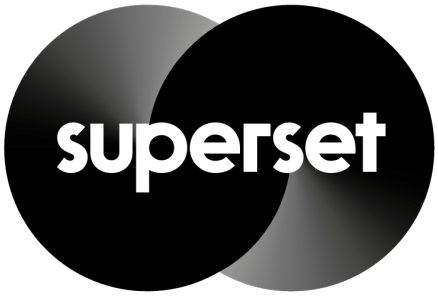
Native Liquidity Network For Stablecoins

Whitepaper

Superset Technologies Inc | neil@superset.finance

August 2025



Neil Staunton | CEO
Ben Haslam | CTO

INTELLECTUAL PROPERTY NOTICE

Contents

1 Introduction

Thanks to cross-chain interoperability protocols like LayerZero, it is now easier than ever for token projects to deploy their tokens on multiple chains. Using token standards like LayerZero's OFT, tokens can be moved between chains by burning them on one chain and minting them on another. Although this allows token projects to reach a wider audience, deploying or taking a token multichain can take weeks of development time and resources, and it also creates a problem for liquidity providers. This protocol is primarily designed for Stablecoins and RWAs, but it can be used with any OFT token.

The Superset Native Liquidity Network introduces the SuperFactory, which automatically deploys and manages multichain tokens in one transaction, and SuperPools, which solve the problem of fragmented liquidity.

If a token is deployed on multiple chains, the liquidity for that token is fragmented across all those chains, which can lead to high slippage for large trades. This problem is exacerbated as there are often multiple DEXs on each chain, each with their own liquidity pools. This means that a user who wants to make a large trade between two tokens on different chains would have to make multiple trades, each with its own slippage, which can result in a significant loss of value.

DEX aggregators can make a start at solving this problem of fragmented liquidity, by routing trades through multiple DEXs to find the best price. However, these solutions are limited by the liquidity available on a particular chain, and cannot take advantage of the liquidity available on other chains. This is where SuperPools come in.

A virtual pool is a liquidity pool spread over multiple blockchains, where they all share the same liquidity, and therefore the largest trade sizes can be executed with minimal slippage.

Each liquidity pool is a pair, made up of a token (StableGBP, StableEURO, RWA...) paired with a liquidity token, a stablecoin with deep liquidity (USDT). For clarity, this token will only be denominated in USD.

When a user makes a swap between two assets using SuperPools, say StableGBP and StableEURO, they will in fact be doing two swaps: StableGBP $\rightarrow$ USD, USD $\rightarrow$ StableEURO. They will make swaps through a router contract which will then create the swap requests to be sent to SuperPools. Most of the tokens being traded are OFTs, including USDT via USDT0, so they can effectively be moved cross-chain by burning and minting.

When a user makes a swap request to be paid out to them on the same chain (not a cross chain swap), there is no need to actually transfer the tokens to the hub chain, rather the intent of what they want to do is sent to the hub chain, which will calculate the output of the swap request, and then send the output back to the user, who is then paid back using funds stored in the pool on the local chain.

This means that cross chain transfers of tokens are only needed when the user wants to be paid out on a different chain than the one they made the swap request on, or for when the hub chain needs to rebalance the liquidity in the pools. For this, OFT transfers will be used by the protocol. In the case where the token is not an OFT, for example Eth or Sol, cross chain bridges with large liquidity (e.g. Stargate Protocol) will be used to transfer the tokens between chains.

2 The SuperFactory

2.1 Introduction to the SuperFactory

In addition to SuperPools, we have created the SuperFactory, which allows any token project or issuer to create their own OFT token, which can be used in the SuperPools system. The SuperFactory is a smart contract and SDK that can be used to automatically deploy and setup multi-chain tokens with a single transaction to one chain, then manage the token across all chains after deployment. The user will then be able to use this OFT token in the SuperPools system, and will be able to transfer it between chains using the LayerZero protocol.

The SuperFactory can deploy and configure OFT tokens and adapters, but is also an OApp itself. This means that a factory on one chain is able to send and receive messages from factories on other chains. The reason for adding this OApp capability is so that a user is able to deploy a new OFT token to multiple chains with just one transaction to one chain. The initial factory will receive the transaction, and split it into sub-transactions for each selected chain, and will then forward those messages to factories on the other chains, where they will then deploy the OFT contracts on each chain. When all is finished, through one transaction a user will have deployed a multichain token across as many chains as they like, with all the setup taken care of.

The SuperFactory therefore is an abstraction layer that allows users to deploy and manage their own OFT tokens, without having to worry about the underlying complexity of the LayerZero protocol. The SuperFactory is a powerful tool that will enable users to easily create and manage their own OFT tokens, and will also onboard users into the SuperPools protocol, by setting them up with the correct standard.

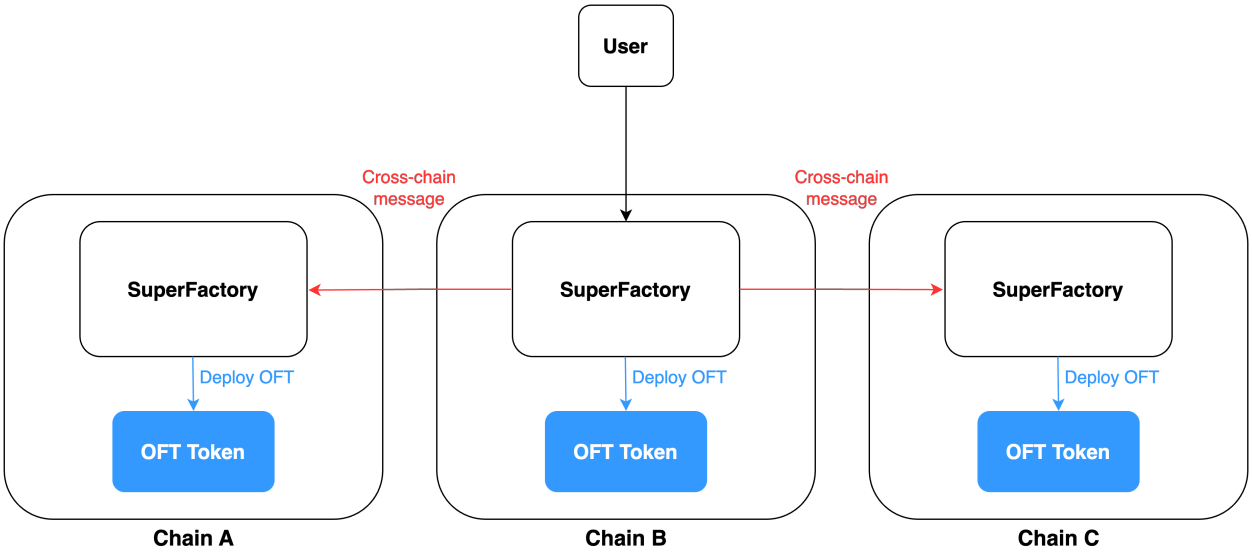


Figure 1: The factory on the initial chain receives the transaction, and splits it into sub-transactions for each selected chain, to deploy and configure the OFT token all at once

2.2 Local OFT Deployments

2.2.1 Create Functions

The SuperFactory has functions `_CreateOFT` and `_CreateOFTAdapter`, which both use the `create2` opcode to deploy an OFT contract to a deterministic address. To load the bytecode, external libraries are used, which are set in the constructor along with other default values or can be optionally passed into the functions for a custom implementation.

The factory automates the deployment, setup and peering of an OFT token by calling LayerZero contracts and functions directly. This can be seen in the `configure` function, which is called by both `_CreateOFT` and `_CreateOFTAdapter`. These functions take the parameters used to create the token, as well as an array of `peer` structs, which define the peers on different networks that will automatically be added.

2.2.2 Compute Address

The factory has an overloaded function `computeAddress`. This takes the same parameters as `_CreateOFT` and `_CreateOFTAdapter`, and will return the address that those functions would deploy to. This means that it can be used to get the address of an OFT token before it is deployed, so if an OFT token is deployed to two different chains at the same time, they can automatically peer with each other on deployment. This functionality is exposed with the SDK, so using any one factory the user can find the deployment address of an OFT on any other factory. If the parameters for this being passed to `create2` are consistent (factory address, token name, symbol, etc.), then the token will be deployed to the same address on all the different chains.

2.2.3 Token Bytecode

For deploying a new OFT or OFTAdapter, the SuperFactory uses libraries to get the deployment bytecode. These libraries must be deployed first and then passed in as constructor arguments for deploying the factory. This repo provides default libraries and the SDK will automatically deploy them before deploying the factory, however, these libraries can be swapped out for custom implementations of OFT and OFTAdapter for a specific use case. The SDK exposes this functionality so any implementation of an OFT can be deployed across multiple chains as long as bytecode libraries for those are pre-deployed on each chain.

2.2.4 OAPP Admin

The OAPP factory (and therefore its implementations like the SuperFactory) maintains a mapping of deployed OAPP addresses to user addresses called admins. This admin can be thought of as the owner of the deployed OAPP (OFT), who can control the OFT through the factory. When `_CreateOFT` and `_CreateOFTAdapter` are called, the factory itself is set as the owner and delegate of the new OFT contract, but sets the sender as the admin. This means that the factory can be used in the future to remotely or locally add new peers or update the ULN config for that OFT, but will always check that the sender == admin for that OFT.

If the admin wants to take full control of a new OFT themselves without the factory, they can call `transferOwnership` on the factory to set themselves as the owner and delegate. Conversely, if a user wants to bring an already existing OFT into the Factory ecosystem, so the factory can be used to manage future deployments and changes, they can call `setAdmin`, then set the factory as the owner and delegate for that OFT.

2.2.5 Configure an OFT

The `_configureOAPP` function on the factory is called to setup a new or existing OFT, to connect it to multiple peers on other networks, so they will be able to send OFT transfers cross chain in the future. This function takes an array of `Peer` structs, which define the endpoint ID, address and ULN config for that peer. The `_configureOAPP` function iterates through this array and peers the local OFT with its multichain counterparts, and sets up the configuration so they will then be able to send messages to each other with LayerZero. This function can be called locally by the admin with the public function `_configureOAPP`, or it is called remotely when the factory receives a cross chain message.

2.3 Remote OFT Deployments

2.3.1 Remote Create

In order to deploy OFT tokens to other chains, the OAPP Factory (abstract contract that is implemented by the SuperFactory) has a `_remoteCreate` function, which calls the LayerZero endpoint contract to send a cross chain message. This message includes standard LZ data for cross chain messages, and instructions of what the receiving factory should do; i.e. deploy a new OFT token or adapter and add new peers, or configure an existing OFT contract (to add new peers). For sending messages to other EVM chains, the `_remoteCreate` function will test the cross chain message that is about to send, to check that the format is correct to execute on the receiving chain.

2.3.2 Create Multiple Contracts at Once

The `_remoteCreate` function is internal and cannot be called directly. Instead, it is called by one of two external functions, `createOappMulti` or `configureMulti`. These functions first complete an operation on the local chains

(creating or configuring an OFT), then will iterate through an array of remote messages to call `_remoteCreate` for each one, to either deploy new contracts or to configure contracts on other chains.

These functions are what allows the SuperFactory (and any other contract implementing the OAPP Factory) to deploy or update a multichain token at once with only one transaction. That transaction is split into operations that are sent to each destination chain, to deploy and setup all the individual smart contracts at once.

In order to achieve this, the single transaction that is sent to the initial chain must include all the data to tell each factory on every other chain what to do. This means, for deploying a new token to multiple chains for example, each cross chain message should include all the peers that the new token should add, and optionally what ULN config to use for each one. This would be a lot of orchestration to expect end users to do themselves, so we have implemented functionality in the SDK to do this automatically, with the functions `createOftMulti` and `configureOappMulti`.

2.3.3 Receiving a LayerZero Message

In order to receive a cross chain message, the SuperFactory overrides the `_lzReceive` function (as it is implementing the `OAPP` abstract contract). This takes the encoded data, and decodes it to `LzReceiveData` struct, which contains information on the sender and what operation the factory should carry out, as well as more encoded data, which will be used to carry out that operation, either calling `_createOFT`, `_createOFTAdapter` or `_configureOAPP`.

3 SuperPools

SuperPools is a set of virtual liquidity pools spread over multiple blockchains, that will allow each chain to share the same global liquidity. This means that traders on any chain will be able to access all the token's liquidity when they make a trade, so they will be able to make large trades with minimal slippage.

The infrastructure of SuperPools consists of a hub chain and multiple spoke chains. The hub chain is the chain that will be used to aggregate liquidity from all the spoke chains, and will be used to process trades between the spoke chains. The hub chain has an internal AMM used to simulate trades using the virtual liquidity, which is how it processes user trades. The spoke chains are all the chains where the token is supported, each will have a local liquidity pool of supported tokens, which are used for liquidity onboarding and rebalancing, as well as processing the swap result; paying a user out with the result of their swap.

A user can make a swap request on any supported chain, and the hub chain will process the swap request and send the swap response back to the spoke chain. The spoke chain will then pay the user out with the result of the swap. If the user requests to be paid out on a different chain than the one they made the swap request on, the hub chain will send the result of the swap to the spoke chain, where it will be paid out to the user, and send a message to the initial spoke chain to close the swap request. This means that cross chain swaps are possible with SuperPools.

3.1 Hub Chain virtual AMM

The hub chain has 'mirror tokens'', which mirror all the tokens contained in the spoke chains within the protocol. Whenever more tokens enter the protocol, via liquidity onboarding or a user making a swap, mirror tokens are minted to represent the new tokens on the hub chain. Mirror tokens are burned whenever tokens leave the protocol in a similar fashion. This means that the internal mirror tokens on the hub chain will perfectly match the tokens out the spoke chains, so any AMM operation applied to the mirror tokens can be used to get exchange tokens on the spoke chains.

The AMM in question is Uniswap v3, which is on the hub chain and is used to create pools of mirror tokens. When a swap request comes in from a spoke chain, the input tokens are mirrored (mirror tokens minted), then the output of the swap is calculated using Uniswap on the pools of mirror tokens (by just calling swap on the hub chain). The output mirror tokens are then burned, and the result (including the amount out) is sent back to the user on the spoke chain.

USDT0 will be used as the base trading asset for the majority of tokens, i.e. pools will be setup for a given token against USDT0, but it will be possible to create different trading pairs for specific use cases, i.e. GBP - EURO.

3.2 Spoke chain liquidity pools

As mentioned, liquidity pools on the spoke chains hold the active liquidity in the protocol. For this we are using a singleton model, meaning there will only be one liquidity pool per chain, which will hold all the protocol assets for that chain. This means that for example if there are four different virtual pools that have USDT0 as a token that include a given chain, the spoke chain liquidity pool contract will hold all the USDT0 liquidity for that chain, rather than having four different contracts, one for each virtual pool.

Of course, in the hub chain these different pools are accounted for, but by sharing one pot of liquidity on the spoke chain, the effective liquidity available for trades is much higher. When there is un-even rates trading on the same pool across different chains, the protocol needs to rebalance the liquidity tokens to fix the ratio on each chain to what it should be as per the hub chain (this mechanism is described in more detail in section 4.3). With this singleton design, the protocol can effectively rebalance several pools on the spoke chain at once, rather than having to rebalance each pool individually.

Finally, rebalancing will be required less often, as the liquidity drain of a token from one pool on the spoke chain could be cancelled out by the liquidity added of that token to another pool on the same chain, so rebalancing that token is not necessary. These liquidity abstractions are needed to make the protocol as efficient for end users as possible.

3.3 SuperPools Onboarding

After a new or existing token has been deployed across several chains using the SuperFactory as shown above, it can be integrated into SuperPools. This involves deploying liquidity pools on each chain for that token, which then connect to the hub chain to aggregate liquidity from all the spoke chains.

When liquidity is added or removed from the pools, through onboarding or due to a swap, the new balance of the pool is sent to the hub chain, where it is aggregated with the other pools. This means that the hub chain will always have the most up to date reserves of the token, and will be able to process trades between the spoke chains.

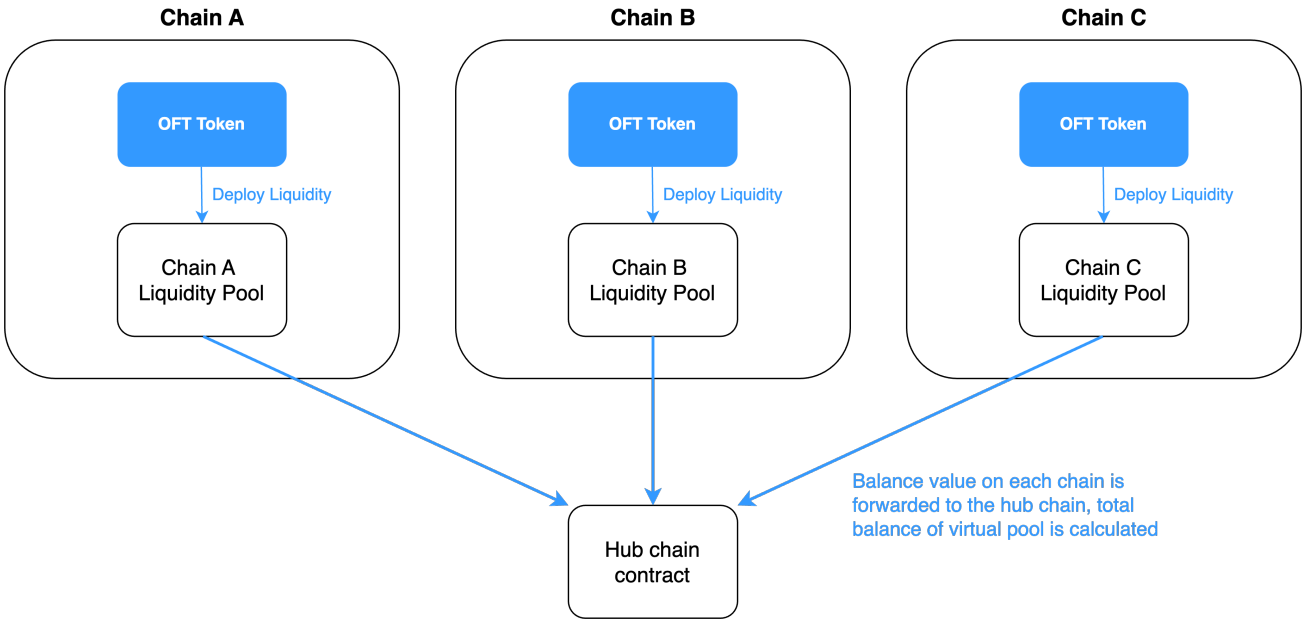


Figure 2: Token integrates with SuperPools: Liquidity is deployed locally on each chain, total reserves are calculated by the hub chain contract

3.4 SuperPools User Flow

3.4.1 Step 1 - User makes a swap request

The user makes a swap request between two tokens, which is passed from the router contract to SuperPools. The swap requests can be thought of as intents, to swap one token for a given range of another, which will then be received on a given chain. This swap request is sent to the hub chain, which will then be used to process the result of the swap request based on the total virtual liquidity pool available across all chains.

3.4.2 Step 2 - Swap is executed

The hub chain contract receives the swap request, and will then calculate the amount out using the total liquidity of the virtual pools of the two tokens being swapped. Generally there is one pool per token, which is a pair of the token and USD. So if a user wants to swap between two tokens, say TA and TB, the hub chain contract will first swap TA for USD, then swap USD for TB. As the USD being moved is the same for both of these, none actually needs to be moved, so the net result is that the balances of the pools of TA and TB are updated, and the user is paid out with the result of the swap.

The hub chain will then send the result of the swap request back to the spoke chain, where the user will be paid out with the result of the swap (TB). As the ratios of the tokens in the pools on each chain have now changed (and will be continuously changing with every swap), the protocol can rebalance the pools on individual chains, to ensure that there is enough liquidity in each pool for future swaps. We have an incentive mechanism in place to ensure that the pools are rebalanced, which will be described in more detail later.

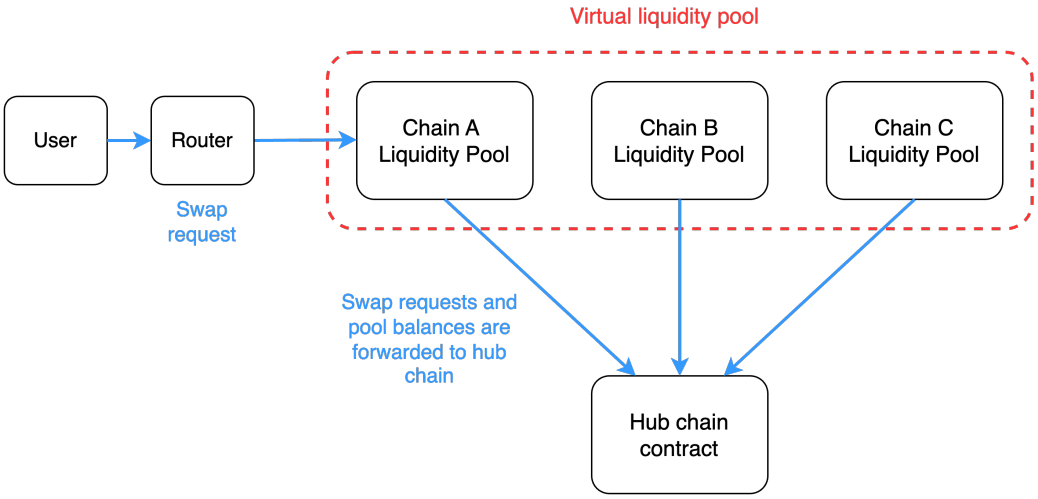


Figure 3: User makes a swap request to SuperPools. Request is forwarded to the hub chain where amount out can be calculated using global reserves

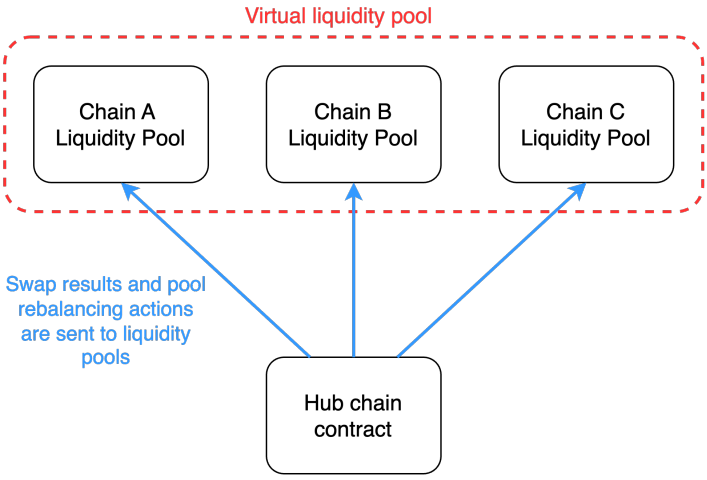


Figure 4: The hub chain contract executes the swap request, and sends the result back to the spoke chain

3.4.3 Step 3 - User payment and pool rebalancing

The output of the swap request is now sent to the user, and pools exchange liquidity to ensure that each one has enough for future swaps. There is no reason why a user would have to be paid out from the same chain they paid in. It would be simple to provide a destination chain and address, making cross chain swaps possible with SuperPools.

3.5 Additional Liquidity Onboarding

In order to add more liquidity to a given virtual pool, a user has a couple of options. The most simple option that does not shift the price is to provide an equal value of both tokens supported by the pool (TokenA / USD). This will increase the volume of both sides of the pool, but will not change the price/ratio. This is the standard for constant product AMMs.

Another option is to add an amount of the given token (TokenA) as well as the equal value of another token (TokenB) that is supported by SuperPools. This will internally swap TokenB for USD, then use that to add liquidity to the TokenA / USD pool. This will have the potentially unintended effect of increasing the TokenB / USD ratio, therefore decreasing the price of TokenB.

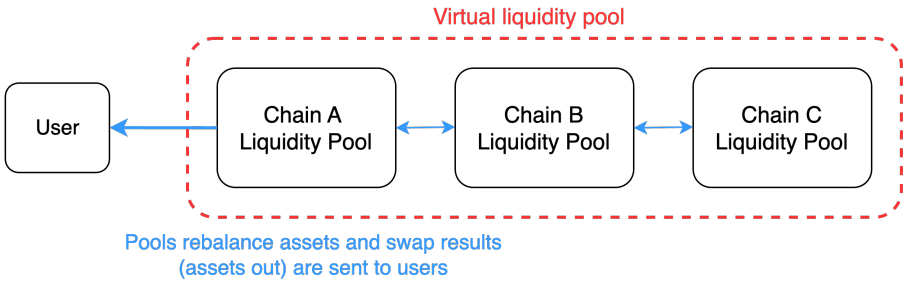


Figure 5: Pools exchange liquidity and the user receives their funds

The final option would be to just add TokenA with no other tokens for the other side of the pool. This approach would increase the ratio of TokenA / USD, therefore would decrease the price of TokenB.

For a new token that is not currently supported by SuperPools, a vote would be done by Superset token holders to whitelist the new token on all supported chains and create the new pools and virtual pool. From there, the liquidity must first be added using one of the first two methods described above to set the initial price for the new token.

3.6 Security Measures

SuperPools is built on top of LayerZero, as it was chosen as the most secure and efficient cross chain messaging protocol, with zero hacks to date. Despite this, we have implemented additional security measures to ensure that the system is secure and robust.

A concern a user could have is that their swap request is sent on the spoke chain, but something goes wrong so it does not reach the hub chain, or the response is not sent back to the spoke chain. In this case, the user would be left with no funds and no way to get them back. To prevent this, we have implemented a timeout mechanism, where if the swap response is not received after a certain block limit, the user can call a function on the spoke chain to get their funds back. This function will check the status of the swap request on the spoke chain, and if it was not executed, the user will be able to get their funds back.

3.7 Just In Time Liquidity

We also introduce the concept of Just In Time (JIT) liquidity, which allows the protocol to automatically move liquidity from the hub chain to one of the spoke chains if the required liquidity is not available on that chain. As this requires an OFT transfer to move the assets, it will cost more gas for the user, but it allows a user to make a trade on a chain that does not have enough liquidity. Ideally, through the rebalancing mechanism described below, the active chains will always have enough liquidity to settle trades, but in the case that they do not, JIT liquidity will be used to move the liquidity from the hub chain to the spoke chain.

3.8 What About Atomic Swaps?

Due to the cross chain nature of SuperPools, there is a delay between the time a user makes a swap request and the time they receive their funds. This means that atomic swaps are not possible on the spoke chains, which could limit their use cases is places like DEX integrations with other protocols, and in high frequency trading strategies. We are implementing a solution to this however, with the following mechanisms:

1. **Hub Chain Atomic Swaps**: For users trading on the hub chain itself, the process for this will be the same as for spoke chains described above, except instead of making cross chain messages to the hub chain contract, it will simply be local calls to the hub chain contract. This means that the user will be able to make atomic swaps on the hub chain, and will be able to use the liquidity on the hub chain for their trades. This will enable high frequency trading strategies on the hub chain, in situations where any latency is unacceptable.
2. **Post-Swap Execution Logic**: We will allow users to pass in post swap execution logic with their swap request, which is additional logic that will be executed after the swap is executed, so when the user has

the liquidity in their wallet. This means that SuperPools can be used with other protocols, such as lending protocols, where the user can pass in the logic to lend out their funds after they have been paid out. This will require the user to add extra gas to pay for the execution of the post swap logic, which can be specified by developers using the ‘extraOptions‘ field for making a cross chain message. This will be a powerful tool for developers to use, as they can create their own logic to be executed after the swap is executed, and can be used to integrate with other protocols.

3. **Hooks**: Similar to Uniswap V4, we will allow issuers creating a new pool to add hooks to the pool, which are additional logic that will be executed before or after a swap is executed, or before or after liquidity is added or removed. This will allow issuers to create their own logic to be executed when users interact with their pools, which can be used to integrate with other protocols, or to add additional functionality to their pools. This is completely optional for issuers, but allows them to build on top of SuperPools in any way they choose.

4 Cross-Chain Liquidity Optimization

4.1 Executive Summary

Superset introduces a global liquidity system that addresses the fundamental inefficiency of fragmented blockchain liquidity while building lasting infrastructure for cross-chain capital optimization. The system's economic architecture recognizes that while chains currently compete for TVL, liquidity will ultimately flow to genuine utility. It is therefore important to rebalance the liquidity on individual chains to ensure there is always enough liquidity to sufficiently settle trades on each chain. Therefore a fee mechanism must be used to incentivize LPs to reallocate liquidity between chains, to ensure that the pools are always well balanced.

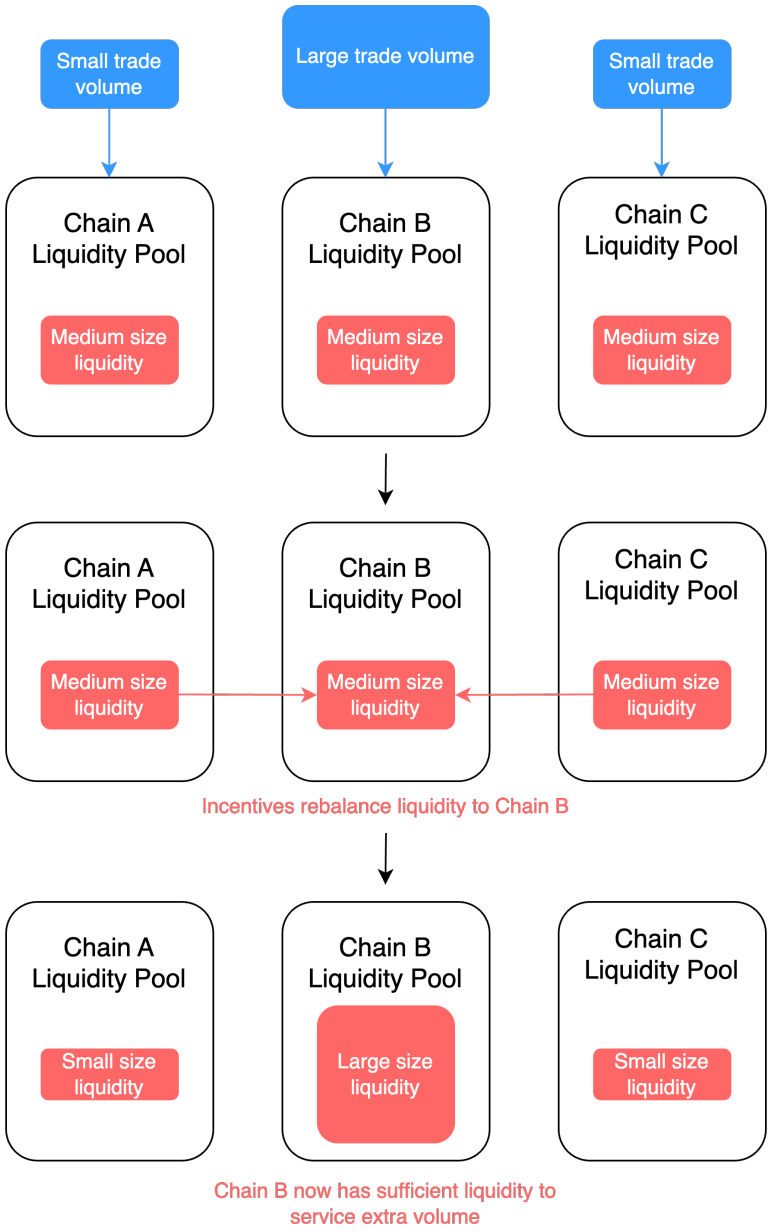


Figure 6: The protocol uses incentives to rebalance the pools on each chain, to ensure that there is always enough liquidity to settle trades

4.2 Two Key Mechanisms for Rebalancing

Superset's economic architecture is designed to optimize cross-chain liquidity through two key mechanisms:

- Rebalancing token ratios across spoke chains, described in section 4.3.
- LPs stake their LP position to a specific chain, which redirects their liquidity to that chain and allows them to earn rewards based on the usage of the liquidity on that chain. This process is described in more detail in section 4.5.

When tokens are being rebalanced using either of the mechanisms described above, during the cross chain transfer the representation of the tokens to be moved on the hub chain go from 'src chain' to 'pending'. These tokens can no longer be used to pay out users from any chain (although they do still go into the total liquidity of the virtual pool). Once they are confirmed to have landed on the destination chain, the hub chain will update its internal representation and these tokens can then be used to pay out users on the destination chain.

4.3 Rebalancing Token Ratios

In addition to the LP staking mechanism, Superset implements a rebalancing mechanism to ensure that the token ratios across spoke chains are always in line with the hub chain. This is done by monitoring the token ratios on each chain, and if they deviate from the ideal ratios as described on the hub chain (the sum of LP positions staked to a given chain across pools operating on that chain), the protocol will rebalance the pools on each chain to bring them back in line.

As alluded to in section 3.2, when there is un-even rates trading on the same virtual pool across different chains, this will cause the token ratios to deviate from the ideal ratios as set by the hub chain. For example, say you have the same virtual pool of Token 1 and Token 2 on two chains, Chain A and Chain B, with the same token ratios. If a users on Chain A are on average buying more of Token 1, and users on Chain B are on average buying more of Token 2, this will cause a net flow of Token 2 to Chain A and Token 1 to Chain B.

This means that the token ratios on Chain A will be different from those on Chain B, so the protocol should rebalance the pools on each chain to bring them back in line, effectively cancelling out the net flow of tokens between the two chains. This action can be done by anyone for an incentive reward. The user will specify the chains and tokens they wish to rebalance, and the protocol will check that this action is needed, and if so, will execute the rebalancing action. The user will then be rewarded with Superset tokens, in proportion to the amount of liquidity rebalanced.

In practice, with the singleton model described in section 3.2, there will be multiple pools using the same tokens on each chain, so the protocol will effectively rebalance all these pools at once. It is also likely that there will be several different chains using a given token rather than just two as in the given example. A rebalancing action will define a single token moving between two chains, but multiple rebalancing actions can be bulk executed together, in the case the token ratios need to be fixed on multiple chains at once.

4.4 Investment Thesis

The blockchain ecosystem currently suffers from artificial barriers to capital efficiency, where Layer 1 chains compete to accumulate and trap liquidity (TVL) regardless of actual utility. This creates market inefficiencies that both limit growth and present opportunities. Superset's economic framework:

1. **Short-term**: Captures value from current market inefficiencies.
2. **Medium-term**: Facilitates natural evolution toward capital efficiency.
3. **Long-term**: Builds infrastructure for optimized cross-chain liquidity.

4.5 Economic Architecture

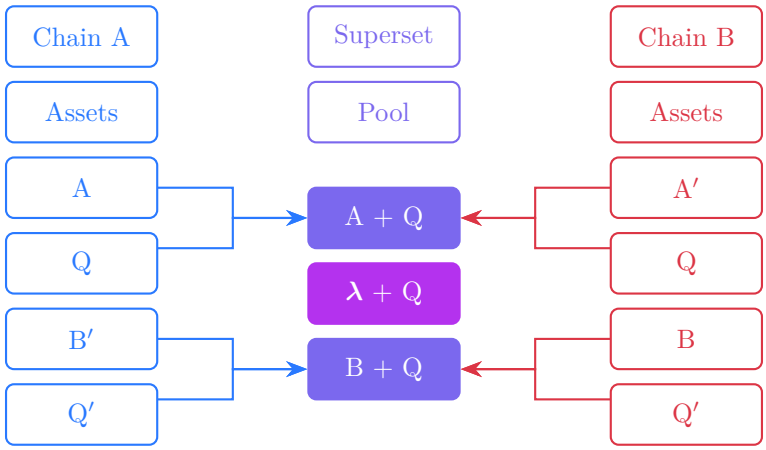
At its core, Superset recognizes a fundamental truth: the only sustainable source of long-term value in liquidity markets is fee generation from actual usage. The system's economic design uses token mechanics not as direct value drivers but as co-ordination mechanisms to optimize fee generation and distribution.

When a LP deploys assets into a given pool, they will receive a liquidity token representing your position in the pool. By deploying liquidity to a virtual pool, the LP will start to accrue trading fees depending on the usage of the virtual pool across all chains, but by staking the LP tokens you can earn further rewards from the usage of the liquidity on a specific chain.

4.6 Virtual Pool Structure

This LP token can then be staked in the hub contract to a specific chain. This will redirect the deployed liquidity to that chain, and will give the LP a proportional cut of the LP rewards for that chain. This mechanism ensures that the liquidity of the protocol is always efficiently managed, as users will be incentivied to stake their liquidity to whichever chains are best performing (highest tx volume) to maximize fees. The result of this is that due to MEV all the chains will have the correct amount of liquidity allocated to them and will give approximately equal incentive rewards, as users will move funds if an opportunity appears.

Figure 7: Cross-Chain Asset Pool Structure



The Virtual Pool enables truly aggregated liquidity across chains through LayerZero's OFT standard, creating a unified capital base that can be optimally allocated. When users stake their LP tokens as described above, they will start to earn superset tokens ($\boldsymbol{\lambda}$). The $\boldsymbol{\lambda}$ tokens represent ownership of the protocol, and therefore will be used for governance (adding new token protocols, chains), as well as earning a cut of the protocol revenue, generated through the protocol fee. They can start to earn from the protocol fee as soon as they stake their $\boldsymbol{\lambda}$ tokens, and will be able to vote on the protocol governance proposals. The $\boldsymbol{\lambda}$ token is an ERC20 token, and can be traded on any DEX that supports it.

The protocol fee is an additional fee taken on swaps (~0.1%), that is given to $\boldsymbol{\lambda}$ token holders. The $\boldsymbol{\lambda}$ token holders will be the team, investors, the public (through a token sale) and also is given as rewards for efficiently rebalancing LP tokens. This is how the investors in the protocol (both early VCs and LPs) realize profit from the protocol.

4.7 Bribes

The protocol will also allow users or chains to bribe the LPs to move their liquidity to a different chain. This will be done in with a plugin, that will allow users to add additional incentives to the LPs to move their liquidity to a specific chain. The plugin will be added to the hub chain, as that is where the liquidity and volume is aggregated. The nature of the bribe is up to the user, but it could be a token or a stablecoin. This is completely optional for chains who want to incentivize LPs to move their liquidity to their chain. They could achieve a similar effect by just LPing themselves to that chain, or using a market maker to do so, and not rebalancing the liquidity from that chain.

4.8 Economic Mechanism

The system acknowledges two apparent revenue streams - $\boldsymbol{\lambda}$ token incentives and LP fees - but recognizes these as a single unified value flow.

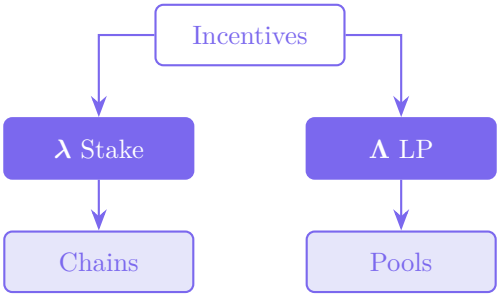
LP fees represent the 'True' revenue that the system is designed to maximize, while the $\boldsymbol{\lambda}$ token provides proportional dynamic access to this (future) revenue stream, based on contribution to system utility over time.

This creates an elegant alignment where:

- Token value derives from optimization of extraneous asset flows
- Incentives naturally balance between staking and liquidity provision
- System utility directly impacts reward distribution
- Market participants optimize for long-term fee generation

4.9 Self-Balancing Mechanisms

Figure 8: General Incentive Waterfall



The economic model introduces sophisticated yet natural balancing mechanisms:

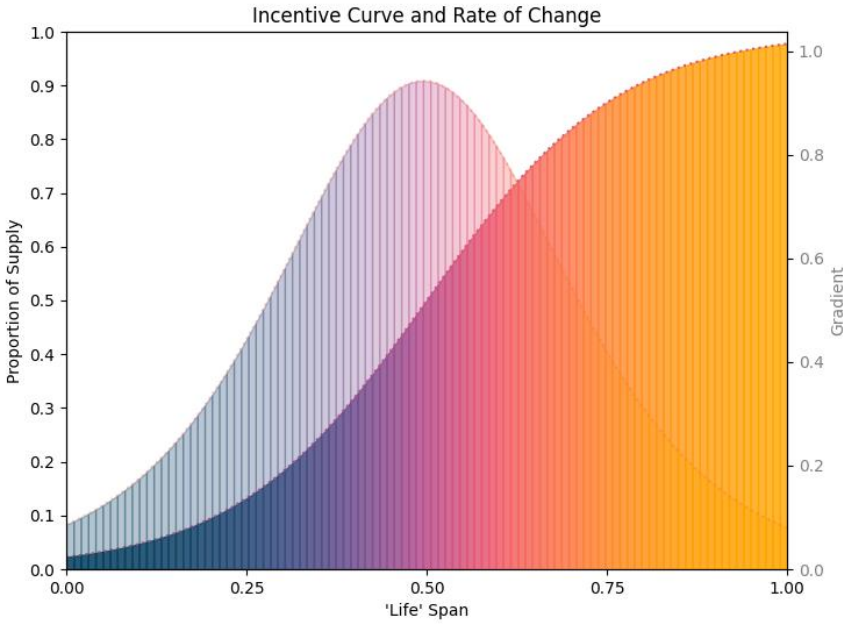
1. **Utility-Driven Incentives**: Rewards automatically adjust based on absolute system utility and the balance between staking and liquidity.
2. **Stake-Liquidity Balance**: Inverse incentive relationships maintain equilibrium - excessive staking increases rewards for liquidity provision and vice versa.
3. **Pool Optimization**: $\boldsymbol{\lambda}$ holders can direct incentives to specific pools, creating a natural mechanism for liquidity optimization.

5 Token Model & Distribution

The λ token model consists of a fixed supply of 1 Billion tokens, distributed through a carefully designed issuance schedule that mirrors expected market evolution.

5.1 Distribution Schedule

Figure 9: Incentive Issuance

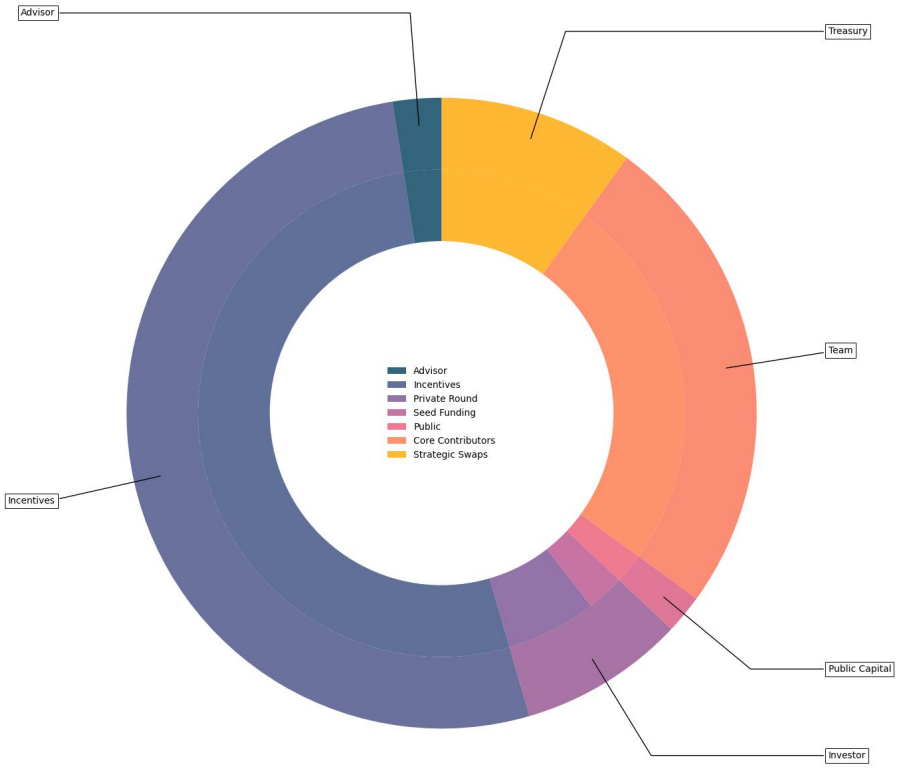


The token distribution follows a logistic curve calibrated to:

- Initial circulation of 2.25% at TGE.
- Majority allocation (52%) reserved for ecosystem incentives.
- Strategic vesting schedules for key stakeholders

5.2 Allocation Breakdown

Figure 10: Gross Allocations



Detailed allocations ensure long-term alignment:

- 52% Ecosystem Incentives.
- 25% Core Contributors (vested).
- 10% Strategic Swaps.
- 8.5% Investment Rounds.
- 2.5% Advisors.
- 2.0% Public Launch.

6 Mathematical Framework

The system's stability is mathematically constructed through:

1. **Incentive Adjustment**:
   - Utility factor: Measures and rewards system usage.
   - Balance factor: Maintains staking-liquidity equilibrium.
   - Treasury allocation: Natural emission control
2. **Fee Distribution**:
   - Dynamic LP fee share based on system utility.
   - Proportional distribution based on contribution
   - Risk-adjusted reward allocation

7 Market Evolution

The system's economic architecture captures value throughout market evolution:

Current State

- Chains competing for TVL create immediate opportunities.
- Market inefficiencies generate excess returns.
- Token mechanics capture value from temporary imbalances.

Transition Period

- Natural price discovery for liquidity value.
- Gradual optimization of capital allocation.
- Emergence of utility-driven flows.

Future State

- Efficient cross-chain liquidity distribution.
- Value capture from optimized fee generation.
- Sustainable long-term revenue streams.

8 Conclusion

Superset presents a sophisticated economic framework that solves the current inefficiencies of fragmented liquidity while building the foundation for a more efficient blockchain ecosystem. By recognizing LP fees as the fundamental value driver and using token mechanics as coordination tools rather than direct value generators, the system creates sustainable long-term value while capturing opportunities throughout market evolution.

The elegance of the design lies in its ability to maintain this alignment using adaptive techniques rather than prescriptive controls - market participants, acting in their own interest, naturally contribute to system optimization and value generation invoking the 'Invisible Hand'.