Enhancing Capital Efficiency in DeFi Lending and Liquidity Provision

Adam Smehyl and Ivan Homoliak
Brno University of Technology, Faculty of Information Technology
Email: adam.smehyl@seznam.cz, homoliak@fit.vut.cz

***Abstract*—Decentralized Finance (DeFi) continues to experience rapid growth, yet a significant portion of capital remains inefficiently utilized in overprovisioned lending reserves or inactive liquidity positions. This paper presents two extension-based improvement proposals aimed at increasing capital efficiency in DeFi protocols. The first addresses idle capital in pool-based lending by adding an allocation layer that can deploy otherwise unused liquidity into external yield-generating strategies. The second targets inactive concentrated-liquidity positions through a position-management layer that automates range migration. Both proposals are examined in terms of motivation, mechanism design, expected effects, implementation approach, and practical limitations. Evaluation results are proposal-specific: the lending analysis indicates meaningful supplier-yield uplift under selected external-yield assumptions, while the range-migration analysis focuses on active-time sensitivity and execution cost. Prototype benchmarks suggest that both mechanisms can be implemented as modular extensions, but also expose additional gas overhead and proposal-specific risks, including external-strategy dependence, recall and loss-allocation concerns, and range-policy misconfiguration.**

I. Introduction

As DeFi is connected to real-world finance, DeFi protocols and their users become attractive targets for exploiters and scammers [1], [2], [3]. This often leads users to default to protocols with a strong security track record, even at the expense of absolute yield gains. This concentrates capital in long-standing protocols that tend to prioritize stability over innovation and therefore do not always utilize capital efficiently. As a result, capital worth billions remains partly idle or earns only limited returns [4].

Within this setting, the paper focuses on two capital-efficiency limitations that arise even in correctly functioning protocol designs. Pool-based lending preserves unborrowed liquidity for withdrawals and future borrowing demand, whereas concentrated-liquidity AMMs leave positions inactive once prices exit the specified price interval. These limitations motivate two extension-based mechanisms: idle-capital allocation and automated range migration.

Asset lending is a prominent DeFi application, with users often preferring pool-based models that use utilization-based interest-rate curves to automatically adjust borrowing rates in response to changes in supply and demand [5], [6], [7]. As utilization rises, borrowing rates increase to improve yields and discourage further borrowing, and vice versa. However, these models often maintain substantial unborrowed reserves to preserve liquidity for future borrowing and supplier withdrawals, leaving part of the supplied capital without direct borrower-paid yield. This work explores a reserve-aware allocation layer that can deploy excess idle liquidity while retaining local reserves for ordinary withdrawals and borrowing demand.

In token trading, most short-term activity occurs within a relatively narrow price range, so reserving substantial liquidity for larger moves is inefficient. Providing trading liquidity over a defined range is therefore more capital-efficient, but once the price moves outside that range, the position becomes stale. This problem is especially relevant for pairs involving yield-bearing tokens, where the relative price may exhibit persistent directional drift.

Within this scope, the paper makes three main contributions. First, it studies idle-capital allocation as a reserve-aware extension pattern for pool-based lending, and evaluates it through an analytical supplier-yield model and prototype gas measurements. Second, it presents automated range migration as a user-controlled vault mechanism for concentrated-liquidity positions, including a prototype architecture and benchmark results. Third, it identifies the practical limitations of both mechanisms, including external-strategy risk, loss allocation, execution cost, and policy-configuration risk.

II. Background and Scope

A. Pool-Based Lending and Idle Reserves

Pool-based lending protocols aggregate deposits of a given asset into a shared liquidity pool from which eligible borrowers can borrow funds [8]. Within each pool, borrowing conditions are typically set by a utilization-based interest-rate model. Utilization denotes the proportion of pooled capital currently borrowed and serves as a key measure of liquidity scarcity. As utilization increases, borrowing rates typically rise to discourage additional borrowing, encourage repayment, and attract additional supply.

Because persistent full utilization would leave the pool without immediately available liquidity, pool-based lending protocols typically operate around a managed utilization trade-off. Supplied capital is effectively divided into two functional categories: capital lent out to borrowers, which generates borrower-paid interest, and liquid inactive reserves that remain unborrowed and serve as a buffer for normal withdrawals and additional borrowing demand. The borrowed portion of capital is determined primarily by borrower behavior, while the protocol can influence that behavior only indirectly through utilization-based interest-rate curves.

The design objective of the first proposal is to improve supplier-side capital productivity without forcing the lending pool to reduce its internal liquidity buffer. Compared with simply lowering reserve overprovisioning, the mechanism should preserve a larger share of immediately available liquidity for borrowing demand and withdrawals. Its objective is therefore not to maximize nominal APY, but to improve average supplier yield while preserving a conservative liquidity and risk profile.

B. Concentrated Liquidity and Drift-Prone Pairs

Liquidity providers (LPs) supply assets to on-chain pools against which traders perform swaps. In a full-range AMM, a substantial share of liquidity may be positioned far from the price range in which trading activity actually occurs, reducing overall capital efficiency. A concentrated liquidity position allows LPs to allocate capital only within a selected price interval defined by lower and upper bounds [9]. Within this interval, the position actively facilitates trading, whereas outside it is inactive and consists entirely of a single asset.

The choice of the price interval directly determines the balance between capital efficiency and active trading time: narrower ranges concentrate more capital around the expected trading region and can improve fee generation, but even modest price movement may push the position out of range. Wider ranges increase the likelihood that the position remains active, at the expense of less efficient capital deployment.

The proposed range-migration mechanism is primarily motivated by trading pairs involving yield-bearing tokens (YBTs). A YBT represents a tokenized claim on capital deployed in yield-generating strategies. Because the YBT is economically linked to its underlying asset, the pair may be less volatile than uncorrelated pairs, making narrow liquidity ranges more defensible from a risk perspective. However, the YBT's gradual appreciation introduces a persistent directional component into the relative price. As a result, the same drift that makes the pair suitable for narrow liquidity provision also shortens the time for which each narrow interval remains productive.

III. Idle Capital Allocation in Pool-Based Lending

This section introduces the idle-capital allocation mechanism as a proposed capital-efficiency enhancement for pool-based lending protocols. The discussion proceeds from positioning to mechanism design, analytical evaluation, prototype implementation, and practical risk boundaries.

A. Related Approaches and Positioning

The proposed mechanism should not be understood as claiming that redeployment of inactive capital is itself a new idea. Euler and Fluid both use liquidity-layer designs that redirect idle lending capital into trading liquidity positions on their respective DEX platforms. In these systems, the same liquidity may support borrowing, collateralization, and market making [10], [11].

These approaches are constrained by their dependence on suitable trading pairs. Because idle lending liquidity is deployed as trading liquidity, uncorrelated pairs may introduce impermanent-loss risk. The mechanism is therefore most naturally limited to correlated assets, while assets without an appropriate paired market cannot benefit from the same capital-efficiency technique.

A different form of reusing idle capital appears in Loopscale on Solana. Although Loopscale is an orderbook-based lending protocol rather than a pool-based one, it illustrates the broader idea of reallocating lender capital to a separate yield venue while it is not yet matched with a borrower [12]. Aave's proposed V4 Reinvestment Module is especially relevant because it treats idle liquidity as a protocol-level capital-efficiency issue within a large pool-based lending protocol. The module addresses excess liquidity by deploying it into governance-approved low-risk strategies and recalling it as utilization rises [13].

The paper does not claim novelty in the abstract idea of earning yield on idle liquidity. Rather, it uses existing approaches as motivation for studying a specific pool-based design pattern: a modular allocation layer that preserves the lending pool's ordinary behavior while introducing reserve-aware external deployment and recall. The evaluation focuses on the economic effect, implementation overhead, and additional risk boundaries of this specific instantiation.

B. Mechanism Design

The proposed mechanism extends, rather than replaces, the pool-based lending architecture with a capital-allocation layer. Otherwise idle capital is temporarily deployed into an external yield-generating strategy, such as a sufficiently liquid lending venue, where it can earn supplemental return while remaining recallable when needed. Because this introduces additional complexity and external dependency, the mechanism should remain active only when the expected return justifies the added risk.

As discussed above, pool-based lending protocols set borrowing costs through utilization-based interest-rate curves. A typical pool at 65% utilization is shown in Fig. 1: 65% of deposited capital is actively borrowed, while the remaining 35% remains idle and generates no direct yield. The purpose of the proposed mechanism is to reallocate this idle portion into an external yield-generating strategy. As illustrated in Fig. 2, this would allow most of the protocol's capital to remain

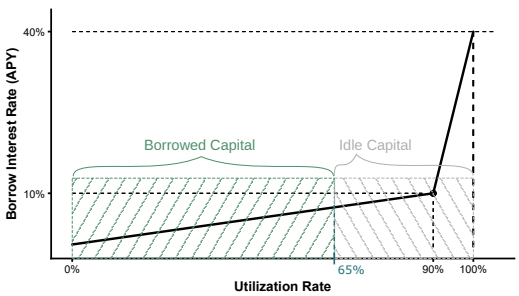

Fig. 1. Illustration of capital split in a lending pool at 65% utilization, where borrowed capital earns interest while the remaining share stays idle.

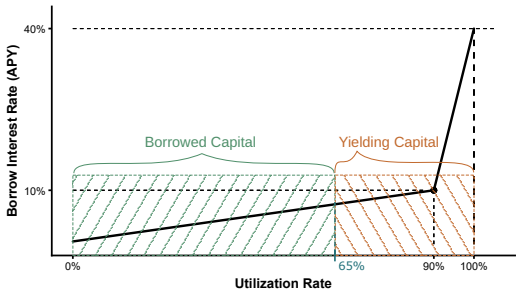

Fig. 2. Reallocation of idle pool capital into an external yield strategy under the same 65% utilization scenario.

yield-producing, aside from any internal reserve retained for immediate liquidity needs.

For the purposes of this mechanism, pool utilization continues to measure the share of capital borrowed through the protocol itself. The non-borrowed portion, therefore, includes both locally idle liquidity and capital temporarily deployed in a liquid external strategy. This distinction matters because the mechanism should not redefine utilization or disrupt the base interest-rate model.

Overprovisioning in pool-based lending has both costs and benefits. While it leaves part of the supplied capital idle, thereby reducing capital efficiency, it also preserves immediate protocol liquidity. Internal reserves are needed to meet ordinary borrowing and withdrawal demands, so they cannot be treated as entirely available for external deployment. If the protocol is to preserve liquidity while reallocating idle capital, the external strategy should therefore be limited to venues with strong withdrawal capacity and low expected exit friction.

This implies a reserve-aware operating model. A portion of capital should remain inside the protocol as an internal liquidity buffer, while the remaining idle share may be deployed externally. Under this design, ordinary borrowing demand and smaller withdrawals are first served from internal reserves, whereas larger shortfalls trigger a recall from the external strategy. Because such a recall introduces additional external calls into otherwise common deposit and withdrawal flows, it may increase transaction costs. This further supports satisfying smaller requests locally whenever possible.

The resulting architecture can be understood as a separation between the lending protocol core and a dedicated allocation extension, as shown in Fig. 3. Users continue to interact with the lending protocol core, while the extension controls the allocation and recall of eligible idle capital. The administrative component in the figure represents protocol-level control over extension configuration and replacement, rather than a requirement for any specific governance implementation.

This proposal intentionally leaves the external yield strategy abstract. The most suitable strategy depends on protocol-specific factors such as capital scale, user behavior, utilization patterns, governance preferences, and risk tolerance. A conservative implementation should prioritize venues with instant or near-instant withdrawal capacity to ensure that external deployment does not materially impair the protocol's ability to meet ordinary borrowing and withdrawal demand. Tokenized access to short-duration U.S. Treasury exposure may also be a plausible candidate class, for example, through projects such as Nest [14]; however, the tokenized wrapper adds separate operational, custody, redemption, regulatory, and liquidity assumptions.

C. Yield Model and Analytical Evaluation

To evaluate the efficiency gains of the proposed mechanism, it is useful to begin with the baseline relation between borrower-paid interest and supplier yield. The purpose of the following formulation is not to provide a fully optimized quantitative model, but rather to clarify the economic logic of the standard pool-based lending design.

For the numerical examples and figures below, the borrow rate is assumed to follow a stylized piecewise-linear utilization curve: 0% APY at zero utilization, 10% APY at 90% utilization, and 40% APY at full utilization. This assumption is used only as an analytical reference model:

$$r_b(U) = \begin{cases} \frac{10}{0.9}U, & U \in [0, 0.9], \\ 300U - 260, & U \in (0.9, 1], \end{cases} \tag{1}$$

where $U \in [0, 1]$ denotes the pool-utilization ratio and rates are expressed as annualized percentages.

While borrow rates in pool-based lending often rise approximately linearly up to the optimal-utilization threshold, the supplier-side yield follows a different relation:

$$r_s(U) = U r_b(U)(1 - \rho), \tag{2}$$

where $r_s(U)$ denotes the supply rate, $r_b(U)$ the borrow rate as a function of utilization, and $\rho \in [0, 1]$ the reserve factor retained by the protocol. The reserve factor represents the share of borrower-paid interest withheld for protocol funding or reserve accumulation. For simplicity, the following examples assume $\rho = 0$.

The proposed mechanism extends the baseline pool model by adding an externally sourced yield component to the supplier-side return. Because this external yield applies only to the non-borrowed share of capital, its contribution declines as pool utilization rises and disappears once the entire capital base is borrowed or otherwise reserved internally. At the supplier-accounting level, the external strategy is internalized by the pool. Suppliers continue to hold a claim on the lending pool itself, while income from borrower interest and externally

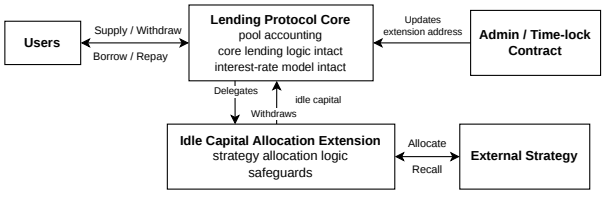

Fig. 3. Extension-based architecture for idle-capital allocation, preserving core lending logic while delegating external deployment to a separate module.

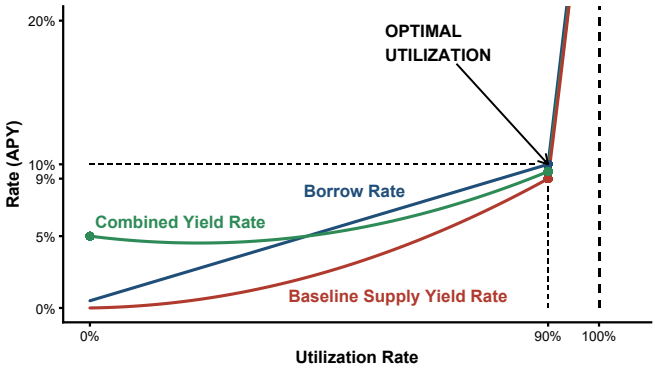

Fig. 4. Comparison of borrow rate, baseline supply yield, and combined supplier yield, showing how external allocation of idle capital can lift supplier returns above the pool-only baseline.

deployed idle capital is aggregated and distributed pro rata as pool yield.

By incorporating the external-yield component, the combined supplier yield can be written as

$$r_c(U) = U r_b(U)(1-\rho) + (1-U-\lambda) r_{\text{ext}}, \quad 0 \le \lambda \le 1-U, \tag{3}$$

where $r_c(U)$ denotes the combined yield rate, r_{ext} the external yield rate earned on otherwise idle capital, and λ the fraction of supplied capital retained as liquid reserve and therefore not allocated to the external yield source. Under the special case $\lambda = 0$, the full idle share is externally deployed. This formulation captures the fact that the external-yield contribution decreases with utilization and vanishes at full utilization.

Fig. 4 illustrates the resulting change in supplier yield under an external-yield assumption of 5% APY by comparing the borrow rate, baseline supply yield rate, and combined yield rate. Under the baseline pool-based lending model, the supply yield at 20% utilization is approximately 0.44% APY. Under an illustrative external-yield assumption of 5%, the mechanism adds approximately 4.0 percentage points of APY at the same utilization level, raising the combined supplier yield to approximately 4.44% APY.

Tab. I summarizes this added yield across selected utilization levels and external-yield assumptions. The values express the increase on top of the baseline lending return, rather than the total resulting APY. The improvement is more pronounced at lower utilization levels, where a larger share of supplied capital is typically idle and can therefore be reallocated to an external yield source. As utilization rises, more capital is already deployed through native borrowing activity and less remains available for external allocation.

TABLE I
ABSOLUTE INCREASE IN SUPPLIER APY, IN PERCENTAGE POINTS, RESULTING FROM EXTERNAL DEPLOYMENT OF IDLE POOL CAPITAL UNDER SELECTED UTILIZATION LEVELS AND EXTERNAL STRATEGY YIELDS.

Ext. APY	Pool utilization								
	10%	20%	30%	40%	50%	60%	70%	80%	90%
2%	1.8	1.6	1.4	1.2	1.0	0.8	0.6	0.4	0.2
3%	2.7	2.4	2.1	1.8	1.5	1.2	0.9	0.6	0.3
4%	3.6	3.2	2.8	2.4	2.0	1.6	1.2	0.8	0.4
5%	4.5	4.0	3.5	3.0	2.5	2.0	1.5	1.0	0.5
7%	6.3	5.6	4.9	4.2	3.5	2.8	2.1	1.4	0.7
10%	9.0	8.0	7.0	6.0	5.0	4.0	3.0	2.0	1.0

The values in Tab. I represent gross yield effects and should not be interpreted as sufficient deployment criteria. In practice, the extension should allocate capital only when the expected net improvement remains positive after accounting for strategy fees, additional transaction costs, operational monitoring, liquidity risk, and expected loss from the external venue. This implies a minimum external-yield threshold: low external APY may be economically unattractive even when it produces a positive gross yield uplift, because the added return may not compensate suppliers for the additional complexity and risk introduced by the allocation layer.

D. Prototype Validation and Gas Evaluation

To examine feasibility beyond the analytical yield model, the proposed mechanism was implemented as a prototype. The prototype should be understood as an experimental validation environment rather than a production-ready lending protocol, a public deployment, or a complete fork of an existing system such as Aave, Euler, or Morpho. Its purpose was to test whether idle-capital allocation can be added to a simplified pool-based lending architecture while preserving the core user-facing lending behavior.

At a high level, the prototype recreates enough lending-pool functionality to exercise the allocation mechanism in realistic transaction flows. It separates ordinary lending behavior from a replaceable allocation layer that can retain local reserves, park excess idle assets, delegate eligible capital to a strategy adapter, and recall allocated funds when liquidity is needed. This separation was used to evaluate whether the mechanism can remain extension-based rather than requiring a redesign of the lending pool itself.

The prototype served two main evaluation purposes. First, it validated the behavioral integration of allocation and recall paths across user-facing operations such as supplying, withdrawing, borrowing, and repaying. Second, it made it possible to estimate the gas overhead introduced by the additional allocation logic and token transfers. Gas usage was therefore benchmarked for successful user-facing transactions across three configurations: a baseline lending pool without the allocation extension, a parked-allocation configuration in which the extension is active but does not delegate into an external strategy, and a strategy-allocation configuration in which eligible idle liquidity is delegated into the mock strategy adapter.

The benchmark confirmed the routing distinction described above. The allocation layer does not add gas to withdraw or borrow transactions when pool-local liquidity is sufficient; those unchanged cases are omitted from Tab. II. The additional cost appears when liquidity must be recalled from the allocation layer, or when returned liquidity is reallocated during

TABLE II
GAS USAGE COMPARISON FOR SELECTED USER-FACING IDLE-ALLOCATION PROTOTYPE TRANSACTIONS. PERCENTAGES REPORT THE INCREASE RELATIVE TO THE BASELINE CONFIGURATION.

Operation	Baseline	Extension	Ext. + strat.
Supply	57,368	102,628 (+79%)	181,687 (+217%)
Withdraw recall	103,153	126,643 (+23%)	141,975 (+38%)
Borrow recall	166,461	189,951 (+14%)	205,283 (+23%)
Repay	63,710	110,970 (+74%)	190,029 (+198%)

supply and repayment flows. In relative terms, these increases are significant, especially when the extension delegates assets into the strategy adapter. In absolute gas terms, however, the measured transactions remain within a range that is not unusually high for DeFi interactions. The strategy-allocation figures should still be treated as a lower-bound estimate because the prototype uses a deliberately minimal mock strategy adapter, and a real protocol integration would likely be larger and more costly.

E. Risk Boundaries and Practical Considerations

The main risk introduced by the mechanism is dependence on external venues. Once idle liquidity is deployed outside the lending pool, the protocol's ability to satisfy withdrawals or new borrowing demand depends partly on the liquidity and solvency of the external strategy. Strategy selection, therefore, becomes an operational control problem rather than a simple implementation detail.

A production protocol would need governance-defined rules for admissible venues, allocation caps, recall settings, strategy replacement, monitoring, and emergency pauses. It would also need explicit procedures for cases in which liquidity cannot be recalled as expected. If an external venue becomes illiquid or impaired, losses may propagate back into the lending pool. This is especially relevant during periods of elevated withdrawal demand, where delayed exits may coincide with reduced external liquidity.

Under the aggregated-pool model considered here, such losses would be distributed proportionally across supplier claims, potentially mitigated by reserves accumulated through protocol fees. A more conservative alternative would be to divide suppliers into opt-in and non-opt-in tranches. Only opt-in suppliers would then be exposed to external-strategy losses, while non-opt-in capital would remain inside the base protocol. This would provide a clearer risk boundary, but it would also require more complex accounting, liquidity management, and user disclosure.

IV. AUTOMATED RANGE MIGRATION FOR CONCENTRATED LIQUIDITY

A distinct capital-efficiency problem arises in concentrated-liquidity provision. Unlike the idle-capital allocation discussed above, the issue here is not the deployment of inactive reserves in lending pools, but the loss of fee-earning exposure once market activity moves outside a user-defined price range.

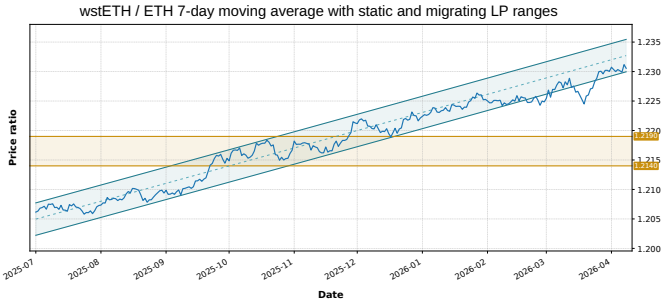

Fig. 5. Price-ratio development of the wstETH/ETH pair with example static and migrating liquidity ranges, illustrating how persistent directional drift can push narrow positions out of range.

A. Problem Framing and Related Approaches

For higher capital efficiency, trading liquidity is usually provided within a relatively narrow price range, where it is available in full. The drawback is that once market activity moves outside the chosen range, the position becomes stale and no longer earns trading fees. Yield-bearing tokens that persistently appreciate against the underlying asset make this more pronounced. Their relative stability attracts significant liquidity, which must use narrower ranges to maintain attractive returns.

Fig. 5 illustrates this problem using the wstETH/ETH pair. The underlying price path is based on historical market data for the shown period, while the highlighted liquidity ranges are synthetic overlays used to illustrate the position-management problem. A typical static range can gradually fall behind the prevailing price movement, whereas an automatically migrating range preserves active exposure by shifting with the drift. The depicted migration is aligned with an annualized upward drift of three percentage points.

Since the introduction of concentrated liquidity, several projects have developed automated liquidity-management systems for concentrated-liquidity AMMs. Examples include Gamma LP Vaults, which provide automated rebalancing and multi-position liquidity strategies [15]; Arrakis Pro, which focuses on issuer and treasury-oriented liquidity management [16]; and Revert, which offers automation tools such as auto-compounding and auto-range management for individual Uniswap positions [17]. These examples show that automated management of concentrated liquidity is an established design space rather than a wholly new problem.

The proposed mechanism belongs to the same automated liquidity-management design space, but targets a narrower problem: directional range migration for drift-prone pairs through user-configured dedicated vault instances, rather than pooled strategy vaults or full re-centering around the current spot price. The proposed design is closer to user-owned automation than to a pooled managed strategy. The intended user-facing interface is a vault-creator application that allows the user to specify the asset pair, range width, position granularity, and migration policy before deploying a dedicated vault instance. Administrative control remains with the deployer.

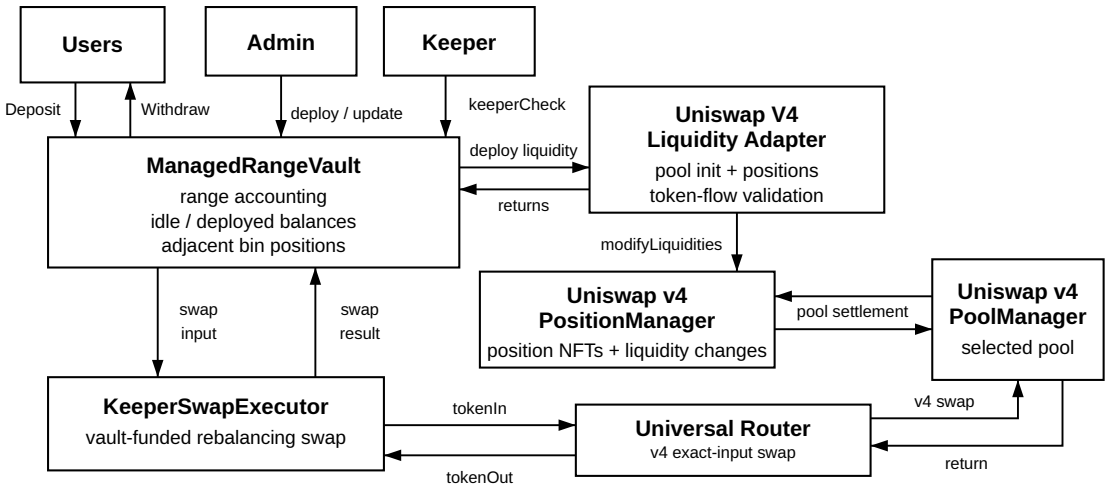

Fig. 6. Capital-flow architecture of the managed range vault, separating user deposits and withdrawals from keeper-triggered range migration and adapter-mediated Uniswap v4 interactions.

The architectural changes introduced by Uniswap v4 make this form of user-owned automation more practical. Its singleton architecture reduces some of the overhead associated with liquidity management, while hooks provide a programmable integration point for pool-specific logic [18], [19]. This does not remove trust assumptions entirely. The vault code, adapter logic, behavior configuration, and any emergency permissions still require careful review. However, the trust boundary is narrower and more explicit: the user accepts the behavior of a specific deployed vault instance and its configured operators, rather than depositing into a shared strategy controlled by an external project.

B. Mechanism Design and Vault Architecture

The proposed mechanism introduces automated range migration for concentrated-liquidity positions exposed to directional price drift. Its purpose is to adjust the maintained price interval so that liquidity can remain productive within the range without repeated manual repositioning to newly selected bounds. This mechanism should be distinguished from full position re-centering or periodic portfolio rebalancing. The objective is not to repeatedly reconstruct the entire position around the current spot price into a fresh symmetric asset composition. Instead, the intended design leaves the in-range portion of the position unchanged and migrates only liquidity that has drifted beyond the maintained price interval.

The mechanism was implemented as a prototype around a dedicated vault that owns and manages the liquidity positions for its deployer. The prototype uses Uniswap v4's programmable integration points to separate three responsibilities: user deposit and withdrawal accounting, keeper-triggered range migration, and protocol-specific liquidity operations. Its capital flow is summarized in Fig. 6, which should be read as a prototype architecture rather than a prescribed production design. Deposits and withdrawals remain distinct from maintenance actions, while migration can update the maintained range, optionally rebalance assets before redeployment, and route AMM-specific operations through a separate integration layer.

Within this architecture, the maintained range is represented through adjacent vault-side micro-positions, or bins. Each bin corresponds to an independent concentrated-liquidity position owned by the vault, rather than to a native Uniswap accounting primitive. As Fig. 7 shows, migration removes trailing liquidity, swaps withdrawn assets as needed, shifts the remaining position records, and resupplies liquidity on the advancing side. This makes the mechanism a means of preserving the benefits of narrow ranges when price dynamics exhibit a sufficiently stable directional component.

At the implementation level, the prototype keeps Uniswap-specific interactions separate from vault accounting and uses helper logic to derive tick spacing, align ranges, split ranges into bins, and compute proportional deposit shares. It can also perform a vault-funded swap before redeploying liquidity into the new edge position. These details are included to explain the source of the benchmarked gas overhead, not to prescribe a particular component structure.

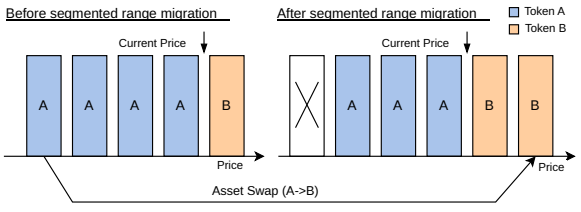

Fig. 7. Segmented range migration in a concentrated-liquidity position, showing withdrawal of trailing liquidity and reallocation on the advancing side.

The range migration mechanism is intended to operate according to predefined policy rules rather than discretionary intervention. In basic form, those rules may be time-based, with the admissible price range shifting upward according to preset parameters, or reference-based, with the shift tied to an oracle-reported redemption rate, NAV, or a similar valuation anchor. In the prototype experiments, the tested policy is a preset time-based drift specified through annualized or hourly movement parameters. This matches the intended YBT setting, where the maintained range is expected to advance with persistent appreciation rather than oscillate around a stable price level.

C. Evaluation Scope and Prototype Benchmarks

Because concentrated-liquidity returns depend on price movements, trading volume, and available liquidity, the mechanism cannot be evaluated with a single static APY estimate. The relevant comparison is therefore not against an abstract maximum APY, but against practical alternatives such as static ranges or manual repositioning. Range migration improves the outcome only when the additional fee-generating time outweighs the costs of migration, including keeper execution, slippage, and any losses introduced by repositioning.

Fig. 8 illustrates the active-time component of the range-migration problem. The figure is not a market backtest or a prototype result; it shows how average realized fee APY declines as the share of time spent in range decreases under selected illustrative in-range APY assumptions. It therefore motivates why active-time improvement matters, while leaving the actual profitability question to a position-specific comparison against migration costs.

The prototype benchmarks should be read within that limited scope. They do not establish that automated migration will outperform manual or alert-assisted repositioning under all market conditions. Instead, they show the execution overhead of maintaining and migrating a vault-managed range, which can then be weighed against the expected benefit of reducing out-of-range time for a drift-prone pair. Automated range migration is therefore best framed as a mechanism for reducing out-of-range latency under predictable drift, not as a general guarantee of higher APY.

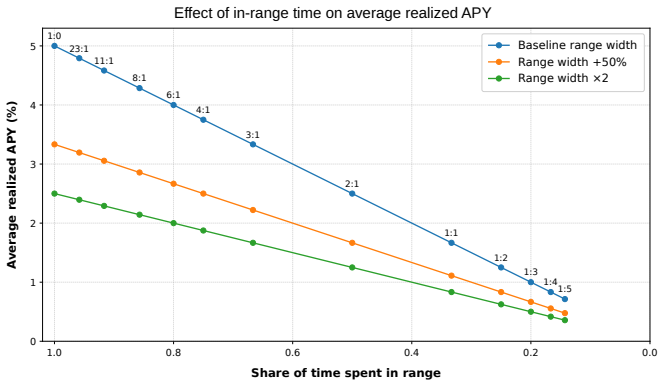

Fig. 8. Conceptual sensitivity of realized fee APY to active time in range. The curves use illustrative in-range APY assumptions and do not report measured market performance.

TABLE III
RANGE-VAULT GAS BENCHMARK RESULTS BY CONFIGURED VAULT BIN COUNT, ROUNDED TO THE NEAREST THOUSAND.

Path	1	4	8	12	16	20
Direct init + mint	520k	–	–	–	–	–
Vault deployment	5,527k	5,633k	5,775k	5,917k	6,059k	6,201k
Liquidity update	652k	1,661k	2,975k	4,294k	5,614k	6,934k
Migration, no swap	486k	595k	706k	816k	928k	1,039k
Migration with swap	503k	779k	858k	967k	1,079k	1,190k

Gas usage was also measured for the managed range-vault prototype on Unichain mainnet. The benchmark covers deployment, direct Uniswap position creation, vault-mediated liquidity updates, and keeper-triggered one-bin migration. These measurements should be interpreted as prototype-level indicators rather than universal gas costs, since the final cost depends on the network, pool configuration, bin count, and whether migration includes a swap.

Tab. III highlights the setup and recurring management costs. Deploying a vault instance consumed approximately 5.5M–6.2M gas, but this should be interpreted as a one-time setup cost for a dedicated vault instance rather than an ordinary recurring transaction. The most relevant recurring maintenance paths are keeper-triggered one-bin migrations, whose frequency is policy-dependent rather than continuous. Under a slow-drift YBT pair such as the wstETH/ETH example in Fig. 5, a 3% annualized movement policy reaches the next migration threshold only after the configured bin width has been traversed; for sufficiently coarse bins, this can correspond to intervals on the order of weeks. Full liquidity updates remain more expensive, costing approximately 6.9M gas at 20 bins, illustrating why range granularity must be treated as an economic parameter rather than merely a strategic preference.

D. Execution Model and Configuration Risks

In principle, range migration could be integrated directly into the pool execution path through a Uniswap v4 hook. Selected pool interactions, such as swaps, could then evaluate whether the maintained range should move and initiate the corresponding migration. This would reduce reliance on a separate maintenance schedule, but it would also create a cost-allocation problem: repositioning concentrated-liquidity positions is gas-intensive, especially when the operation touches multiple micro-positions or requires asset conversion. If performed during a hook-triggered swap, the additional execution cost would be imposed on the transaction that activates the hook.

The prototype therefore uses a keeper-assisted execution model. The vault exposes one path to check whether the configured movement threshold has been reached and another to execute the migration. Since smart contracts cannot initiate transactions on their own, the triggering actor must be external:

an externally operated account, an automation network, or another off-chain service. This model is less tightly integrated than hook-native automation, but it separates the expensive maintenance operation from ordinary swap execution.

Automated range migration also introduces risks that are not purely technical. The mechanism improves capital efficiency only if the maintained range moves consistently with the trading pair's actual development. If the configured drift is too aggressive, too slow, or based on an inaccurate reference value, the mechanism may reduce realized performance rather than improve it. Reference-based policies may improve responsiveness, but they also introduce assumptions about data freshness, manipulation resistance, and the relationship between the reference value and the tradable market price.

E. Deployment Limits and Future Work

The implementation shows that range granularity is constrained by execution costs. A vault that divides the maintained range into many micro-positions can express a more detailed liquidity structure, but it also touches more positions during deployment and maintenance. In the tested prototype, 20 micro-positions represented a practical upper bound under the selected conditions. This should not be interpreted as a universal technical maximum; rather, it shows that any production design must treat range granularity as an economic and operational parameter.

Future work should therefore focus on reducing maintenance cost and improving policy design. One possible optimization is to initialize position slots in advance, so that migration can move liquidity into an already prepared edge position instead of creating new position state during the most time-sensitive part of execution. Another direction is a more expressive liquidity-distribution model, including uniform, linear, bid-ask-style, or otherwise weighted structures. More advanced movement policies could track net asset value, redemption value, oracle-reported reference rates, realized market behavior, or historical trend estimates, but these additions would also increase model risk, oracle dependence, and configuration burden.

V. Conclusion

Both proposals presented in this work are intentionally extension-based rather than replacement-based. That makes them attractive from an integration perspective, but it also means that feasibility depends on the constraints of the integrating protocol. For idle capital allocation, suitable external strategies must align with liquidity requirements, governance constraints, and risk tolerance. For automated range migration, realized benefits depend on whether the underlying market actually exhibits sufficiently persistent directional drift to justify repeated repositioning.

The prototypes confirmed feasibility while exposing practical limits. Idle-capital allocation adds external dependency, recall risk, and loss-allocation questions. Automated range migration adds execution overhead, policy-configuration risk, and position-management complexity. The main conclusion is therefore not that either mechanism is universally optimal, but that DeFi protocols still leave room for meaningful capital-efficiency improvements that can be approached through modular upgrades. Automation or yield gains should remain secondary to transparency, auditability, economic justification, governance, risk boundaries, dependencies, and loss-allocation rules.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT-5.5 and Grammarly for rephrasing, sentence polishing, and formatting assistance. After using these tools/services, the author(s) reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] De.Fi, "De.fi rekt database," De.Fi, 2026, accessed: 2026-01-06. [Online]. Available: https://de.fi/rekt-database

[2] SlowMist, "Slowmist hacked – blockchain hack & exploit database," SlowMist Zone, 2026, accessed: 2026-01-06. [Online]. Available: https://hacked.slowmist.io/

[3] T. Khaitan, "Defi lender euler suffers $200m exploit," The Defiant, Mar. 2023, accessed: 2026-03-31. [Online]. Available: https://thedefiant.io/news/defi/euler-200m-exploit

[4] DefiLlama, "Defi yields dashboard," DefiLlama, 2026, accessed: 2026-04-01. [Online]. Available: https://defillama.com/yields

[5] Aave, "Aave protocol documentation," Aave Documentation, 2025, accessed: 2026-04-01. [Online]. Available: https://aave.com/docs

[6] Euler Labs, "Introduction," Euler Documentation, 2025, accessed: 2026-04-01. [Online]. Available: https://docs.euler.finance/introduction/

[7] J. Xu and N. Vadgama, "From banks to DeFi: The evolution of the lending market," in *Enabling the Internet of Value: How Blockchain Connects Global Businesses*, ser. Future of Business and Finance, N. Vadgama, J. Xu, and P. Tasca, Eds. Cham: Springer International Publishing, Jan. 2022, pp. 53–66, also available as arXiv:2104.00970. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-78184-2_6

[8] M. Bartoletti and E. Lipparini, "A theory of lending protocols in DeFi," online, arXiv, Preprint arXiv:2506.15295, Jun. 2025, [cs.GT]. [Online]. Available: https://arxiv.org/abs/2506.15295

[9] Uniswap Labs, "Uniswap documentation," Uniswap Documentation, 2025, accessed: 2026-04-01. [Online]. Available: https://docs.uniswap.org/

[10] Instadapp, "Introducing fluid dex!" Instadapp Blog, Jun. 2024, accessed: 2026-05-07. [Online]. Available: https://blog.instadapp.io/fluid-dex/

[11] Euler Labs, "Eulerswap," Euler Documentation, 2025, accessed: 2026-05-07. [Online]. Available: https://docs.euler.finance/concepts/advanced/euler-swap

[12] Loopscale, "Advanced lending," Loopscale Documentation, 2025, accessed: 2026-05-07. [Online]. Available: https://docs.loopscale.com/using-loopscale/advanced-lend#lend-idle-capital

[13] Aave Labs, "Aave v4's reinvestment module," Aave, Mar. 2026, accessed: 2026-05-07. [Online]. Available: https://aave.com/blog/aave-v4-reinvestment-module

[14] Nest Credit, "Nest credit," Nest Credit, 2026, accessed: 2026-05-07. [Online]. Available: https://www.nest.credit/

[15] Gamma, "Lp vaults: Introduction," Gamma Documentation, 2025, accessed: 2026-05-07. [Online]. Available: https://docs.gamma.xyz/gamma/lp-vaults/introduction

[16] Arrakis Finance, "Arrakis pro," Arrakis Documentation, 2025, accessed: 2026-05-07. [Online]. Available: https://docs.arrakis.finance/text/introduction/arrakisPro.html

[17] Revert Finance, "Auto-range," Revert Documentation, 2026, accessed: 2026-05-16. [Online]. Available: https://docs.revert.finance/revert/auto-range

[18] Uniswap Labs, "Architecture," Uniswap Developers Documentation, 2025, accessed: 2026-05-07. [Online]. Available: https://developers.uniswap.org/docs/protocols/v4/concepts/architecture

[19] ——, “Uniswap v4 overview,” Uniswap Documentation, 2025, accessed: 2026-04-01. [Online]. Available: https://docs.uniswap.org/contracts/v4/overview