Utilizing NotArb's onchain program with Arb-Assist

This document will explain how to utilize Notarb's onchain program alongside running arb-assist to automatically find and execute on arbitrage opportunities utilizing arb-assist's advanced filtering system and the speed of notarb's onchain execution to create profitable strategies.

Requirements

In order to utilize this strategy, you'll be required to have a few things first:

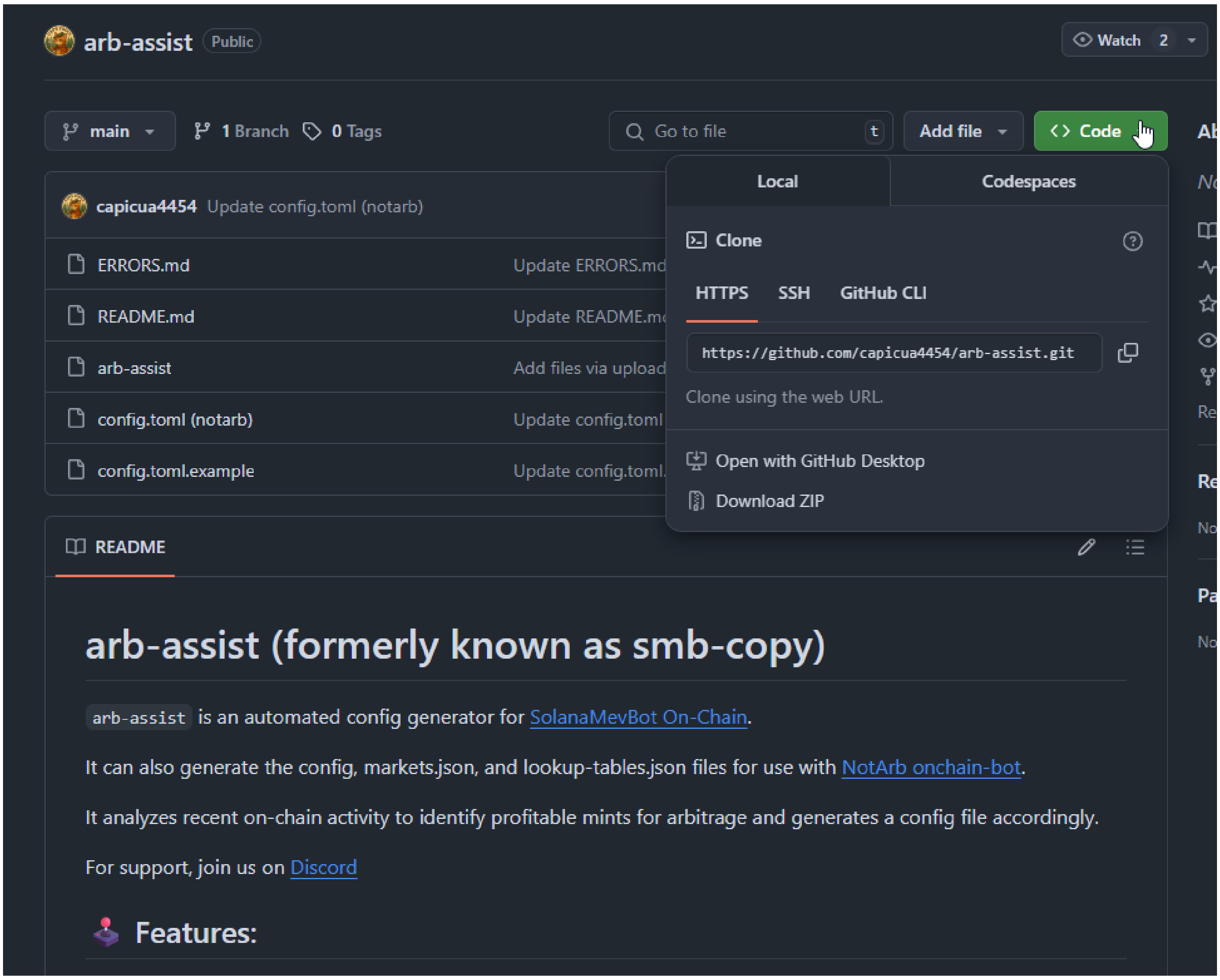
- RPC with grpc capabilities (If you need access to one you can contact Capicua (*tag: capicua4454*) on discord)
- A server to run the programs (baremetal or VPS will work)
- Arb-assist access
- Notarb

We'll go over installing and setting up Arb-assist and Notarb so they work in tandem but we won't dive deep into where to find RPC's that have grpc capabilities, you'll need to rent one from Capicua or find another source somewhere.

Initial setup

You'll begin installing Notarb by going to the releases page located on the programs github and downloading the latest release, if you're installing it onto a baremetal server you'll need to use wget or SFTP to put the files onto the server and if you're using a VPS you'll need to go to the page and download / extract the files to the location you want to run them.

You'll then need to go to the github page for Arb-Assist and install that aswell, It currently doesn't have releases setup similar to Notarb's page so you'll simply install it by going to Code -> Download Zip.



capicua4454 Update config.toml (notarb)

ERRORS.md	Update ERRORS.md
README.md	Update README.md
arb-assist	Add files via upload
config.toml (notarb)	Update config.toml
config.toml.example	Update config.toml.

```
https://github.com/capicua4454/arb-assist.git
```

Clone using the web URL.

arb-assist (formerly known as smb-copy)

`arb-assist` is an automated config generator for SolanaMevBot On-Chain.

It can also generate the config, markets.json, and lookup-tables.json files for use with NotArb onchain-bot.

It analyzes recent on-chain activity to identify profitable mints for arbitrage and generates a config file accordingly.

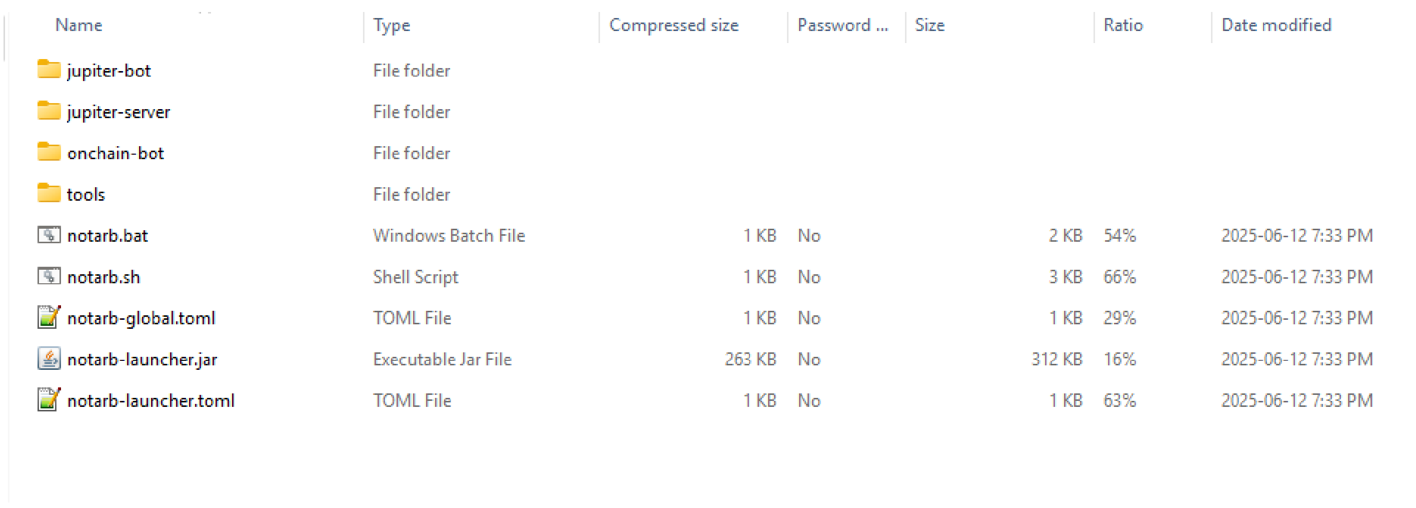
For support, join us on Discord

Features:

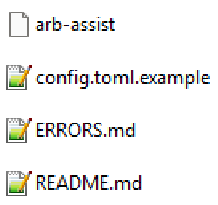
You'll want to also put the files from this github onto your server somewhere accessible, we'll be utilizing the files from it to assist Notarb in finding pools/opportunities.

At this point you should have all of the following files accessible to you:

Notarb files:



Name	Type	Compressed size	Password ...	Size	Ratio	Date modified
jupiter-bot	File folder					
jupiter-server	File folder					
onchain-bot	File folder					
tools	File folder					
notarb.bat	Windows Batch File	1 KB	No	2 KB	54%	2025-06-12 7:33 PM
notarb.sh	Shell Script	1 KB	No	3 KB	66%	2025-06-12 7:33 PM
notarb-global.toml	TOML File	1 KB	No	1 KB	29%	2025-06-12 7:33 PM
notarb-launcher.jar	Executable Jar File	263 KB	No	312 KB	16%	2025-06-12 7:33 PM
notarb-launcher.toml	TOML File	1 KB	No	1 KB	63%	2025-06-12 7:33 PM



arb-assist
config.toml.example
ERRORS.md
README.md

Arb-assist files:

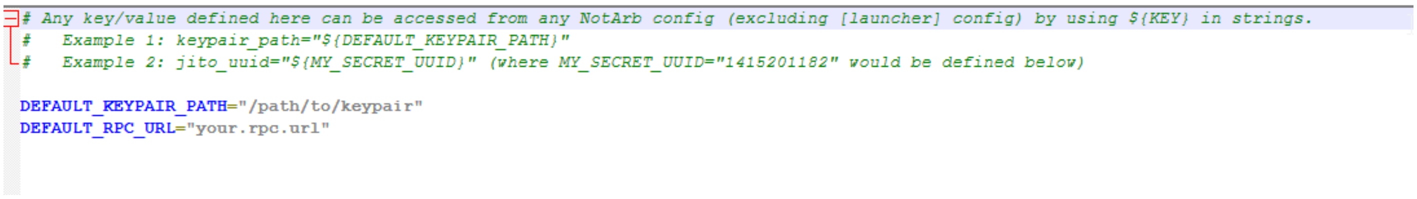
You'll now want to drag your license file you received from Capicua into the file containing the Arb-assist files, this will allow you access to utilize the Arb-assist program.

Editing your files

Now that you've obtained all the required files, it's time to set them up with our parameters so we can begin refining our strategy and making profits.

We'll begin by setting up Notarb with all the required information to get it up and running properly.

So begin by opening up notarb-global and adding the required information displayed:



```
# Any key/value defined here can be accessed from any NotArb config (excluding [launcher] config) by using ${KEY} in strings.
#   Example 1: keypair_path="${DEFAULT_KEYPAIR_PATH}"
#   Example 2: jito_uuid="${MY_SECRET_UUID}" (where MY_SECRET_UUID="1415201182" would be defined below)

DEFAULT_KEYPAIR_PATH="/path/to/keypair"
DEFAULT_RPC_URL="your.rpc.url"
```

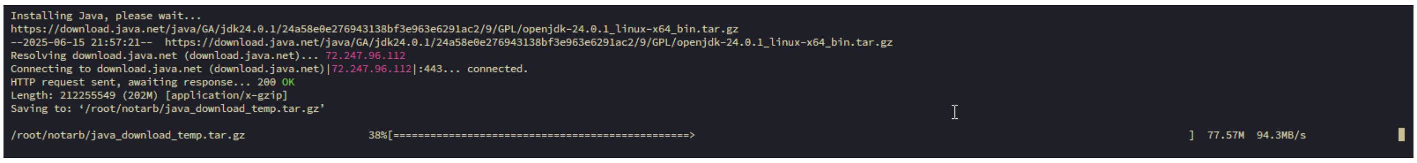
The fields are pretty self explanatory, you'll need to add in the URL to your RPC as well as a directory to your private key. Which will be a file that obtains your private key in a .txt format. **Ensure you don't share this key with anybody.**

Once that's been installed - we'll need to run the program to install all the dependences required along Java.

You'll run it using the following command in your terminal:

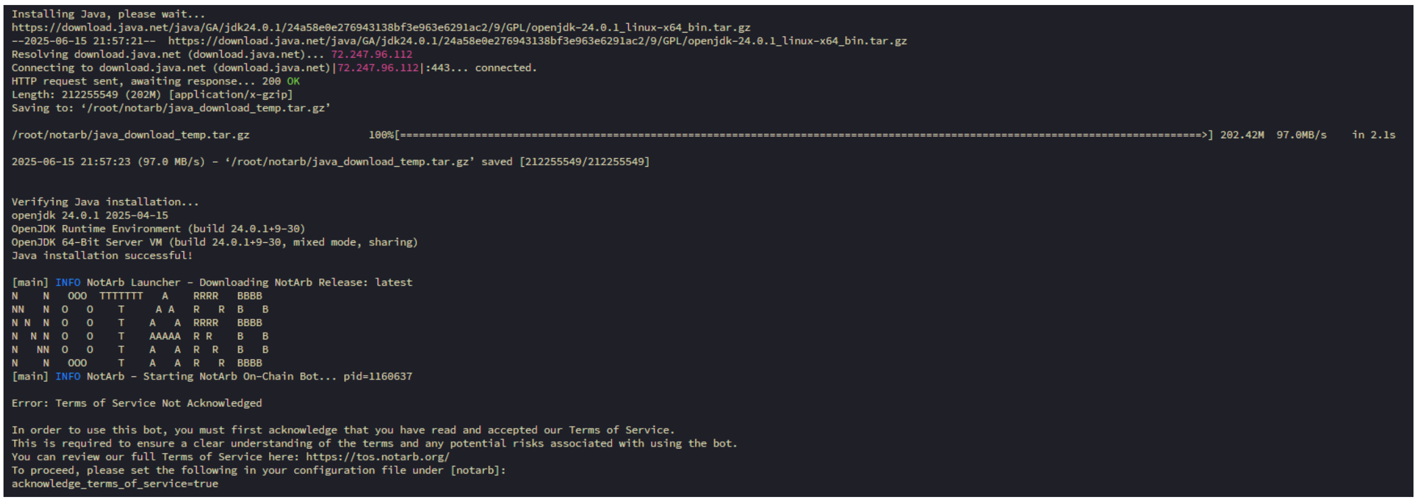
```
bash notarb.sh onchain-bot/example.toml
```

You should then see it start to install of of the requirements for Notarb.



```
Installing Java, please wait...
https://download.java.net/java/GA/jdk24.0.1/24a58e0e276943138bf3e963e6291ac2/9/GPL/openjdk-24.0.1_linux-x64_bin.tar.gz
--2025-06-15 21:57:21--  https://download.java.net/java/GA/jdk24.0.1/24a58e0e276943138bf3e963e6291ac2/9/GPL/openjdk-24.0.1_linux-x64_bin.tar.gz
Resolving download.java.net (download.java.net)... 72.247.96.112
Connecting to download.java.net (download.java.net)|72.247.96.112|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 212255549 (202M) [application/x-gzip]
Saving to: '/root/notarb/java_download_temp.tar.gz'

/root/notarb/java_download_temp.tar.gz              38%[===============================================>                                        ]  77.57M  94.3MB/s
```

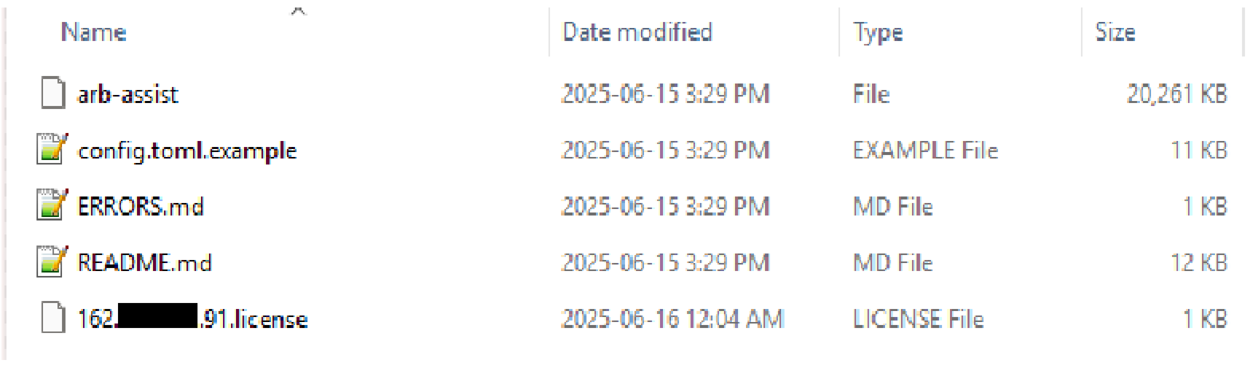
Then when it's finished installing everything, it'll look something like this:



```
Installing Java, please wait...
https://download.java.net/java/GA/jdk24.0.1/24a58e0e276943138bf3e963e6291ac2/9/GPL/openjdk-24.0.1_linux-x64_bin.tar.gz
--2025-06-15 21:57:21--  https://download.java.net/java/GA/jdk24.0.1/24a58e0e276943138bf3e963e6291ac2/9/GPL/openjdk-24.0.1_linux-x64_bin.tar.gz
Resolving download.java.net (download.java.net)... 72.247.96.112
Connecting to download.java.net (download.java.net)|72.247.96.112|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 212255549 (202M) [application/x-gzip]
Saving to: ‘/root/notarb/java_download_temp.tar.gz’

/root/notarb/java_download_temp.tar.gz                   100%[==================================================================================================================================>] 202.42M  97.0MB/s    in 2.1s

2025-06-15 21:57:23 (97.0 MB/s) - ‘/root/notarb/java_download_temp.tar.gz’ saved [212255549/212255549]


Verifying Java installation...
openjdk 24.0.1 2025-04-15
OpenJDK Runtime Environment (build 24.0.1+9-30)
OpenJDK 64-Bit Server VM (build 24.0.1+9-30, mixed mode, sharing)
Java installation successful!

[main] INFO NotArb Launcher - Downloading NotArb Release: latest
N    N   OOO  TTTTTTT   A    RRRR   BBBB
NN   N  O   O    T     A A   R   R  B   B
N N  N  O   O    T    A   A  RRRR   BBBB
N  N N  O   O    T    AAAAA  R R    B   B
N   NN  O   O    T    A   A  R  R   B   B
N    N   OOO     T    A   A  R   R  BBBB
[main] INFO NotArb - Starting NotArb On-Chain Bot... pid=1160637

Error: Terms of Service Not Acknowledged

In order to use this bot, you must first acknowledge that you have read and accepted our Terms of Service.
This is required to ensure a clear understanding of the terms and any potential risks associated with using the bot.
You can review our full Terms of Service here: https://tos.notarb.org/
To proceed, please set the following in your configuration file under [notarb]:
acknowledge_terms_of_service=true
```

You've now successfully installed all the required files to run Notarb.

Now we'll set up Arb-Assist. Begin by adding your license file obtained from Capicua to the file with the rest of your Arb-assist files, your file should look similar to this afterwards:



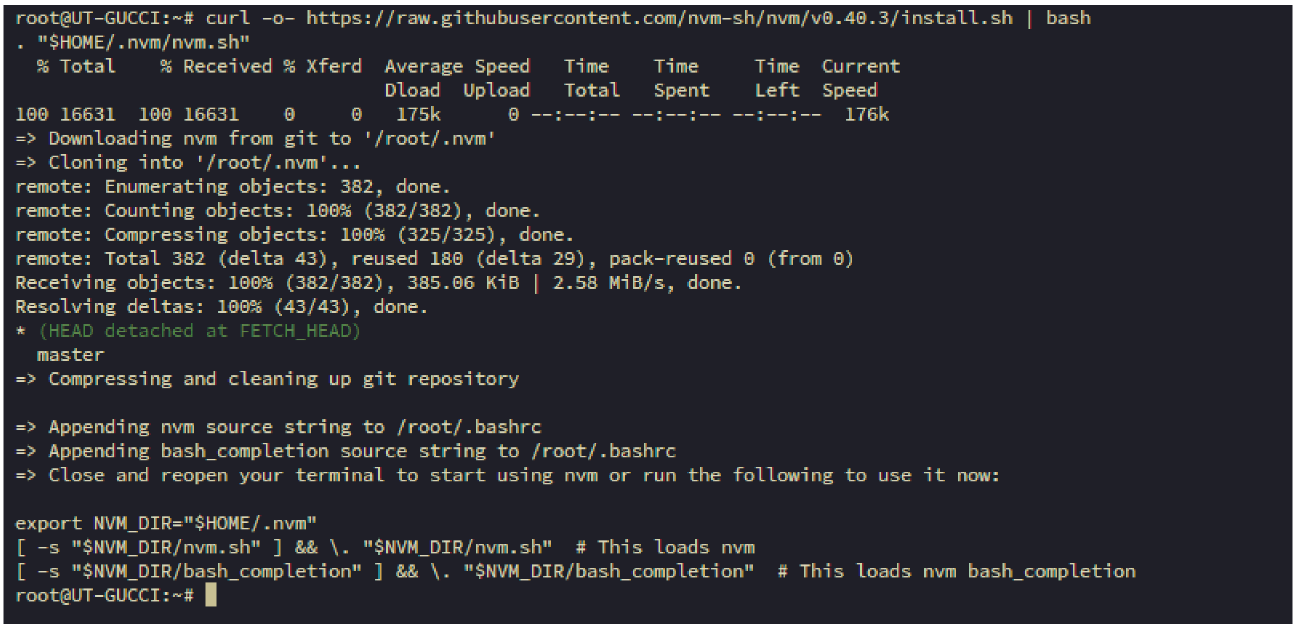
Name	Date modified	Type	Size
arb-assist	2025-06-15 3:29 PM	File	20,261 KB
config.toml.example	2025-06-15 3:29 PM	EXAMPLE File	11 KB
ERRORS.md	2025-06-15 3:29 PM	MD File	1 KB
README.md	2025-06-15 3:29 PM	MD File	12 KB
162[redacted].91.license	2025-06-16 12:04 AM	LICENSE File	1 KB

Now we're going to install all of the requirements to be able to run Arb-assist onto our server, this may differ depending on what you're using to run the bot but my example will utilize Ubuntu, if you have any troubles you can simply message me on discord Spoof (tag: spooft) or anyone else in the discord.

You'll begin by installing Node.js using NVM by typing the following command into your terminal:

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.3/install.sh | bash
. "$HOME/.nvm/nvm.sh"
```

It should show something similar to this:

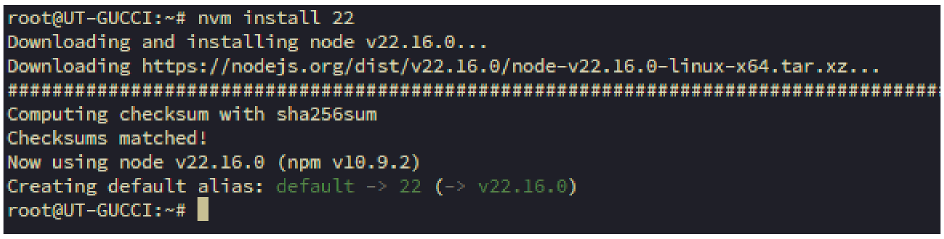


```
root@UT-GUCCI:~# curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.3/install.sh | bash
. "$HOME/.nvm/nvm.sh"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 16631  100 16631    0     0   175k      0 --:--:-- --:--:-- --:--:--  176k
=> Downloading nvm from git to '/root/.nvm'
=> Cloning into '/root/.nvm'...
remote: Enumerating objects: 382, done.
remote: Counting objects: 100% (382/382), done.
remote: Compressing objects: 100% (325/325), done.
remote: Total 382 (delta 43), reused 180 (delta 29), pack-reused 0 (from 0)
Receiving objects: 100% (382/382), 385.06 KiB | 2.58 MiB/s, done.
Resolving deltas: 100% (43/43), done.
* (HEAD detached at FETCH_HEAD)
  master
=> Compressing and cleaning up git repository

=> Appending nvm source string to /root/.bashrc
=> Appending bash_completion source string to /root/.bashrc
=> Close and reopen your terminal to start using nvm or run the following to use it now:

export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"  # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"  # This loads nvm bash_completion
root@UT-GUCCI:~# 
```

Then we'll run the following command in the terminal:

```
nvm install 22
```

It should look something similar to this:


```
root@UT-GUCCI:~# nvm install 22
Downloading and installing node v22.16.0...
Downloading https://nodejs.org/dist/v22.16.0/node-v22.16.0-linux-x64.tar.xz...
##################################################################################
Computing checksum with sha256sum
Checksums matched!
Now using node v22.16.0 (npm v10.9.2)
Creating default alias: default -> 22 (-> v22.16.0)
root@UT-GUCCI:~#
```

Now lets verify that we've obtained the correct version so that our program won't run without any errors, so run the following command in the terminal:

```
node -v
```

It should print at least version 22.15.0 or higher.

Then run:

```
nvm current
```

We want this to be 22.15.0 or higher.

Now we'll want to install the last two required programs which are PM2 and TMUX.

We'll do that simply by using the following terminal commands:

```
npm install -g pm2
```

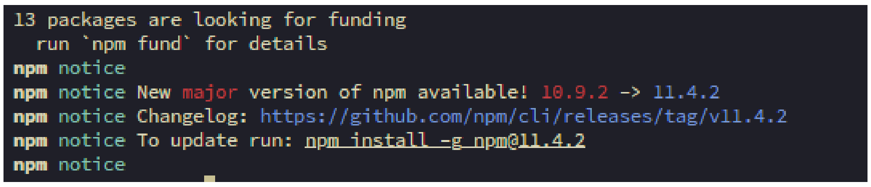
and

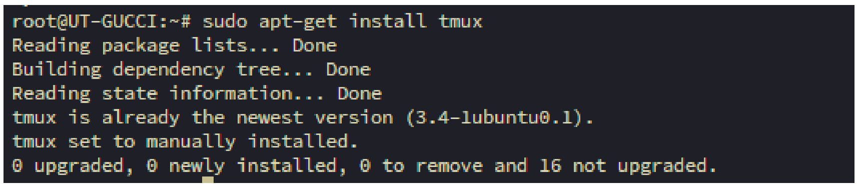
```
sudo apt-get install tmux
```

I personally also recommend utilizing a program called Screen, and will be using it during my explanation/guide. It can be installed by typing the following terminal command:

```
sudo apt install screen -y
```

The display for these installations are pretty straight forward, but it should look similar to this when a program is successfully installed/installed onto your server.

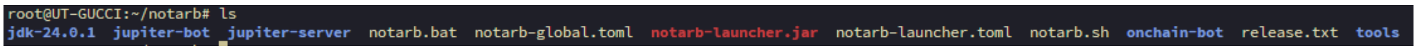

```
13 packages are looking for funding
  run `npm fund` for details
npm notice
npm notice New major version of npm available! 10.9.2 -> 11.4.2
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.4.2
npm notice To update run: npm install -g npm@11.4.2
npm notice
```


```
root@UT-GUCCI:~# sudo apt-get install tmux
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
tmux is already the newest version (3.4-1ubuntu0.1).
tmux set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 16 not upgraded.
```

Moving the files into proper layout

Now that we've successfully gotten Arb-Assist and Notarb in a position where they are capable of running independently, we'll need to merge them into the same file paths so we're able to use them in tandem.

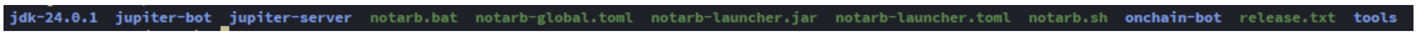
We'll begin by going into the directory where our Notarb files are located.


```
root@UT-GUCCI:~/notarb# ls
jdk-24.0.1  jupiter-bot  jupiter-server  notarb.bat  notarb-global.toml  notarb-launcher.jar  notarb-launcher.toml  notarb.sh  onchain-bot  release.txt  tools
```

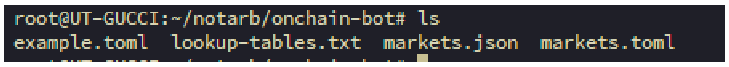
Now we'll want to ensure that all of the files in this location have privilege, so we'll run the following command to ensure that:

```
chmod +x *
```

After running that command, all files in that location will receive permissions to run - Most terminals will give a visual indication of the permission status of each file so you may double check if they have received permission depending if the colors have updated accordingly:

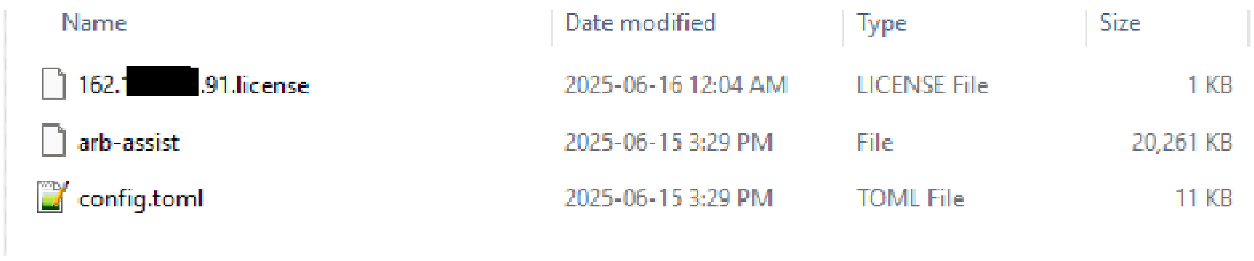

```
jdk-24.0.1  jupiter-bot  jupiter-server  notarb.bat  notarb-global.toml  notarb-launcher.jar  notarb-launcher.toml  notarb.sh  onchain-bot  release.txt  tools
```

Now we'll want to enter into the directory of the on-chain bot itself, when inside of the directory you should see the following files within:



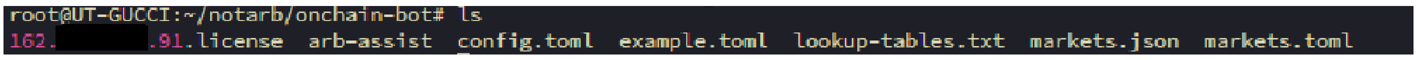
```
root@UT-GUCCI:~/notarb/onchain-bot# ls
example.toml  lookup-tables.txt  markets.json  markets.toml
```

Now we'll want to place the following contents from the Arb-assist program into this directory:



Name	Date modified	Type	Size
162.1[redacted].91.license	2025-06-16 12:04 AM	LICENSE File	1 KB
arb-assist	2025-06-15 3:29 PM	File	20,261 KB
config.toml	2025-06-15 3:29 PM	TOML File	11 KB

After moving these files inside of your onchain-bot directory, the directory should now contain all of the following files:



```
root@UT-GUCCI:~/notarb/onchain-bot# ls
162.[redacted].91.license  arb-assist  config.toml  example.toml  lookup-tables.txt  markets.json  markets.toml
```

Now we'll want to follow the steps again of giving all of the files in this directory permissions so we'll run the command once again:

```
chmod +x *
```

Editing your Arb-assist program's configuration

Now we'll need to edit our Arb-assist program in order for it to start generating configurations for us properly, we'll utilize Capicua's default configuration as an example of how we can get started, open the following URL:

```
https://github.com/capicua4454/arb-assist/blob/main/config.toml%20(notarb)
```

Now you'll notice right off the bat the first thing we'll need to do is fill in the following fields:

```
 rpc_url
grpc_url
grpc_token
grpc_engine
```

So we're going to start by simply providing our RPC and gRPC that we added earlier into notarb, into this configuration. If your GRPC uses a token then you'll provide that in the grpc_token field. As for the engine of your grpc that'll simply come down to the type of node you're using. But more often then not it will almost always be yellowstone.

Then we'll run into the following fields:

```
 mode
helius_key
```

The mode is the different types of configuration you'll want to generate, I recommend sticking with NA since that's the program we've used to set this up.

As for the helius key, you can get a free one simply by going to helius and registering an account, I'd recommend doing it for the benefit of dynamically gathering priority fees.

Next we'll be looking at the following:

```
 dexes
arb_programs
```

Dexes are the types of decentralized exchanges that will be added to your configuration, notarb currently supports the majority of them but it's in your best interest to check in and continue to add them as the onchain program is updated.

arb_programs are the on chain programs that Arb-Assist utilizes to read and gather data from the different transactions that happen on chain, you're able to add as many of these as you see fit but just keep in mind that you'll need to adapt your strategy as you add more.

Next we'll be looking at:

```
 exclude_mints
output
update_interval
run_interval
halflife
ignore_filters
include_token2022
```

Exclude mints is pretty self explanatory, it's the list of mint addresses you wish to be ignored by the program, I recommend always keeping USDC / SOL as an ignored intermediatory mint address.

Output is simply the file name, this only pertains if you're using "smb" mode.

Update interval is the frequency in which you want to update your data for the program, I'd recommend leaving this number low so your getting fresh up to date data.

Run interval is the duration in which you want a captured mint address to be attempted for after passing your filters.

Half life will essentially be the duration in which older transactions impact score effect the current rankings of your opportunuties, if you wish to be more selective and shut off the bot quickly after a spike occurs, keep this number low. If you want to continue arbing even after the activity has died off a bit, increase the number. This will depend on your own strategy.

Ignore filters - self explanatory.

include_token2022 is also pretty self explanatory, leave it true.

Now we'll be going into fields that really separate your strategy from the rest of the grouping. You'll need to do a lot of tweaking in order to really refine your strategy but there are some default considerations to take into account.

For filter thresholds we've got the following fields:

```
 min_profit
min_roi
min_txns
min_fails
min_net_volume
min_total_volume
min_imbalance_ratio
max_imbalance_ratio
```

Keep in mind off the bat that each of these numbers are represented in **lamports**. and it may take a bit of time to refine your strategy and may alter depending on the day of the week or the general condition of the market.

min_profit is simply the minimum amount of profit that has occured from a particular mint address.

min_roi is the minimum return on investment from other arbitraging on the mint address, which can be broken down into profit versus gas paid. (gas is the fees you pay when spamming)

min_txns are the minimum amount of successful arbitrage transactions on a particular mint.

min_fails are similar, the amount of total failed transactions on a mint address.

min_net_volume is the amount of volume after subtracting the sell volume from the buy volume.

min_total_volume is just the buy and sell volume added together

min_imbalance_ratio is the net volume divided by the total volume, I personally don't use this at all but you're welcome to test the waters with it and see if it can help with your personal strategy.

Now that you have a bit of a grasp on the different sorts of filters that are available to you, you can play with them and sort of feel out the sort of strategy you wish to go for. There are many factors to take into consideration when creating your strategy for filtering particular mints out. It's important to understand general market condition on a day-to-day basis and update your filters accordingly, such as weekends will be a bit slower so it might be best to drop your filters a bit - or when the market is super hot it may be better to increase your ROI so you're attempting on mints that are the best and not cluttering your strategy with low-profit mints. It's important to view your strategy from multiple angles when filtering mints as it's a make or break for profitability.

Next we'll look at the strategy_level fields:

```
 process_delay
max_cu_limit
top_pool_num
memo
```

Process delay is simply how frequently your transactions will be sent out, I don't recommend going higher then 400MS since it's crucial that you're hitting each block that occurs on chain.

Max Computer Unit Limit is the amount of computer units you allow an opportunity to utilize, I would recommend setting this a bit higher to start and monitoring and reducing as you see fit.

Top pool number, the number of top pools from your filtered out mints to utilize for sending transactions.

Memo I mostly utilize to differentiate between my strategies to view there success rates on chain, but you can use this for anything you see fit.

Next we'll view the seperate spam_levels:

```
spam
spam_bundle_groups
min_cu_percentile
max_cu_percentile
min_cu_price
max_cu_price
cu_strategy
```

Spam = true is simply setting spam to an enabled feature at that particular threshold.

Spam bundle groups is the groups you wish to utilize.

Minimum/Maximum CU percentile is utilizing the fees monitored on chain and putting you into a grouping of fees based on what others are paying. I recommend keeping this a modest level unless the market is extremely hot or you could burn gas very quickly.

Minimum/Maximum CU price is the total fee in lamports your willing to pay in total (It will not go under or surpass this amount if the percentile is higher/lower then the set amount)

CU Strategy is the way you add a fee to each transaction you send out, by default it's set to random which I also recommend using but many people will also try out linear / exponential.

Then we'll be setting our seperate sending_rpc_urls.

You'll want to add as many RPCs as you can into this field, utilizing any free ones you can find including helius etc. The more RPCs you can add typically the better off you'll be - again you want to be landing in every block possible to maximize your chance at landing a good opportunity.

Next we have our jito_levels, these follow a similar theme to the spam_levels although jito tends to be a lot cheaper, so I recommend in a quieter market you utilize Jito spam but in a more heated market you swap over to spam or utilize both spam and Jito to send transactions out.

Now we've got our seperate Jito fields to fill in:

```
use_dynamic_jito_tip
jito_uuid
jito_ips
jito_min_profit
jito_use_min_profit
jito_use_seperate_tip_account
block_engine_strategy
no_failure_mode
jito_urls
```

Use dynamic jito tip is just enabling / disabling the usage of dynamic tipping

Jito UUID is where you'll place your UUID key if you've got one. This will allow you to use a higher amount of RPS then the default 1.

Jito IPs is to use multiple ips to send requests to jito, I recommend this if you don't have a UUID.

Jito minimum profit is the amount of minimum profit to return from a trade, I'd recommend never enabling this.

Jito use minimum profit is enabling/disabling minimum profit.

Jito use seperate tip account, use a seperate tipping account for Jito, I recommend leaving it disabled for Jito spam.

Block engine strategy is the way your transactions will be sent out, there is OnebyOne and AllAtOnce, pretty self explanatory what these would do.

No failure mode is basically sending Jito transaction regardless of if they pass simulation, it'll constantly attempt to spam them and hunt for profits. I recommend always leaving this enabled.

Jito URLS are the different Jito endpoints that you'll be sending transactions to, you'll want to have as many of these added as possible and if more are added in the future be sure to add those in.

Then finally we'll be looking at the Notarb specific configurations.

```
output
keypair_path
protect_keypair
threads
meteora_bin_limit
ips_file_path
prefunded_keypairs_path
jito_regions
mints_to_arb
max_bundle_transactions
borrow_amount
invalid_account_size
buffer_size
```

Output is the file name that'll be output from the configuration, leave this as is or change it if you wish.

Keypair path you've already set earlier in the document, but it's just the directory to your private key.

Protect Keypair, I highly recommend this as it will encrypt your private key on your system to make it more difficult for anybody to access.

Threads are simply the amount of threads to utilize, I recommend just keeping this to 0 as it will utilize as dynamic cached thread pool.

Meteora bin limit, the amount of bins the bot will search in to try and find an opportunity, by default there are 68 bins but I don't recommend setting this to high as it can cause transactions to fail.

IP files path is the same as before, it's the path to IPs to utilize if you don't have a Jito UUID.

Prefunded Keypairs is a list of private keys of prefunded wallets if you wish to utilize proxy wallets, I don't recommend it for our strategy.

Jito regions is the same as before, it'll be the regions you send transactions to.

Mints to arb is the number of top mints to utilize from our filtered ones.

Max bundle transactions is the maximum amount of mints to search for opportunities with in a bundle, setting this to high can cause the transaction to fail but utilizing 2-3 is fine if you set the proper CU limits.

Borrow amount is the amount of SOL to borrow in a transaction, I recommend definitely keeping this always as there is no real downfall to having more SOL to use for arbitrage except for a bit of CU. The number is set in lamports.

Invalid account size is the default size to use for invalid account that contain no data.

Buffer size is the amount of extra bytes to append to the total size, this can be set to 0 but I recommend having a bit of buffer.

Running our program

Now that we've set up our program properly and have set up our strategy, it's time for the fun part!

We're going to begin by opening a new Screen instance on our server by running the following command (keep in mind you can name the screen whatever you want, I just name mine arb-assist.)

```
screen -S arb-assist
```

This will open a new Screen instance on your server which we can utilize to run Arb-Assist on without interruptions.

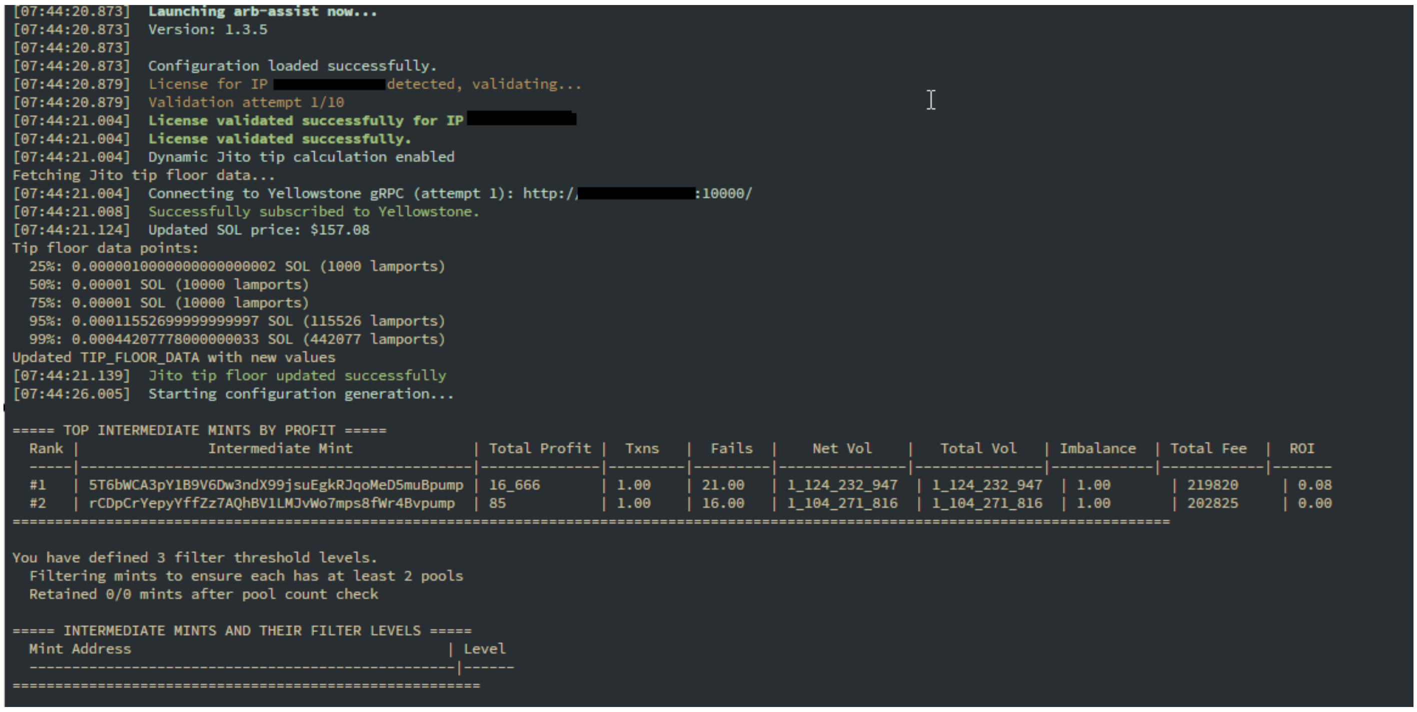
First we'll start by increasing our ulimit with the following command:

```
ulimit -n 65536
```

Now we're going to run the Arb-Assist program by running the following command:

```
./arb-assist
```

You should see it load your license and begin to pull data.



```
[07:44:20.873]  Launching arb-assist now...
[07:44:20.873]  Version: 1.3.5
[07:44:20.873]
[07:44:20.873]  Configuration loaded successfully.
[07:44:20.879]  License for IP [redacted]detected, validating...
[07:44:20.879]  Validation attempt 1/10
[07:44:21.004]  License validated successfully for IP [redacted]
[07:44:21.004]  License validated successfully.
[07:44:21.004]  Dynamic Jito tip calculation enabled
Fetching Jito tip floor data...
[07:44:21.004]  Connecting to Yellowstone gRPC (attempt 1): http:/[redacted]:10000/
[07:44:21.008]  Successfully subscribed to Yellowstone.
[07:44:21.124]  Updated SOL price: $157.08
Tip floor data points:
  25%: 0.0000010000000000000002 SOL (1000 lamports)
  50%: 0.00001 SOL (10000 lamports)
  75%: 0.00001 SOL (10000 lamports)
  95%: 0.00011552699999999997 SOL (115526 lamports)
  99%: 0.00044207778000000033 SOL (442077 lamports)
Updated TIP_FLOOR_DATA with new values
[07:44:21.139]  Jito tip floor updated successfully
[07:44:26.005]  Starting configuration generation...

===== TOP INTERMEDIATE MINTS BY PROFIT =====
  Rank |               Intermediate Mint                | Total Profit |  Txns    |  Fails   |    Net Vol     |   Total Vol    | Imbalance  | Total Fee  |  ROI
  -----|------------------------------------------------|--------------|----------|----------|----------------|----------------|------------|------------|-------
  #1   | 5T6bWCA3pY1B9V6Dw3ndX99jsuEgkRJqoMeD5muBpump | 16_666       | 1.00     | 21.00    | 1_124_232_947  | 1_124_232_947  | 1.00       | 219820     | 0.08
  #2   | rCDpCrYepyYffZz7AQhBV1LMJvWo7mps8fWr4Bvpump  | 85           | 1.00     | 16.00    | 1_104_271_816  | 1_104_271_816  | 1.00       | 202825     | 0.00
=====================================================================================================================================================

You have defined 3 filter threshold levels.
  Filtering mints to ensure each has at least 2 pools
  Retained 0/0 mints after pool count check

===== INTERMEDIATE MINTS AND THEIR FILTER LEVELS =====
  Mint Address                                     | Level
  -------------------------------------------------|------
=======================================================
```

Now that the program is running, it should be successfully outputting files that our Notarb bot can utilize and work off of, double check to ensure it's working as intended and check for the output file name (this will match the one you placed earlier) as well as outputting a markets.json file.

Once we've confirmed it's outputting everything correctly, we'll simply back out of our screen instance by typing the following:

```
 ctrl + A
d
```

If we ever want to re-enter the screen instance, we'll type:

```
screen -r NAME
```

Name being what you initially called the screen instance, and if we ever want to kill the screen we can do it in two ways:

```
killall screen
```

or you'll go into the screen instance and type:

```
 ctrl + A
k
y
```

Now that we're done with Arb-assist, we're going to turn on our Notarb bot and have it listen to our files for any updates - and upon them updating our bot will automatically restart with the new data that Arb-Assist output for us.

Back at the main screen of your server we're going to input the following command:

```
pm2 start ./notarb.sh --interpreter bash --name notarb-onchain --watch onchain-bot/notarb-config.toml -- onchain-bot/notarb-config.t
```

We'll need to ensure that we're using the correct output name that we had previously though, by default it's set to notarb-config but if you've renamed it then you need to replace notarb-config with the name you chose for your configuration file.

And that's it! We're up and running.

Final thoughts

Running an arbitrage bot is not a quick and easy way to get money - as nothing in life is. It requires a lot of learning and reading/understanding data along with market conditions.

You'll need to constantly update you strategies and also find which strategy works best for you - rather it be to heavily spam hot mints with high tipping fees or to utilize a more reserved strategy that only uses Jito spam and low tips. It's not something that anybody can figure out for you but rather a trial and error process.

I hope that this document has helped you learn a bit more about utilizing Arb-Assist with Notarb, and if you have any questions feel free to reach out to me on discord Spoof (tag: spooft).

It's a long journey for everyone to find there profitable strategy. But being profitable is definitely possible with a bit of effort!