MeV Front-Running & Slippage Simulation Bot

Raul Ernesto Guillen Hernandez
Fernando Daniel González Batarsé
Aleksandr Zvonarev

Agenda

- 🎯 Project context & goals
- 🥪 System architecture
- 🚀 End-to-end user flow
- 👁️ Router tracking & mempool filter
- 📈 Swap decoding & on-chain market data
- 🛠️ Sandwich-attack workflow
- 💰 Slippage / profit maths
- ➡️ Results & next steps

“It's not whether you're right or wrong that's important, but how much money you make when you're right and how much you lose when you're wrong”

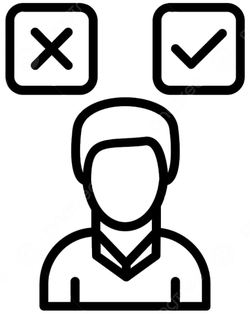
George Soros

MeV

Listen & Detect

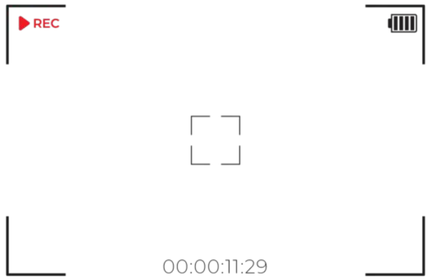
Subscribe to Sepolia’s pending-tx pool via QuickNode WebSocket and filter for Uniswap V2 swap calls

Simulate & Decide



Submit the front-run and back-run transactions around the victim swap to realize the predicted profit

Execute & Capture



For each detected swap, fetch on-chain reserves, estimate slippage, and calculate the optimal front-run size that maximizes profit after gas

System Architecture

Service Layer

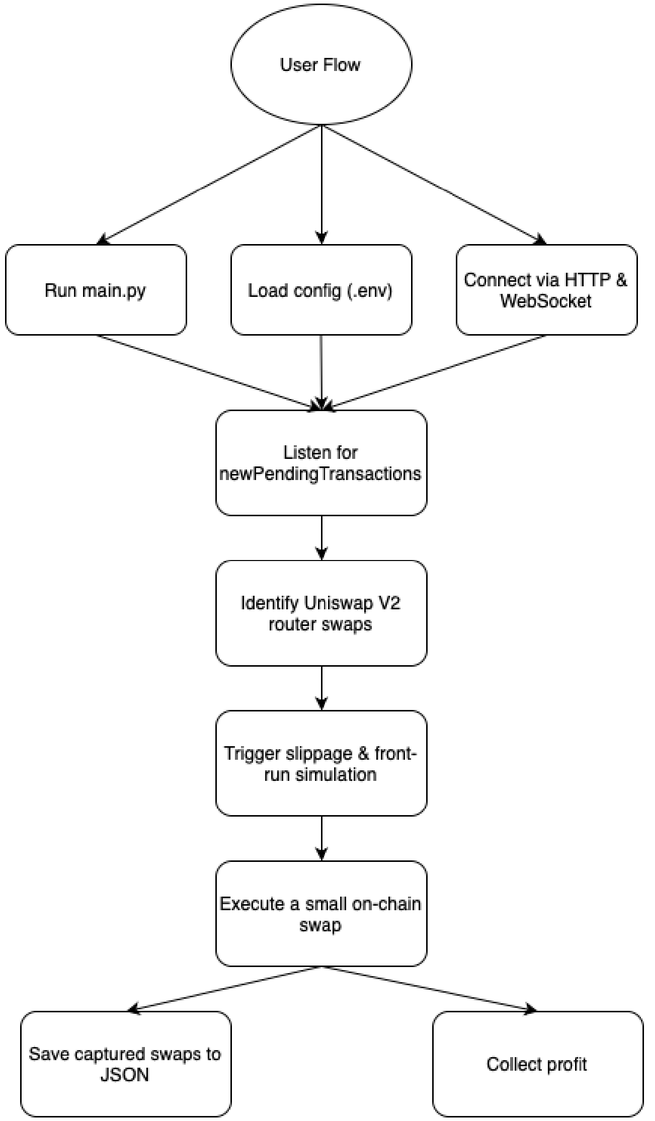
- HTTPProvider (QuickNode HTTPS) for signing & broadcasting transactions, fetching on-chain data (contract ABIs, reserves)
- WebsocketProvider (QuickNode WSS) for real-time pending-tx stream

Core Components

- Mempool Tracker (core/track_mempool.py) for tracking the Ethereum mempool for Uniswap router transactions
- Swap Execution (core/execute_swap.py) for building, signs, and broadcasts a test swap on the Uniswap router
- Calculating slippage (core/slippage.py) for computing price impact (slippage) using constant-product formulas and estimates optimal front-run size along with expected profit

User Flow

- Startup (main.py)
- Subscription (core/track_mempool.py)
- Filtering (core/track_mempool.py)
- Analysis (core/slippage.py)
- Execution (core/execute_swap.py)
- Results (main.py)



Router & Mempool

Router

Initialize web socket and http connection with QiuckNode. Load ABI & address for UniswapV2Router02
Call contract instance with web3.eth.contract(address, abi)
Your wallet
└──(Router pulls tokens in)—▶️ First Pair
└──swap▶️ Second Pair ⋯
└──swap▶️ Your “to” wallet

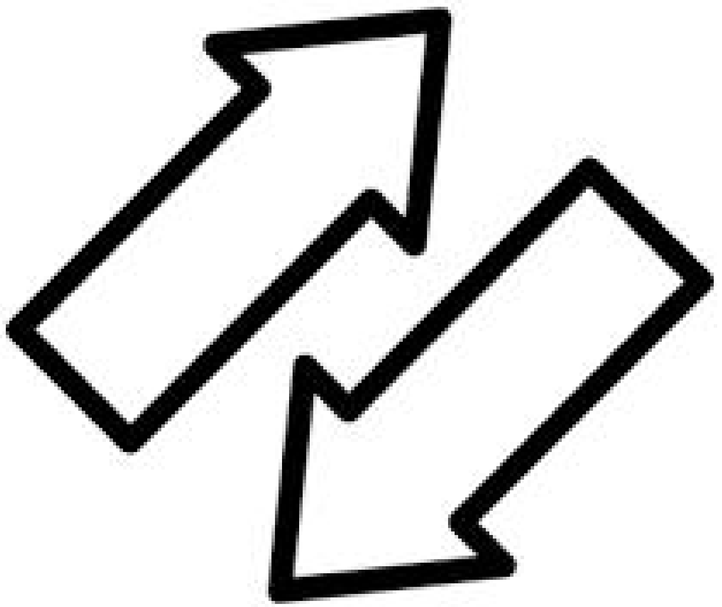
Mempool

Subscribe to “newPendingTransactions” via web3_wss.eth.subscribe
Filter transactions where “to” matches router address and input selector in known swap functions
Loop until max_swaps reached or timeout

Decoding Swap Transactions

Decode

Decode input data via router.contract.decode_function_input
Extract parameters: amountOutMin, path, recipient, deadline
Compute gas used, effectiveGasPrice, fee in ETH, and fee percentage



Retrieve

Call getReserves on the USDC-WETH pair contract to retrieve reserve_usdc and reserve_weth
Compute price_weth_in_usdc = (reserve_usdc/1e18)/(reserve_weth/1e18) and price_usdc_in_weth = 1/price_weth_in_usdc
Multiply by mainnet USDC price for USD market price

Sandwich

Victim Swap

Victim executes against the inflated price, increasing slippage

Profit & Risks

profit = (tokens bought × price difference) − gas/fees

Front-Run Buy

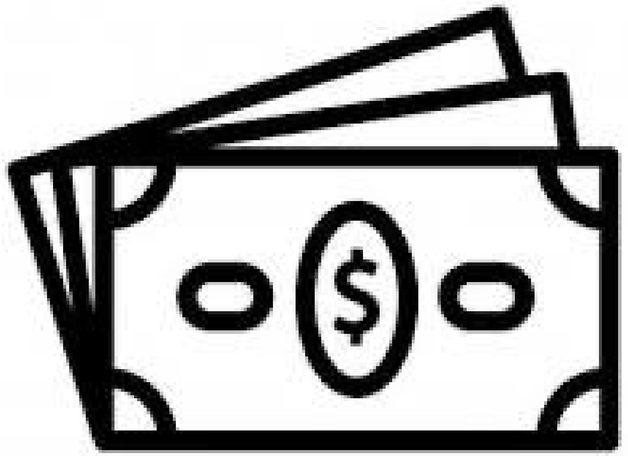
Submit a buy order with higher gas to execute before the victim and push price up

Back-Run Sell

Sell tokens at the victim-inflated price to capture the spread

Math

- R, T – pool reserves (ETH, token) before attacker
- γ - fee factor
- F – attacker front-run ETH
- V – victim ETH (known)
- G – gas cost of the two attacker legs (ETH)



Math

Front-run (attacker buys) ——— $\mathrm{x}_F = \frac{\gamma F T_0}{R_0 + \gamma F}, R_1 = R_0 + F, T_1 = T_0 - x_F$

Victim buy ——— $\mathrm{x}_V = \frac{\gamma V T_1}{R_1 + \gamma V}, slippage: \quad \frac{x_V}{V} \frac{T_0}{R_0} \geq 1 - \phi$

Back-run (attacker sells) ——— $\mathrm{R}_2 = R_1 + V, T_2 = T_1 - x_V, E_B = \frac{\gamma x_F T_2}{R_2 + \gamma x_F}$

Profit function ——— $\Pi(F) = E_B - F - G$

Demo

Thank you

Good trading is like a razor blade; it's easy
to cut yourself if you are not careful