ChatGPT

Building a Sniper Bot on Abstract Chain (Moonshot & AbstractSwap)

Abstract Ecosystem mapping highlighting Moonshot and other launchpads in Abstract's network [1] .

Overview

Abstract Chain is an Ethereum Layer-2 blockchain (built with zkSync's ZK-stack) optimized for consumer dApps [2] [3] . It uses ETH as the native currency and supports standard Ethereum tooling (EVM-compatible). **Moonshot** is Abstract's official meme token launchpad (developed by DexScreener) for fair token launches, while **AbstractSwap** refers to the chain's DEX (an implementation of Uniswap) where liquidity pools reside [4] . A sniper bot on Abstract should automatically detect new token launches or liquidity events and execute trades in the **very first block**, then optionally sell the tokens for profit. Below we compile the technical details – contract addresses/ABIs, RPC endpoints, and strategies – needed to build such a bot.

Abstract Chain Technical Basics

- **RPC Endpoints & Network Info:** Abstract provides public JSON-RPC endpoints for Mainnet and Testnet. For Mainnet, use `https://api.mainnet.abs.xyz` (WebSocket: `wss://api.mainnet.abs.xyz/ws`), chain ID `2741`. Testnet (an overlay on Sepolia) is at `https://api.testnet.abs.xyz` (WS: `wss://api.testnet.abs.xyz/ws`), chain ID `11124` [5] . The native token is ETH for gas fees [6] , so no custom currency wrapper is needed.

- **Node Infrastructure & Block Times:** Currently Abstract runs with a centralized sequencer (as is typical for new L2 networks). Block times are very fast – on the order of ~1 second per block under normal conditions (blocks are produced in quick succession as seen on the explorer) [7] [8]. This means **transactions confirm within 1–2 seconds** on-chain, though finality on Ethereum (ZK proof validity) comes later. The high block frequency requires your bot to react extremely quickly (ideally sub-second) to beat competition. Use the WebSocket RPC or run a node (if available) to get the lowest-latency feed of new blocks and pending transactions.

- **Account Abstraction:** All user accounts on Abstract are smart contract wallets under the hood, but externally they can act like normal EOAs due to native account abstraction [9]. This has minimal direct impact on your bot logic, except that gas management may differ slightly (e.g. **gas refunds** and paymaster fees are possible as shown on the explorer) [10] [11]. In practice, you can treat your bot's account as an EOA (using a standard private key) unless you intentionally use Abstract's AA features. Ensure your web3 library supports EIP-4337 if you plan to leverage sponsored transactions or paymasters, but otherwise the usual `Wallet` from ethers.js or web3.py works for signing and sending transactions.

- **Connecting and Funding:** You can add Abstract's network to MetaMask or web3 providers using the above RPC URL and chain ID. The block explorer is **Abscan** (`abscan.org`) which supports contract verification and a "Pending Transactions" view. To fund your bot's address on Abstract, you'll typically bridge ETH from Ethereum (Abstract has a bridge UI and also third-party bridges like Jumper/Relay [12]). Make sure your bot's address has enough ETH on Abstract for gas fees and initial snipe capital.

Moonshot Launchpad (Token Launches on Abstract)

Moonshot is the official launchpad for new memecoins on Abstract. It allows anyone to permissionlessly create and launch a token with a **fair, first-come first-served distribution**. Some key properties of Moonshot launches on Abstract:

- **Fixed Supply & Fair Launch:** Every Moonshot token has a fixed total supply of **1,000,000,000 (1 billion)** tokens [13]. There is no presale or insider allocation – tokens are only minted when public buyers purchase them via the Moonshot contract. The creator does not get a reserved supply (they must buy in like everyone else, though they can be the first to buy). This design prevents pre-launch dumps and rug-pulls by insiders. The token code is **fully audited** by Ackee Blockchain and cannot be altered by the creator, so it *cannot implement malicious mechanics like disabling sells* [14] (Moonshot explicitly guarantees that the launch contracts **"cannot be arbitrarily changed to prevent individuals from selling their tokens."** [14]).

- **Bonding Curve Pricing:** Moonshot uses a **bonding curve AMM** built into each token's smart contract for the initial distribution [15]. The price starts very low and increases exponentially as more tokens are bought, following a constant product formula. In fact, the curve is designed such that at ~80% of tokens sold, the price is ~16.56× the initial price [4]. This mechanism rewards early buyers with lower prices. The curve formula uses "virtual" reserves of token and ETH internally [16] [17], ensuring smooth price movement rather than a sudden listing at market price.

- **Trading & Liquidity Migration:** All buys and sells before the cap are done against the token's contract (no external liquidity pool needed initially). Once **~80% of the supply is sold**, Moonshot **automatically halts the curve and migrates liquidity to AbstractSwap**. Specifically, at the migration point (when Fully Diluted Market Cap ≈ 25 ETH is reached), the contract creates a Uniswap v2 pool on AbstractSwap containing the **remaining ~20% of tokens** and the accumulated ETH collateral (minus a small fee) [4] [18]. Approximately **4.9 ETH** of ETH liquidity is paired with the unsold tokens in the new pool, and any excess tokens (beyond those moved to the pool) are **burned** to tighten supply [19]. After this migration, trading continues on the DEX like a normal ERC-20 token market.

- **Platform Fees and Limits:** Moonshot charges a **small fee** for its service. On Solana it's 1% of each trade [20], and on Abstract the model is similar (with fees likely taken in ETH). The contract also imposes **minimum trade amounts** to deter spam: e.g. on Solana, users must buy at least 0.005 SOL and sell at least 0.001 SOL [20]. We can expect Abstract's minimums to be analogously low (likely on the order of a few thousand gwei of ETH). Additionally, at migration a fixed **migration fee of 0.15 ETH** is deducted from the collateral to fund the pool creation and Moonshot's service [21]. There is **no fee to create a token** aside from a minimal gas cost (and previously a ~0.02 SOL fee on Solana) [22]. These fees and thresholds mean your bot should account for ~1–2% slippage from fees and not attempt trades below the minimum size.

- **Moonshot Contract Addresses & ABIs:** On Abstract, the Moonshot functionality is implemented via a **single launch contract** (often called the Moonshot Factory or similar) which spawns each token's contract. The main Moonshot contract address on Abstract mainnet is `0x99Bb83AE9bb0C0A6be865CaCF67760947f91Cb70` [23]. When a new token is launched, the user's wallet calls this contract (e.g. `createToken(name, symbol, …)`) and it internally deploys a new token contract. All trading (buy/sell) for that token then occurs by sending transactions to the *token's own contract* (each token is an ERC-20 with added buy/sell functions). The ABI for Moonshot token contracts isn't officially published in docs, but it's standardized due to the audit. You have two practical options for obtaining ABIs: (1) Use the **Moonshot SDK** (see next section) which abstracts the ABI calls for you, or (2) fetch the verified source once someone verifies a token contract on Abscan (many may not be verified immediately). The audit guarantees the same code for all, so an ABI from any launched token or from the Moonshot SDK should work for others. At minimum, the token contract will have methods like `buy`/`sell` or a unified swap function and possibly read functions for price, as implied by the SDK. The Moonshot main contract likely emits an event on token creation (for example, a `TokenCreated(address tokenAddr, string symbol, …)` event) – your bot can listen to this to catch new launches in real time.

- **Using the Moonshot SDK (Node.js):** DexScreener provides an official NPM package `@wen-moon-ser/moonshot-sdk-evm` to simplify bot development [24]. This SDK supports both **Abstract and Base** networks [25]. It handles price calculation on the bonding curve and transaction building for buys/sells. With the SDK you can:

- **Compute Prices and Amounts:** e.g. get how many tokens you'd receive for X ETH or vice-versa at current curve position [26].
- **Prepare Buy/Sell Transactions:** It constructs the transaction data for you, including slippage control and distinguishing exact-in vs exact-out trades [27] [28].

- **Minting (Token Launch):** The SDK can even initiate launching a token (likely not needed for sniping, but available).

Using the SDK is straightforward in Node.js. For example, to buy tokens:

```
const provider = new JsonRpcProvider(ABS_RPC_URL);
const signer = new Wallet(PRIVATE_KEY, provider);
const moonshot = new Moonshot({ signer, env: Environment.MAINNET, network:
Network.ABSTRACT });
const token = await Token.create({ moonshot, provider, tokenAddress });
const quote = await token.getTokenAmountByCollateral({ collateralAmount:
ethers.parseEther("0.5"), tradeDirection: 'BUY' });
const txReq = await token.prepareTx({ tradeDirection: 'BUY', collateralAmount:
ethers.parseEther("0.5"), tokenAmount: quote, slippageBps: 500, fixedSide:
FixedSide.IN });
const txResponse = await signer.sendTransaction(txReq);
```

This would generate and send a buy transaction of 0.5 ETH (with 5% slippage tolerance) for the given token. The SDK internally knows how to call the token's contract correctly. **Using the SDK is highly recommended** for Node.js bots since it encapsulates the ABI and bonding curve math (avoiding manual reverse-engineering). It also includes a symmetrical `prepareTx` for selling [29] [30]. Ensure you initialize the SDK with `Network.ABSTRACT` and the appropriate environment (Testnet vs Mainnet) [31].

- **Manual Contract Interaction (Python or Low-level):** If you prefer Python (which doesn't have the official SDK), you can still interact with Moonshot contracts using Web3.py or any EVM library. You'll need the ABI definitions. Typically, a Moonshot token contract might have functions like `buy(uint256 minTokensOut) payable` and `sell(uint256 tokenAmount, uint256 minEthOut)` or similar. These functions likely handle the internal price curve logic. For example, to buy, you might call the contract function and send ETH in the transaction; to sell, you approve the token to itself or to the Moonshot contract and then call sell. Without the SDK, one approach is to watch how the **Moonshot SDK's transactions** look (you can log the `txReq.data` or observe a mempool tx from the SDK) to deduce function signatures. Alternatively, monitor the **event logs**: Moonshot tokens likely emit a `Transfer` event for each buy/sell (from/to the contract address) which can confirm a trade's success and amount. **Note:** Using the SDK's logic or a verified contract's ABI is much easier than fully manual encoding. If you go manual, thoroughly test on Abstract's testnet.

AbstractSwap DEX (Uniswap on Abstract)

AbstractSwap is the DEX where tokens trade freely after launch (and where liquidity-add sniping happens). AbstractSwap is essentially **Uniswap V2** (and V3) deployed on Abstract. The official docs list the deployed Uniswap contracts on Abstract [32]:

- **Uniswap V2 Factory (Abstract mainnet):** `0x566d7510dEE58360a64C9827257cF6D0Dc43985E` [33]
- **Uniswap V2 Router02:** `0xad1eCa41E6F772bE3cb5A48A6141f9bcc1AF9F7c` [33]

(The above addresses are for mainnet; testnet addresses are the same in docs, suggesting they deployed identical contracts on testnet, but you can use the testnet explorer for confirmation.) These are standard Uniswap v2 contracts, so you can use the usual ABI for Factory (to watch `PairCreated` events) and Router (to perform swaps or add liquidity). The **init code hash** for the factory is provided as well [34] if needed for computing pair addresses off-chain. Uniswap V3 is also deployed (factory at `0xA1160e73B63F322ae88cC2d8E700833e71D0b2a1`), but **most meme tokens will use the V2 pools** for simplicity.

Liquidity Adds (Pair Creations): When a token is launched via Moonshot, its initial trading is on the Moonshot curve. But tokens can also be launched directly by adding liquidity on AbstractSwap. In either case, the moment liquidity is added (or migrated) to a Uniswap pool is a prime sniping opportunity. Your bot should monitor the Uniswap **Factory's** `PairCreated(tokenA, tokenB, pairAddress)` **event**. Specifically, filter for events where one of the tokens is WETH/ETH (since new pools against ETH are the usual case for new tokens). Abstract's WETH address (if any) is `0x3439153EB7AF838Ad19d56E1571FBD09333C2809` (though since ETH is native, "ETH" itself might be used in Router calls) [35]. The moment a new pair with ETH and some token is created, you can fetch that pair address and then watch for the first liquidity addition. Often, developers will create the pair and add liquidity in back-to-back transactions (or even within one transaction using the router). A robust bot should catch the **AddLiquidity** event or the mempool transaction and attempt to **buy immediately in the same first block** if possible.

Swapping via Router vs Direct: For automated buys on AbstractSwap, using Uniswap's Router is straightforward – e.g. calling `swapExactETHForTokens` with your parameters. Just ensure to set a high enough gas price and slippage. Given Abstract's low base fees, you might also want to set a high priority fee to outbid others (if the sequencer sorts by fee). Alternatively, a micro-optimized bot could call the pair contract's `swap` function directly (skipping the Router's extra transfer steps) to save a bit of time/gas, but this requires computing the exact `amountOut` and providing the reserves. In practice, **Router calls are fine** for most cases, especially if you are first in line after liquidity is added. Use the Factory's event to get the pair address, then the pair's `getReserves()` to see initial liquidity, and then call Router's swap. *Note:* Right after migration from Moonshot, the pool will have known reserves (~4.9 ETH and ~200M tokens if 80% sold) [4] [18], so a sniper might try to buy heavily immediately if expecting continued demand.

Monitoring Pending TX vs Events: To catch liquidity adds on AbstractSwap optimally, you should use a combination of **mempool monitoring** and **on-chain events**: - Subscribe to the **Uniswap Factory's events** (via WebSocket `eth_subscribe` to logs or using the explorer's API) to detect new pairs. This is reliable but triggers **after** the pair creation transaction is mined, meaning your buy might go into the next block. - For a potential **same-block snipe**, also listen to the **pending transactions** on the mempool. By decoding pending tx input data, you can spot calls to the Uniswap Router or Factory. For example, a pending `createPair(address,address)` call to the factory, or a `addLiquidityETH` call to the router (check the method ID and params). If you see a dev adding liquidity, your bot can prepare a swap tx and broadcast it *with a higher gas fee* hoping to be mined in the same block (front-running the dev's own buy or others). Keep in mind on Abstract the single sequencer has control of ordering; there isn't a public miner network, so standard "priority fee" rules may or may not guarantee placement. Nonetheless, sending your buy with a generous gas tip *as soon as you detect liquidity* is the best strategy to get included ASAP. Abstract likely does not yet implement sophisticated MEV auctions, so simple high gas may suffice.

Bot Implementation Strategy

Combining the above, here's how to build the sniper bot step-by-step:

1. **Connect to Abstract RPC** – Use a Web3 provider (ethers.js or Web3.py) pointing to `api.mainnet.abs.xyz`. Enable WebSocket if possible for subscriptions. Confirm your bot's address has ETH on Abstract and the private key is loaded in your signer.

2. **Monitor Moonshot Launches:** Subscribe to the Moonshot contract for new token launch events. If Moonshot emits a `TokenCreated` event (with the new token's address and perhaps metadata), your bot can immediately begin targeting it. Upon catching a new token:

3. Optionally, do some **whitelist filtering**: e.g. avoid obvious scam tokens or ones not of interest (since anyone can launch tokens, not all are worth sniping). Moonshot tokens have no custom code (audited template), so the main risk is just whether the token will attract buyers. Many snipers will simply buy a small amount of every new launch and then decide to sell or hold quickly based on price action.
4. Use the Moonshot SDK or manual calls to **execute a buy** on the token's contract within the first block of its existence. For speed, you might pre-fund and pre-initialize a `Token` instance (in SDK) if possible. Realize that the *very first buy sets the initial price* (from effectively zero supply); being first yields the cheapest price. However, being first also incurs more slippage if you buy too much at once on the curve. A good strategy is to buy a moderate amount (not pushing the curve too far), then possibly scale in more on subsequent blocks if hype grows.

5. **First-block timing:** Because Abstract blocks are ~1s, the time between the token creation event and the next block is tiny. Your bot likely needs to be running on a low-latency server and respond **within milliseconds**. Using the SDK in-memory is faster than querying chain for reserves. The SDK's `prepareTx` already calculates the required output, so it's ideal.

6. **Monitor AbstractSwap Liquidity:** In parallel, subscribe to Uniswap **PairCreated** events and watch pending tx for liquidity adds. When a new pair with ETH is created:

7. If the token address matches one that was a Moonshot launch, you know this is the **migration event**. At that point, the Moonshot contract has stopped selling, and trading moves to the DEX. Your bot could attempt to snipe here as well – though note that by migration, price is already 16× initial. Often, the price can still pump on the DEX if demand continues. You might decide to **buy right after migration** hoping for further rally, or **sell your holdings into the newly created pool** if you bought on the curve and want to lock profits. Migration will be a known single transaction (from the Moonshot contract) creating the pair and adding liquidity; your bot can listen for that specific tx (likely from `0x99Bb83AE...` address) to trigger its action.
8. If the new pair is a token not from Moonshot (i.e. a dev who manually deploys a token and adds liquidity), you have less info about the token's code. **Important:** Check the token's contract for known honeypot signs *before buying*. For safety, your bot can automatically attempt a **read of the token's source or simulate a sell**:
    - If the contract is verified on Abscan, fetch the source or ABI and scan for suspicious functions (like `_disableSell`, `onlyWhitelisted`, `getTax()` etc.). You can also call `token.balanceOf(YOUR_ADDRESS)` after a tiny buy to ensure transfers out are possible.

    - If not verified, consider doing a **small test buy** and then immediately try a **small sell** (or swap back) to ensure the token can be sold. Doing this with minimal amount and checking that you get ETH back is the safest way to avoid honeypots. Your bot should ideally automate this simulation (maybe on a forked node or by analyzing the bytecode for known patterns). This is a common sniper practice on ETH mainnet that applies here too.

9. If the token passes the anti-honeypot check, proceed to execute a buy via the Router. Construct `swapExactETHForTokens` with a high gas price. Because Abstract's current traffic is lower than Ethereum mainnet, you might get away with moderate gas, but in a competitive scenario, don't skimp on gas tip.

10. **Selling Logic:** The bot should also implement an **auto-sell** feature:

11. For Moonshot trades *before migration*, selling means calling the token's contract sell function to redeem ETH. You can use the SDK (`token.prepareTx({... tradeDirection: 'SELL' ...})`) or manually call the sell method. Ensure you approve the token to itself or follow the contract's procedure (the SDK likely handles any needed approval internally or uses a single contract design that doesn't require an ERC20 approve). Monitor the price – Moonshot's SDK can get the current price or you can compute from the formula. You might set a target (e.g. sell when 2× or 3× your buy price) and let the bot call sell if reached. But be cautious: if the token is nearing 80% sold, the liquidity will migrate. Some bots prefer to **wait for migration** then sell on the Uniswap pool, which might have more liquidity and less price impact for selling large amounts.
12. For tokens on AbstractSwap (after migration or any token launched directly on the DEX), selling is just a Uniswap swap. Use `swapExactTokensForETH` via the Router. You'd need to approve the Router for the token beforehand. You can also target specific conditions: e.g. a stop-loss if price dumps, or a profit target. It's wise to place a **take-profit order** soon after buying (or at least have the logic continuously check price). Abstract's fast blocks mean prices can pump or dump within seconds.

13. **MEV and frontrunning:** Be aware that once your sell (or buy) is in the mempool, others might try to sandwich or front-run you. On Ethereum, this is common; on Abstract, it's less clear how active independent MEV bots are at this stage. The Abstract sequencer could theoretically reorder transactions, but there's no public evidence of aggressive MEV extraction yet. Still, you should simulate the expected output of your trade with a certain slippage and set parameters conservatively. For example, if you set 1% slippage on a low-liquidity token, a bot could try to buy just before your sell (pushing price down) and profit from your slippage allowance. To mitigate, use small slippage percentages and maybe split large sells into chunks.

14. **Frontrunning Opportunities:** In addition to sniping launches, your bot can also implement **frontrunning logic** to capitalize on others' transactions:

15. On Moonshot: If you detect a huge buy coming (perhaps from on-chain activity or a mempool transaction with a large ETH amount to a Moonshot token contract), you could attempt to send your buy with a higher gas to get in the block before them. If successful, their large buy will pump the price *after* your purchase, letting you instantly be in profit. This is tricky on a sequencer model, but you can attempt it. Use mempool subscribe to watch for transactions to `0x99Bb83...` or to specific token addresses and try to jump ahead.

16. On AbstractSwap: Traditional **sandwich attacks** are possible. If someone places a large swap via the Router with low slippage, your bot could buy right before and sell right after their transaction in the same block, pocketing the difference. Implementing this requires parsing pending transactions to the Router, calculating the price impact, and ensuring your bot's transactions bracket the victim's. Given Abstract's current single-block ordering controlled by the sequencer, this might not always work unless the sequencer orders purely by gas price (which you can exploit by paying higher). Proceed with caution and test thoroughly; also be mindful that sandwiches harm regular users, and Abstract might eventually introduce protections or discourage this.

Mempool Monitoring Techniques

Access to the mempool (pending transaction pool) is crucial for timely sniping. Here are some techniques/ tools to use:

- **WebSocket `eth_subscribe`:** Using the WebSocket RPC, subscribe to `"newPendingTransactions"` or `"newHeads"` and `"logs"`. For example, you can subscribe to pending tx and then call `eth_getTransactionByHash` for each to inspect its `to` address and input. Filter for relevant transactions:
- Moonshot: `to == 0x99Bb83AE9bb0C0A6be865CaCF67760947f91Cb70` (the main launch contract) or `to == <token address>` for tokens of interest. When monitoring all tokens, you may simply catch the creation event instead of every pending trade (since many small buys happen).
- Uniswap: `to == 0xad1eCa41E6F772bE3cb5A48A6141f9bcc1AF9F7c` (Router) or `to == 0x566d7510dEE58360a64C9827257cF6D0Dc43985E` (Factory). Decode the first 4 bytes of `input` to identify function: `0xc9c65396` is `createPair` on Uniswap V2 Factory, for instance. Or use an ABI to parse pending tx data into function calls.
- **Running a Full Node:** If Abstract's node software is available (since it's based on zkSync's stack, it might be possible to run a local instance), running your own node could give the fastest feed of pending transactions and the ability to use `txpool` RPC methods. If a public/semi-public node provider (like Alchemy or Infura) adds Abstract, they might also support `alchemy_pendingTransactions` or similar enhanced websockets [36]. As of now, you'll likely rely on the official RPC which may have rate limits; consider using multiple endpoints or contacting the Abstract team for high-throughput access if your bot is very active.
- **DEX Screener API / Alerts:** Since Dexscreener is intimately involved (Moonshot is their product), they have excellent data on new tokens. Dexscreener's API (docs.dexscreener.com) might allow fetching **recent launches or trades** on Abstract. In particular, their "New pairs" or "Trending" endpoints could be used as a backup to verify you didn't miss anything. They also show a "Finalized" status when a Moonshot token migrates to Uniswap. However, **do not rely solely on external APIs for real-time sniping** – they might lag by a few seconds. Use them for monitoring and analytics (e.g. to see volume, holders count via their UI), but your bot's core triggers should be chain-driven.

Risks, Protections, and Best Practices

Building a sniper bot comes with both technical and financial risks. Be mindful of the following:

- **Honeypots & Malicious Tokens:** Thanks to Moonshot's audited contracts, tokens launched through Moonshot are free of honeypot code (they can't lock your liquidity or block selling) [14]. But if you snipe tokens launched outside Moonshot, **assume worst-case** until proven otherwise. Always test sell as described or inspect the code if possible. Even on Moonshot tokens, the economic outcome can be negative if you overpay on the curve and no one else buys after you – so treat low volume launches carefully.

- **Anti-Bot Measures:** Abstract's **portal UI** tries to filter out bots (e.g. requiring social account linking for XP rewards, etc., to discourage sybils) [37] , but on the *protocol level*, there are no explicit anti-sniping measures beyond what the token creators add. The Moonshot T&C asks users not to use bots on their interface [38] , but as a bot developer interacting directly with contracts, this is not technically enforced – just be aware you're stepping outside "intended" use. In the future, Abstract or DexScreener might implement bot mitigation (for example, they could refuse to list a token's info on the portal if only botted addresses hold it, or implement a delay in showing price), but they cannot stop on-chain transactions. Some token creators might attempt anti-bot tricks like setting a **max transaction size** in the token for the first few blocks or an initial high tax. Moonshot tokens *do not have such mechanics by design*. For non-Moonshot tokens, you have to evaluate each case.

- **MEV and Sequencer Behavior:** Because a centralized sequencer currently produces Abstract's blocks, it could theoretically **reorder or insert its own transactions** to capture MEV (Miner Extractable Value). We have no indication that the Abstract sequencer is doing so maliciously, but it's a risk. This means even if your bot reacts fastest, a sequencer or high-priority actor could still beat you to a trade. Over time, if Abstract decentralizes sequencing or integrates fair ordering (e.g. via MEV auctions or time-locks), the landscape may change. Keep updated on Abstract's dev news for any MEV mitigation features. Until then, operate under the assumption that high gas price and rapid submission give you the best chance, but nothing is guaranteed.

- **Gas and Transaction Confirmation:** Always monitor that your transactions get mined. If the network is congested or if you set too low a gas price, you might miss the opportunity entirely or, worse, end up buying after a price spike (and then holding a loss). Abstract's gas fees are extremely cheap (fractions of a cent) so you can afford to overpay gas to prioritize your tx – do it. Use the explorer's "Pending Tx" view to see if your tx is stuck. Also make use of the fact Abstract has **short block time** – if something fails or doesn't confirm in one block, you can quickly adjust and resend in the next block or two. The bot should have logic to cancel or replace transactions (using higher nonce, since native AA might complicate direct cancellation via replace-by-fee – you might need to send a "dummy" tx with same nonce and higher fee to override if using an EOA).

- **Rapid Price Movements & Slippage:** Sniper bots often need to set high slippage tolerance because prices can move drastically from block to block (especially on Moonshot's curve or right after pool creation). However, high slippage invites sandwiches and can result in awful execution if you mis-time. Balance slippage – for example, maybe allow 10-20% slippage on a very hyped launch (to ensure you don't fail due to others moving price), but generally try to keep it as tight as possible. The Moonshot SDK expects slippage in basis points (e.g. 1000 = 10%) [39] . On Uniswap trades, you specify minOut; set it based on your risk tolerance.

- **Concurrency and Multithreading:** Because events on Abstract happen fast, design your bot to handle multiple triggers simultaneously. It's possible two tokens launch minutes apart, or a token migrates to Uniswap while you're watching another. Use async processing or separate threads for Moonshot and Uniswap monitoring. Also, maintain a **state** of what you've bought and what your exit strategy is for each. It's easy to lose track if many snipes trigger – you don't want to forget to sell something or double-buy by mistake. Logging and perhaps a simple web dashboard for your bot can be helpful for manual overrides.

- **Testing:** Test on Abstract **Testnet** (Sepolia testnet) first. The same Moonshot contracts exist on testnet (they likely deployed a version for Sepolia – possibly the addresses given in docs under "Testnet"). You can try launching a dummy token on testnet via Moonshot and have your bot snipe it to simulate the whole flow. Also test Uniswap V2 sniping on testnet by creating a pair and seeing if your bot catches it. This will help fine-tune your timing without risking real ETH.

- **Future Developments:** Keep an eye on Abstract's updates. As the chain matures, they might introduce **official APIs or SDKs** (the Abstract team or community might release Python libraries, etc.), and new DeFi protocols will appear. Notably, **Buzz.fun** is a competing memecoin launchpad on Abstract [1] – it might have different rules or contract addresses. If your goal is comprehensive sniping, you may want to integrate support for Buzz.fun launches too (similar concept to Moonshot). Check if Buzz.fun provides an API or if their contracts are known.

In summary, building a sniper bot for Abstract involves combining **Moonshot's specialized launch contracts** with **traditional Uniswap v2 trading logic**. Leverage the official docs and SDKs where available: the Moonshot EVM SDK is a powerful tool to interact with the bonding curve [40], and Abstract's Ethereum-compatibility means all your usual Ethereum bot techniques apply. By monitoring Moonshot events and Uniswap pool creations in real-time, using fast connections, and implementing safe trading practices, you can achieve first-block snipes on token launches and liquidity additions. Always remember to manage risks – even a perfect technical execution can result in losses if a token's price tanks or if you get caught by a trap. Happy hunting on Abstract, and stay updated as the ecosystem evolves!

Sources: The information above is drawn from Abstract's official documentation (RPC endpoints, contract addresses, etc.) [5] [41], Moonshot developer docs and Terms [13] [4], and third-party analyses of Abstract's ecosystem [1] [42]. These resources provide further technical details and should be consulted directly for deeper understanding or updates.

---

[1] [2] [9] [12] Insight : All you need to know about Abstract airdrop and XP farming | OAK Research
https://oakresearch.io/en/analyses/innovations/insight-all-you-need-to-know-about-abstract-airdrop-xp

[3] What Is Abstract? A Chain From The Creators Of Pudgy Penguins | CoinGecko
https://www.coingecko.com/learn/what-is-abstract-chain-crypto

[4] [15] [16] [17] [18] [21] Bonding curve - EVM | Moonshot
https://docs.moonshot.cc/developers/bonding-curve-evm

[5] [6] Connect to Abstract - Abstract
https://docs.abs.xyz/connect-to-abstract

[7] [8] Abstract (ETH) Blockchain Explorer
https://abscan.org/

[10] [11] [23] ETH Transaction Hash: 0x8b18df7a14... | Abstract Block Explorer
https://abscan.org/tx/0x8b18df7a14a4abcba96a47bb8595ddf84ec62eae1f7996af2ff4b0f0d7c5ee3f

[13] [14] [38] Terms and Conditions | Moonshot
https://docs.moonshot.cc/terms-and-conditions

[19] [20] [22] [42] Moonshot vs. Pump.fun: Is the New Rival as Secure as It Claims? — TradingView News
https://www.tradingview.com/news/cryptonews:af3a299b2094b:0-moonshot-vs-pump-fun-is-the-new-rival-as-secure-as-it-claims/

[24] [26] [27] [28] [39] GitHub - gomoonit/moonshot-sdk-evm: Moonshot SDK for Base EVM chain
https://github.com/gomoonit/moonshot-sdk-evm

[25] [29] [30] [31] [40] Bot SDK - EVM | Moonshot
https://docs.moonshot.cc/developers/bot-sdk-evm

[32] [33] [34] [35] [41] Deployed Contracts - Abstract
https://docs.abs.xyz/tooling/deployed-contracts

[36] alchemy_pendingTransactions | Alchemy Docs
https://alchemy.com/docs/reference/alchemy-pendingtransactions

[37] Portal - Abstract
https://docs.abs.xyz/portal/overview