='time', y='LVR_cumsum', ax=ax,
```

```
    label=r'LVR with $\frac{(\sigma)^2}{8}$')
sns.lineplot(data=data, x='time', y='feeminuslvr', ax=ax, label=r'FEEs -
    (LVR with $\frac{(\sigma)^2}{8}$)')

format_plot(ax, 'P&L (USD 1000s)')
plt.savefig('lvr_data.eps', format='eps', dpi=300)
```

Listing B.1 – Calculate LVR

```
data = reserves_data.copy()
data['time'] = pd.to_datetime(data['pairDayDatas_date'], unit='s')
data = data.sort_values(["time"])
data.drop_duplicates(inplace=True)
data.set_index('time', inplace=True)
data['eth_value'] = data['pairDayDatas_reserve0']/data['
    pairDayDatas_reserve1']
data['eth_volatility_percentage'] = data['eth_value'].pct_change() * 100
data.dropna(subset=['eth_volatility_percentage'], how='all', inplace=
    True)
data['LVR_bp'] = (data['eth_volatility_percentage']*data['
    eth_volatility_percentage'])/8
data['LVR'] = data['pairDayDatas_reserveUSD']*data['LVR_bp']/10000
data['LVR_cumsum'] = data['LVR'].cumsum()
data.reset_index(inplace=True)
data
```

Listing B.2 – Calculate LVR with equation 8

```
module UniswapV2
  module LVR
    class Calculate < ::BaseService
      attr_reader :pool_address

      def initialize(pool_address:)
        @pool_address = pool_address
      end

      def call!
        records = ActiveRecord::Base.connection.exec_query(sql_query)
        records.as_json
      end

      def sql_query
        <<-SQL
              WITH ranked_reserves AS (
                SELECT
                  address,
                  time,
                  -- reserve0,
```

```
22                -- reserve1,
23                reserve_usd,
24                reserve0 / NULLIF(reserve1, 0) AS token1_price,
25                LAG(reserve0 / NULLIF(reserve1, 0)) OVER (PARTITION BY
     address ORDER BY time) AS previous_token1_price
26              FROM
27                uniswap_v2_reserves
28              WHERE
29                address = '#{pool_address}'
30            ),
31            volatility_and_lvr AS (
32              SELECT
33                -- *,
34                time,
35                address,
36                ((token1_price - previous_token1_price) / NULLIF(
    previous_token1_price, 0)) * 100 AS token1_volatility_percentage,
37                -- POWER(((token1_price - previous_token1_price) /
    NULLIF(previous_token1_price, 0)) * 100, 2) / 8 AS LVR_bps,
38                reserve_usd * (POWER(((token1_price -
    previous_token1_price) / NULLIF(previous_token1_price, 0)) * 100, 2)
    / 8) / 10000 AS LVR_in_dollars
39              FROM
40                ranked_reserves
41            ),
42            LVR_cumsum AS (
43              SELECT
44                address,
45                time,
46                -- reserve_usd,
47                -- token1_volatility_percentage,
48                -- LVR_bps,
49                SUM(LVR_in_dollars) OVER (PARTITION BY address ORDER
    BY time) AS LVR_cumsum
50              FROM
51                volatility_and_lvr
52              WHERE
53              token1_volatility_percentage IS NOT NULL
54            ),
55            daily_last_entry AS (
56              SELECT
57                *,
58                ROW_NUMBER() OVER (PARTITION BY address, DATE(time)
    ORDER BY time DESC) AS rn
59              FROM
60                LVR_cumsum
61            )
```

```
62                SELECT
63                  time,
64                  address,
65                  LVR_cumsum
66                FROM
67                  daily_last_entry
68                WHERE
69                  rn = 1
70                ORDER BY
71                  time DESC
72          SQL
73        end
74      end
75    end
76  end
```

Listing B.3 – Calculate LVR with equation 8 in Ruby on Rails