Decentralized Exchange (DEX) and Smart Contract Detection System

Shen PENG, Zixuan GAO, Zixiang YI, Haolong LIU
School of Data Science, Lingnan University, Hong Kong, China

14 November 2024

Abstract
The first part of this paper mainly focuses on the decentralized exchange (DEX) project. It starts with an introduction to the development history of DEX, followed by an in-depth exploration of the Automated Market Maker (AMM) principle. Then, it presents detailed descriptions of this specific DEX project, including its core functions, participants, mathematical principles, and code implementation. Finally, the steps of operating the contract in Remix are demonstrated.

In the second part, we developed a smart contract detection system, which can help developers find potential vulnerabilities and problems before deploying smart contracts. Through real-time detection, developers can instantly see the feedback of the detection, which is very helpful for fixing the problems in the contract. This system also uses AI to assist users in analysing and modifying smart contracts. By calling AliCloud's Tongyi Qianwen big model API, this system can send users' smart contracts to the back-end API, return the results to the user, and provide some modifications to the user.

Keywords:
Decentralized Exchange (DEX); Automated Market Maker (AMM); Liquidity Provider (LP); Uniswap; Smart Contract Detection

1 Introduction

The emergence of decentralized exchanges (DEXs) has brought significant changes to the digital currency trading landscape. With the development of blockchain technology, DEXs aim to address the drawbacks of centralized exchanges (CEXs) and provide users with more secure and decentralized trading options. This paper will introduce a DEX project based on relevant theories and technologies, elaborating on its various aspects.

2 The Development History of Decentralized Exchanges

2.1 Centralized Exchanges (CEX)

In the context of the booming digital currency market, digital currency exchanges that offer liquidity and trade matching services for digital currency assets have witnessed remarkable growth. Although Bitcoin was initially designed to be decentralized, for a long period, due to the complexity of technological implementation and relatively high user access barriers, the majority of users still preferred trading digital currencies on centralized trading platforms. Centralized exchanges have played a crucial role in the digital currency market with a market value reaching thousands or even trillions. Many entrepreneurs have entered this field, which has inadvertently promoted the development of digital currencies.

The trading model adopted by traditional centralized exchanges is the order book model. The order book, a data structure commonly used in stock exchanges, is an order queue organized by price. It consists of a buy book and a sell book, and when conducting trade matching, it adheres to the first-in, first-out principle. Users can place buy and sell orders at fixed prices, and all these orders are stored in the order book on the blockchain. Subsequently, the system undertakes trade matching and settlement based on the orders in the order book. This model offers advantages such as good security and high transparency. However, it also suffers from issues like difficulties in reaching agreements between trading parties, resulting in low trading market efficiency. Moreover, when the selling price and the buying price fail to reach an accord for an extended period, it can significantly impact the trading liquidity.

2.2 Decentralized Exchanges (DEX)

From the establishment of the first exchange around 2010 when Bitcoin emerged until 2018, the market predominantly utilized the order book model for trading.

While centralized exchanges were flourishing, they were also confronted with numerous problems, including frequent hacker attacks and regulatory challenges from governments, which led to a lack of trust among users regarding centralized exchanges. Coupled with the limitations inherent in the order book model for trading as mentioned earlier, questions arose about whether users could keep their funds in personal wallets and utilize smart contracts for trade matching to ensure asset security and the decentralization of the entire trading process. In response to these concerns, decentralized exchanges came into being.

A decentralized exchange (DEX) can be contrasted with a centralized exchange (CEX). In a CEX, when users engage in buying and selling digital currencies, they are required to deposit their digital currencies or fiat currencies into the exchange, and the exchange then takes on the responsibility of custodying these coins. Additionally, the exchange provides digital currency liquidity, facilitates trade matching, and handles settlement and liquidation procedures. In contrast, within a DEX, users' funds are retained in their own wallets. The trading platform's primary role is limited to providing digital currency liquidity, while trade matching is accomplished through smart contracts.

2.3 Uniswap's Development

In June 2017, Vitalik published an article titled "On Path Independence", which explored the possibility of constructing a decentralized exchange on the Ethereum chain. Inspired by this, the Uniswap development team led by Hayden Adams released Uniswap V1 in November 2018 and deployed it onto the Ethereum mainnet, enabling trading between ETH and another ERC20 Token.

In May 2020, Uniswap V2 was launched, supporting trading between any two ERC20 Tokens, which triggered a significant boom in the decentralized finance (DeFi) domain.

In May 2021, Uniswap V3 was introduced, enhancing the utilization rate of Liquidity Provider (LP) funds and transforming Liquidity Provider Tokens (LPT) into Liquidity Tokens based on Non-Fungible Tokens (NFTs).

In June 2023, the draft code for Uniswap V4 was released. However, as of now, the V4 codebase is still undergoing the auditing process and has not been officially launched.

3 Principles of Automated Market Maker (AMM)

3.1 AMM Overview

Automated Market Maker (AMM) represents an algorithm or a smart contract operating on the blockchain. It enables decentralized trading of digital assets and has pioneered a novel trading approach that dispenses with the need for traditional order matching between buyers and sellers. The core mechanism of AMM lies in creating a liquidity pool through preset mathematical formulas, thereby allowing users to conduct transactions at any time they desire.

3.2 Different Types of AMM Models

3.2.1 Constant Sum Automated Market Maker (CSAMM)

CSAMM is the simplest form among AMM models. Its mathematical representation is given by the following formula:

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

One of the notable advantages of CSAMM is its ability to ensure that the relative prices of tokens remain unchanged. However, it also has a significant drawback in that it is prone to the depletion of liquidity.

3.2.2 Constant Product Automated Market Maker (CPAMM)

The Uniswap protocol employs CPAMM. Its mathematical formula is expressed as:

$$k = x * y \tag{2}$$

By determining the exchange ratio between x and y at a specific moment, automatic trading between these two tokens can be achieved. All the rules governing this process are defined within the contract and are automatically executed by the blockchain system (such as Ethereum). This model is also referred to as the "x - y - k automated market maker model". CPAMM offers several advantages, including having what can be considered "infinite" liquidity. Specifically, the relative prices of tokens will vary in accordance with buying and selling activities, and rarer tokens will command higher relative prices, thereby effectively avoiding the depletion of liquidity.

4 Introduction to This Decentralized Exchange Project

4.1 Core Functions and Participants

This project references Uniswap V2. It implements a simple DEX based on CSAMM. We implemented the core functionality of DEX, but the transaction fee functionality is discarded.

The two primary types of participants in a DEX are Liquidity Providers (LP) and Traders. Liquidity Providers play a crucial role in injecting liquidity into the trading pool by depositing their tokens. The LP Tokens represent their share of the liquidity provided. Traders utilize the liquidity in the pool to conduct transactions between different tokens.

4.2 Mathematical Principles

Let there be x units of token0 and y units of token1. token0 changes by Δx and token1 changes by Δy.

Let the total share before the increase or decrease in liquidity be T, and the increase or decrease in liquidity be S.

4.2.1 Providing Liquidity

Adding Liquidity 1. When liquidity is added to the token pool for the first time, the calculation of the LP share (ΔL) is determined by the square root of the product of the quantities of the added tokens. The formula is expressed as:

$$\Delta L = \sqrt{\Delta A * \Delta B} \tag{3}$$

2. When adding liquidity on subsequent occasions (i.e., not for the first time), the LP share is calculated based on the proportion of the added token quantity to the token reserve within the pool. Specifically, the smaller of the two proportions (calculated for each token type) is taken as the basis for determining the LP share. The formula can be written as: (L_0: Liquidity before adding, L_1 Liquidity after adding)

1.
$$\frac{x}{y} = \frac{x + \Delta x}{y + \Delta y} \implies \frac{\Delta x}{\Delta y} = \frac{x}{y}$$

2.
$$\frac{L_0}{L_1} = \frac{T}{T + S}$$

$$\Rightarrow S = \frac{(L_1 - L_0)}{L_0} T = \left(\frac{\sqrt{(x + \Delta x) \cdot (y + \Delta y)} - \sqrt{xy}}{\sqrt{xy}} \right) T = \frac{\Delta x}{x} T = \frac{\Delta y}{y} T$$

3. Therefore, the formula for the LP share is:

$$\Delta L = L \cdot \min \left(\frac{\Delta x}{x}, \frac{\Delta y}{y} \right) \tag{4}$$

i.e.

$$\Delta L = \min \left(\frac{amountA * TotalSupply}{ReserveA}, \frac{amountB * TotalSupply}{ReserveB} \right) \tag{5}$$

Removing Liquidity When a user decides to remove liquidity (ΔL) from the pool, the contract will destroy the corresponding LP share tokens and return the tokens to the user in proportion. The formula for calculating the quantity of tokens to be returned to the user is:

$$\frac{\sqrt{\Delta x \cdot \Delta y}}{\sqrt{xy}} = \frac{S}{T}$$

$$\implies \Delta x = \frac{S}{T} \cdot x, \quad \Delta y = \frac{S}{T} \cdot y$$

$$\Delta x = \frac{\Delta L}{L} \cdot x, \quad \Delta y = \frac{\Delta L}{L} \cdot y \tag{6}$$

4.2.2 Exchange

In a DEX, users have the ability to exchange one token for another. To derive the formula for calculating the quantity of token1 (Δy) that can be exchanged using a certain quantity (Δx) of token0, we rely on the constant product formula. Before the transaction takes place, the relationship is described by:

$$k = x * y$$

After the transaction, it becomes:

$$k = (x + \Delta x) * (y + \Delta y)$$

Since the value of k remains unchanged before and after the transaction, by combining these two equations, we can derive the formula for calculating Δy:

$$\Delta y = -\frac{\Delta x \cdot y}{x + \Delta x} \tag{7}$$

So we can see the Δy is determined by Δx, x, and y.

4.3 Core Elements

The core elements of this DEX can be summarized as follows:

- Any individual has the opportunity to add liquidity and thereby become an LP, receiving LP Tokens in the process.
- LPs possess the right to remove liquidity and destroy their LP Tokens at any time they choose, enabling them to retrieve their original tokens.
- Users are able to conduct transactions based on the trading pool, leveraging the available liquidity for token exchanges.

4.4 Specific Implementation

4.4.1 Function Overview

Only the core functions are listed here. See the contract file for the full code.

The following are the core functions involved in this project, along with a brief description of their respective roles:

- **addLiquidity()**: This function enables users to add liquidity to the trading pool by specifying the quantities of two different tokens (amountA and amountB). It calculates the appropriate LP share based on the provided token amounts and the existing pool conditions, and then returns the liquidity value. Additionally, it involves operations such as authorizing the contract to receive the tokens and updating the token reserves within the pool.

- **removeLiquidity()**: When users wish to remove liquidity from the pool, this function will be called, specifying the amount of liquidity they want to withdraw. The function calculates the corresponding quantities of the two tokens to be returned to the user based on the proportion of the LP share being removed and the current token reserves. It also includes steps such as destroying the LP Tokens and updating the token reserves in the pool.

- **GetAmountOut()**: Given the quantity of an input token ($amountIn$), along with the reserves of the two tokens in the pair ($reserveIn$ and $reserveOut$), this function calculates the quantity of the other token ($amountOut$) that can be obtained through the exchange. It forms the basis for determining the outcome of token transactions.

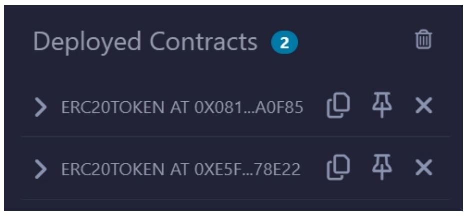
- **swap()**: This function implements the exchange between different tokens. Users specify the quantity of the token they want to exchange ($amountIn$), the token they are using for the exchange ($tokenIn$), and the minimum acceptable quantity of the output token, also known as the slippage point ($amountOutMin$). The function first validates the input parameters, determines the type of token exchange (e.g., tokenA for tokenB or vice versa), calculates the quantity of the output token based on the trading formula, and then conducts the actual exchange if the calculated quantity meets the minimum requirement. It also updates the token reserves in the pool.

- **min()**: This function is designed to obtain the minimum value between two input parameters and return the result . It is utilized in various calculations where determining the smaller of two values is necessary.

- **sqrt()**: Employing Newton's iteration method, this function calculates the square root of the input parameter and returns the result. It plays a crucial role in calculating LP shares during the process of adding liquidity, especially when dealing with the initial addition scenario.

4.4.2 Demonstration of the results of operating the contract on Remix

1. First, deploy two ERC20 tokens contracts, named tokenA and tokenB. Record the contract addresses of these two tokens.

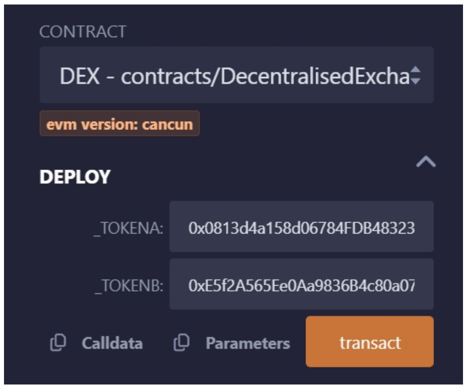

Deployed Contracts 2

ERC20TOKEN AT 0X081...A0F85

ERC20TOKEN AT 0XE5F...78E22

Figure 1: Deploy two ERC20 tokens

2. Fill in the contract addresses of the deployed tokenA and tokenB into the DecentralisedExchange contract and then deploy the DecentralisedExchange contract.


CONTRACT

DEX - contracts/DecentralisedExcha

evm version: cancun

DEPLOY

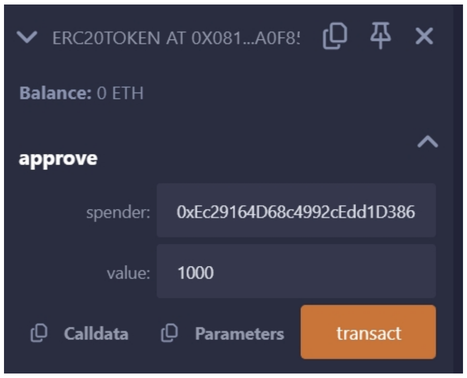
_TOKENA: 0x0813d4a158d06784FDB48323

_TOKENB: 0xE5f2A565Ee0Aa9836B4c80a07

Figure 2: Deploy DEX contract

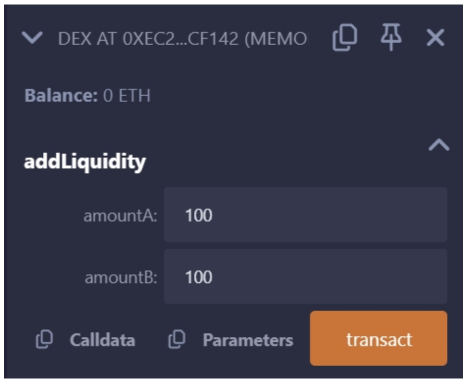
3. Call the approve() function of tokenA and tokenB respectively to authorize 1,000 units of tokens to the DecentralisedExchange contract.

4. Call the addLiquidity() function of the DecentralisedExchange contract to add liquidity to the decentralized exchange, adding 100 units to tokenA and tokenB respectively.



spender: 0xEc29164D68c4992cEdd1D386
value: 1000

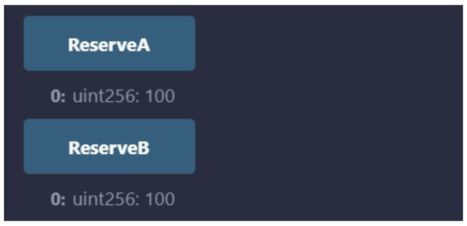
Figure 3: Authorisation



amountA: 100
amountB: 100

Figure 4: Add Liquidity

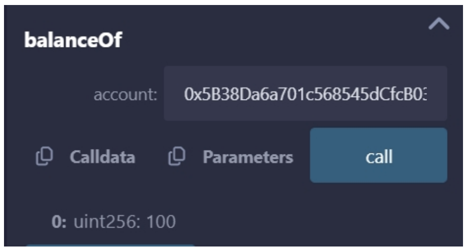
5. Check the ReserveA and ReserveB at this time, which are the token reserves of the contract.



```
ReserveA
0: uint256: 100
ReserveB
0: uint256: 100
```

Figure 5: Enter Caption

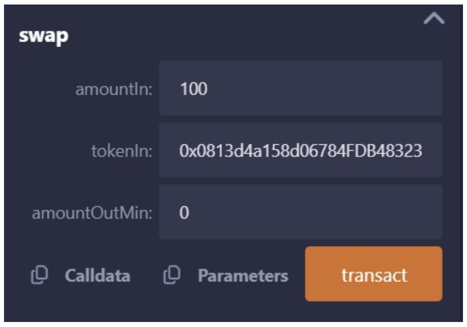
6. Call the balanceOf() function of the DecentralisedExchange contract to check the user's LP share, which is $\sqrt{100 * 100} = 100$.



```
balanceOf
account: 0x5B38Da6a701c568545dCfcB03
0: uint256: 100
```

Figure 6: Enter Caption

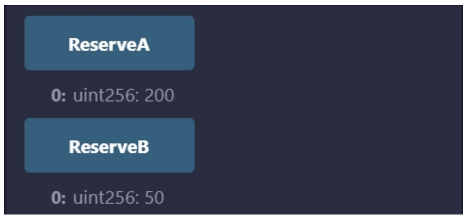
7. Call the swap() function of the DecentralisedExchange contract to conduct token trading, using 100 units of tokenA to exchange for tokenB.



```
swap
amountIn: 100
tokenIn: 0x0813d4a158d06784FDB48323
amountOutMin: 0
```

Figure 7: Enter Caption

8. Check the ReserveA and ReserveB again. The result of the previous step should be 50 units of tokenB ($\frac{100*100}{100+100} = 50$), so the final result should be 200 and 50, which is accurate.



0: uint256: 200

0: uint256: 50

Figure 8: Enter Caption

5 Smart Contract Detection System

5.1 Introduction

This system is used for smart contract detection, which can help developers find potential vulnerabilities and problems before deploying smart contracts. With real-time detection, developers can instantly see the feedback of the detection, which is very helpful in fixing problems in the contract.

5.2 Functional Description

1. Enter the front-end interface by visiting the path http://127.0.0.1:5000/analyze, which is used for the user to input the smart contract that needs to be detected

2. After the user clicks on the execution, listen to the state of the button through the browser, and input the code of the interface packaged into a .json file to send data to the back-end, after the back-end interface receives the data, for the convenience of the back-end will be the user's input of the smart contract will be saved as a contract.sol file, the back-end calls the slitherAPI interface, by analysing the solidity The backend will call the slitherAPI interface and analyse the solidity version, choose the right one and execute it, and finally return the result of the slither to the frontend. Convenient for users to operate and check the smart contract vulnerability.

3. Use AI to assist us in the analysis and modification of smart contracts, this site by calling AliCloud's Tongyi Qianwen big model API, you can send the user's smart contract to the back-end API, and the results will be returned to the user to provide users with some modification advice.

5.3 Tool Description

Operating System:Windows

Front-end:HTML, CSS, JavaScript

Backend: Python (3.0 or above) based on Anaconda, smart contract analysis tool slither, Tongyi Qianqi big model, smart contract library OpenZeppelin

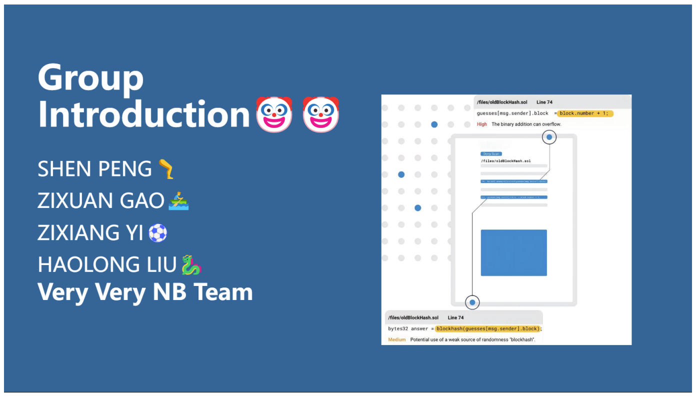
Data interaction: API calls

Web cross-domain: Flask

Browser: Chrome

5.4 System Demonstration

1.Home page overview



Group Introduction 🤡🤡

SHEN PENG

ZIXUAN GAO

ZIXIANG YI

HAOLONG LIU

Very Very NB Team

/files/oldBlockHash.sol Line 74

```
guesses[msg.sender].block  = block.number + 1;
```

High The binary addition can overflow.

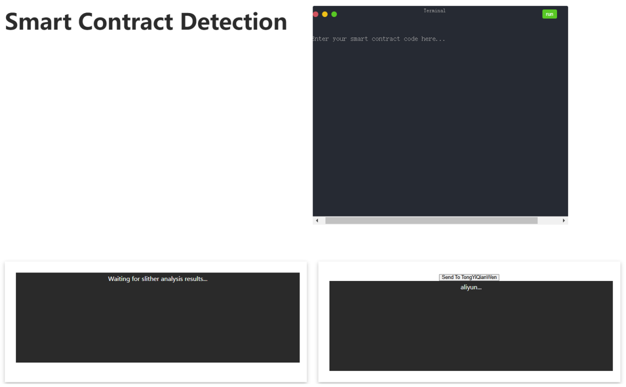
/files/oldBlockHash.sol Line 74

```
bytes32 answer = blockhash(guesses[msg.sender].block);
```

Medium Potential use of a weak source of randomness "blockhash".

Figure 9: Home Page

2.Overview of the operator interface



Smart Contract Detection

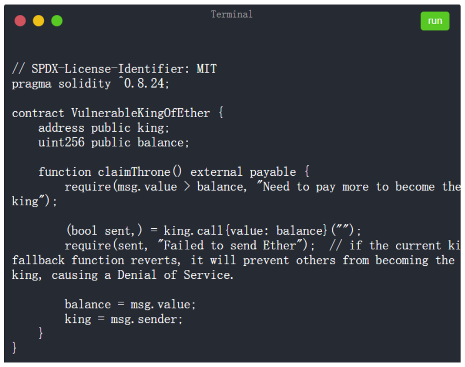
```
Enter your smart contract code here...
```

Waiting for slither analysis results...

aliyun...

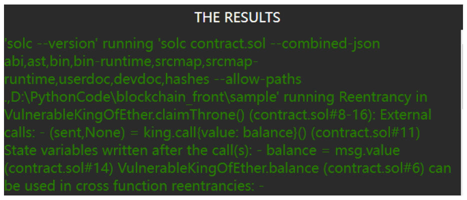
Figure 10: Operator Interface

3.Contract detection enforcement frame. We input a smart contract



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

contract VulnerableKingOfEther {
    address public king;
    uint256 public balance;

    function claimThrone() external payable {
        require(msg.value > balance, "Need to pay more to become the
king");

        (bool sent,) = king.call{value: balance}("");
        require(sent, "Failed to send Ether");  // if the current ki
fallback function reverts, it will prevent others from becoming the
king, causing a Denial of Service.

        balance = msg.value;
        king = msg.sender;
    }
}
```

Figure 11: Contract Detection

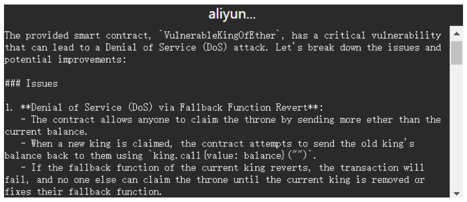


THE RESULTS

```
'solc --version' running 'solc contract.sol --combined-json
abi,ast,bin,bin-runtime,srcmap,srcmap-
runtime,userdoc,devdoc,hashes --allow-paths
.,D:\PythonCode\blockchain_front\sample' running Reentrancy in
VulnerableKingOfEther.claimThrone() (contract.sol#8-16): External
calls: - (sent,None) = king.call{value: balance}() (contract.sol#11)
State variables written after the call(s): - balance = msg.value
(contract.sol#14) VulnerableKingOfEther.balance (contract.sol#6) can
be used in cross function reentrancies: -
```

Figure 12: Result Output

4.Detection result output box
5.AI test result output box



aliyun...

```
The provided smart contract, `VulnerableKingOfEther`, has a critical vulnerability
that can lead to a Denial of Service (DoS) attack. Let's break down the issues and
potential improvements:

### Issues

1. **Denial of Service (DoS) via Fallback Function Revert**:
   - The contract allows anyone to claim the throne by sending more ether than the
current balance.
   - When a new king is claimed, the contract attempts to send the old king's
balance back to them using `king.call{value: balance}("")`.
   - If the fallback function of the current king reverts, the transaction will
fail, and no one else can claim the throne until the current king is removed or
fixes their fallback function.
```

Figure 13: AI Result Output

References

[1] Web3dao-cn. "Constant Product AMM — Solidity by Example — 0.8.10," [Online]. Available: https://web3dao-cn.github.io/solidity-example/defi/constant-product-amm/.

[2] Web3dao-cn. "Uniswap V2 Swap — Solidity by Example — 0.8.10," [Online]. Available: https://web3dao-cn.github.io/solidity-example/defi/uniswap-v2/.

[3] Web3dao-cn. "Uniswap V2 Add Remove Liquidity — Solidity by Example — 0.8.10," [Online]. Available: https://web3dao-cn.github.io/solidity-example/defi/uniswap-v2-add-remove-liquidity/.

[4] Jeiwan. "Programming DeFi: Uniswap V2. Part 1 - Going the distance," [Online]. Available: https://jeiwan.net/posts/programming-defi-uniswapv2-1/.

[5] "What is an Automated Market Maker? (Liquidity Pool Algorithm)," [Online]. Available: https://www.youtube.com/watch?v=1PbZMudPP5E.

[6] "What is Uniswap? (Animated) Decentralized Exchange + UNI Token," [Online]. Available: https://www.youtube.com/watch?v=DLu35sIqVTM.

[7] "04 Constant Product Automated Market Maker Algorithm_Bilibili_bilibili," [Online]. Available: https://www.bilibili.com/video/BV13L411S7he?spm_id_from=333.788.videopod.sections&vd_source=773757a71f9a56d209b90777c053acd3.

[8] "05 Constant Product Automated Market Maker Algorithm - Contract Case_Bilibili_bilibili," [Online]. Available: https://www.bilibili.com/video/BV1w24y157Jx?spm_id_from=333.788.videopod.sections&vd_source=773757a71f9a56d209b90777c053acd3.

[9] Runtimeverification. "verified-smart-contracts/uniswap/x-y-k.pdf at uniswap · runtimeverification/verified-smart-contracts," [Online]. Available: https://github.com/runtimeverification/verified-smart-contracts/blob/uniswap/uniswap/x-y-k.pdf.

[10] "Analysis of the Core Algorithm of Uniswap Automated Market Maker in Blockchain Mathematics — Deng Chain Community — Blockchain Technology Community," [Online]. Available: https://learnblockchain.cn/article/1494.

[11] "An Article to Understand the Development History of Decentralized Exchanges - Zhihu," [Online]. Available: https://zhuanlan.zhihu.com/p=366152927.

[12] "What is an Order Book and How Does it Work?," [Online]. Available: https://b2broker.com/zh-hans/library/what-is-an-order-book-and-how-does-it-work/.

[13] "How to Use Slither to Find Smart Contract Bugs," [Online]. Available: https://ethereum.org/zh/developers/tutorials/how-to-use-slither-to-find-smart-contract-bugs/.

[14] "Slither: A Static Analysis Tool for Blockchain Smart Contracts," [Online]. Available: https://blog.csdn.net/weixin_44217936/article/details/123240460?fromshare=blogdetail&sharetype=blogdetail&sharerId=123240460&sharerefer=PC&sharesource=weixin_52769806&sharefrom=from_link.

[15] "How to Add API Key to Environment Variables in Model Studio," [Online]. Available: https://b2broker.com/zh-hans/library/what-is-an-order-book-and-how-does-it-work/.

[16] "What is an Order Book and How Does it Work?," [Online]. Available: https://help.aliyun.com/zh/model-studio/developer-reference/configure-api-key-through-environment-variables?spm=0.0.0.i2#e4cd73d544i3r.