



BLAZEPHOENIX PROTOCOL · TECHNICAL NOTE

BlazePhoenix

Related Work and Prior-Art Statement

A supplement to the BlazePhoenix Technical Whitepaper, Version 2.3

TECHNICAL NOTE — VERSION 1.0 · SUPPLEMENT TO THE WHITEPAPER V2.3

Written by [Mitra](#)

Version	1.0
Date	2026-10-19
Author	Mitra
DOI	10.5281/zenodo.23460997
Supplement to	doi:10.5281/zenodo.23084090
License	CC BY 4.0 (text) · BUSL-1.1 (code)
Web	https://blazephoenix.xyz

Contents

Abstract.....	3
1. Scope and method.....	3
2. The constructions.....	4
Claim 1 — In-transaction probing and depth-weighted allocation (I-10)	4
Claim 2 — The Iron-Law floor Φ (I-11).....	4
Claim 3 — Quote and execution as one computation (I-1).....	5
Claim 4 — Measurement at every seam (I-12, I-13).....	5
Claim 5 — The Monoslot (I-8)	6
Claim 6 — The self-healing registry and the vitality field Ψ (I-9).....	6
Claim 7 — Derive, then prove: on-chain venue discovery (I-5, I-6)	6
Claim 8 — Hooks screened by arithmetic, fees read as executed (I-6)	7
Claim 9 — The split allocator and its believability band (I-10).....	7
Claim 10 — The Surplus Rule (I-14).....	8
Claim 11 — The Eightfold Dispatcher Ω (I-4).....	8
Claim 12 — Fee-on-transfer assets, routed where natural.....	8
Claim 13 — Uniswap V4 pool-only output computed in-frame, raising the leg bound (I-6).....	8
Claim 14 — Named assurance practices (I-18).....	9
3. Derivation: when a depth-proportional split is optimal	9
4. Evidence of authorship and dates	10
5. Limitations.....	11
6. Prior-art statement.....	11
7. How to cite	11
8. Licence	12
Acknowledgements.....	12
References	12
Appendix A — Quote-bound hook routing (disclosed construction, not implemented)	13

Supplement to: Mitra, *BlazePhoenix Technical Whitepaper: an on-chain liquidity aggregator whose quote and execution are one computation*, Version 2.3, 2026. doi:10.5281/zenodo.23084091 (all versions: doi:10.5281/zenodo.23084090).

Reference source: BlazePhoenix-Dex, commit `982c1de0578c88b92d224d393b3cb5a4599e7416` (2026-10-12). The five source files at that commit are deposited with this record.

Abstract

BlazePhoenix-Dex is an on-chain liquidity aggregator for EVM chains. It discovers venues, plans a route, sets an output floor and executes the route in the same transaction, with no off-chain price server. Its whitepaper names the constructions that make this possible. This supplement does three things the whitepaper does not. First, it places each construction next to the closest prior work we found: published routing theory, aggregator designs, a granted routing patent, a draft Ethereum standard and a protocol fee policy. For each one we say whether it is new, a new combination of known parts, or known. Second, it restates each construction precisely enough to serve as prior art, and pins every statement about behaviour to a file and line of a fixed source commit. Third, it collects the evidence of when each construction was first made public, in a form that can be checked without trusting the author: dated commits, Bitcoin-anchored timestamps, archived captures and DOIs. It also includes an independent derivation of the one optimality result it relies on. The aim is to keep these constructions open: free to study, to cite and to build on, and not available to a later exclusive claim.

1. Scope and method

The whitepaper (Version 2.3) names eighteen innovations, I-1 to I-18, and its §23 already relates the design to convex routing [2, 3], to manipulation of AMM-based oracles [4], to the Uniswap and Curve venue models, and to the MEV and verification literatures. This supplement adds the work that §23 does not discuss and that sits closest to each construction.

Search. For each construction we searched preprint servers, Ethereum research and standards forums, the public documentation and source of the major aggregators (1inch, 0x, CoW Protocol, UniswapX, Paraswap, Odos, KyberSwap) and Google Patents. Patent searches were by keyword and did not sweep a classification exhaustively. A status of **new** means only that a search of this scope found nothing equivalent. It reports the outcome of a search; it does not prove that nothing exists.

Status labels. *New*: no equivalent found. *New combination*: each part is known, and the combination was not found. *Known*: the construction exists in prior work; it is listed only so that the record is complete.

Evidence. Every statement about contract behaviour was checked against the source at commit `982c1de` and is cited as `File.sol:line` at that commit, where `Core`, `Hub`, `Quoter`, `Router` and `Solver` stand for `src/BlazePhoenixCore.sol`, `src/BlazePhoenixHub.sol`, and so on. Where the whitepaper describes something the code does differently, this supplement follows the code. Statements that could not be checked were removed rather than qualified.

2. The constructions

Claim 1 — In-transaction probing and depth-weighted allocation (I-10)

Construction. For each candidate venue the Solver quotes a probe of one hundredth of the order (`Solver.sol:528`) and derives a marginal rate from it (`Solver.sol:575-577`). The rate decides which venues are believable (Claim 9). The allocation weight is a depth read from the venue's state, not from the probe: for constant-product pairs, the shorter reserve side normalised to 18 decimals (`Core.sol:962`); for concentrated-liquidity pools, the depth implied by active liquidity at the current price (`Core.sol:972`). For pools whose holdings can be read, that depth is capped by the output-side balance the pool actually holds (`Solver.sol:615-617`), so a declared depth cannot buy a larger share than the assets behind it. Weights are normalised to the largest, in basis points, with a minimum of one (`Solver._weights` , `Solver.sol:1295`), and each leg receives $x_i = D \psi_i / \sum_j \psi_j$, the last leg taking the remainder (`Solver.sol:752` , `:777`).

Closest prior work. US Patent 12,586,128 B2 [5] (priority 2022-02-06, granted 2026-03-24; original assignee Semiotic AI Inc., current assignee Transaction Assembly Inc.) claims building a graph of assets and pools from ledger state, iterating within the window before the next state update, scoring candidate paths and selecting one. That search runs off-chain, and a routing contract executes its result. The 1inch spot-price aggregator [11] reads liquidity on-chain to weight venues, but as a price oracle for off-chain discovery, not to allocate an order.

Same. Venues are compared by state read from the chain, not by a reported price.

Different. The probe, the weighting and the allocation all run inside the transaction that executes the trade, and the weight is bounded by assets the venue holds.

Status. New combination.

Claim 2 — The Iron-Law floor Φ (I-11)

Construction. A swap whose input is non-zero and whose user minimum is zero is refused (`RouterE(10)` : `Router.sol:409` , `:420` , `:488` , `:525`), so "no floor" cannot be requested. The Router derives a protocol floor in basis points with the same function the Solver uses to plan it, `Core.ironFloorBpsShv` (`Core.sol:2469` ; used at `Router.sol:1484` and `Solver.sol:1676` , `:1775`):

$$\phi = \max(9600 - s_{\text{legs}} - s_{\text{impact}} - s_{\sigma}, 8000),$$

where s_{legs} grows with the effective number of legs, $(\sum a)^2 / \sum a^2$, at 200 bps per effective extra leg (`Core.legShaveBps`), and s_{impact} is the averaged price impact in bps (constants at `Core.sol:299-301`). The protocol floor is that fraction, rounded up, of the in-frame quote for the final hop (`Router.sol:1500`). The enforced minimum is

$$\text{effMin} = \max(\text{userMinOut}, \text{singleOutFloor}, \text{protocolFloorOut}),$$

and the swap reverts unless the measured output reaches it (`Router.sol:1501-1537`); when a transfer fee was measured, `singleOutFloor` is dropped and `protocolFloorOut` rescaled (Claim 12). Delivery to the recipient is measured again as a balance delta and checked against the user minimum (`Router.sol:1577`).

Closest prior work. ERC-8377, *Reference-Relative Slippage Bounds* [6] (draft, created 2026-08-05), proposes that an executor enforce $\max(\text{expectedOut} \cdot (10^4 - \text{maxDeviationBps}) / 10^4, \text{hardFloor})$, with the expected output taken from an external price reference read at execution (an ERC-7726 oracle), and with output measured as the increase in the recipient's balance.

Same. A floor that is the larger of a reference-derived bound and a fixed bound, enforced on a measured balance delta.

Different. Φ uses no external oracle. Its reference is the executing contract's own evaluation of the venues it is about to trade on, computed by the same function that planned the route. The user's floor is mandatory and can only tighten the result.

Status. New combination. The two were developed independently; dates are in §4.

Claim 3 — Quote and execution as one computation (I-1)

Construction. The Solver, the Quoter and the Router price venues through the same Core primitives. `Core.universalQuote` (`Core.sol:1570`) dispatches to `outV2` (`:1144`), `outV3` (`:1174`), `solidlyAskOut` (`:1251`) and the Uniswap V4 book walk (`_v4Walk`, `:2042`, also behind `v4WalkOut`, `:1963`). The Quoter calls `universalQuote`; the Solver calls its ranking variant (`Core.sol:1668`), which delegates to it. The Router calls `outV2`, `outV3`, `solidlyAskOut` and `v4LegOut` (`Core.sol:1785`, which walks the same V4 book). The number the Router computes in-frame is the reference for the floor it enforces (Claim 2).

Closest prior work. The Uniswap V3 QuoterV2 and the V4 Quoter [9] quote by executing the real swap and reverting.

Same. The quote comes from code that matches execution.

Different. The quote is not only advisory: the executing contract recomputes it with the shared primitives and turns it into the bound that execution must meet.

Status. New combination.

Claim 4 — Measurement at every seam (I-12, I-13)

Construction. Each leg's contribution is measured as a balance delta (`Router._execScaled`, `Router.sol:1644`, `:1695`) and must reach 8,000 bps of its bound (`Core.sol:305`; check at `Router.sol:1803`). One transient-storage lock (EIP-1153) is set on entry and cleared on exit, so it is held across the venue callbacks of the whole route (`Router.sol:206`, `:293-299`).

Closest prior work. The fee-on-transfer paths of the Uniswap V2 Router02 [8] measure what a pair received by balance. OpenZeppelin Contracts 5.1 places a reentrancy lock in transient storage [10].

Status. Known. Listed for completeness.

Claim 5 — The Monoslot (I-8)

Construction. A registered venue's record fits in one 256-bit storage word (`Core.sol:2158-2173`; `encodeSlot` at `:2175`, `decodeKind` at `:2206`): an active flag, stable and native flags, fee, venue kind, risk tier, concentration flag, depth bucket, last-update time, exponential moving averages of input and output, a swap counter and two block numbers. It holds no price. A venue can be scored with one storage read.

Closest prior work. Routers such as 1inch's pack flags and a pool address into one 256-bit word of calldata; Uniswap V3 packs pool state into `slot0`.

Different. What is packed is a ranking memory, sufficient to score the venue, and it deliberately excludes price.

Status. New combination, of narrow scope.

Claim 6 — The self-healing registry and the vitality field Ψ (I-9)

Construction. Venues are scored by a vitality that decays with time since the last swap (`Core.vitality`, `Core.sol:2251`; decay at `:2293`) and by a fitness Ψ that multiplies vitality by a depth-bucket weight and bonuses (`Core.psi`, `Core.sol:2271`). The Router reports every executed swap to the Hub (`Hub.recordSwap`, router-only, `Hub.sol:1673`). A venue seen through an executed trade is admitted only if it passes a pair-proof, a check that it trades the pair it claims (`Hub.sol:1734`). When a pair's list is full, a newcomer is admitted only if its projected fitness exceeds that of the weakest incumbent by more than a quarter ($\Psi_{\text{new}} > \Psi_{\text{worst}} + \Psi_{\text{worst}}/4$, `Hub.sol:1930`); the weakest incumbent is then evicted (`Hub._register`, `Hub.sol:1968`; event at `:2004`). The same admission rule applies to a permissionless V4 claim with proof (`Hub.claimV4`, `Hub.sol:969`, rule at `:1016`). An operator can seed venues (`Hub.seedPool`, `:2027`; `Hub.addV4`, `:882`), but the Hub has no function that removes a registered venue by hand: removal happens only by eviction.

Closest prior work. Venue lists curated by aggregator operators, and lifecycle states (active, frozen) set by governance. Our searches found no registry that learns from executed trades in this way.

Different. Admission is earned by trades or by proofs, ranking decays without upkeep, and replacement is automatic and margin-gated.

Status. New.

Claim 7 — Derive, then prove: on-chain venue discovery (I-5, I-6)

Construction. The Hub derives where a venue must be, a CREATE2 address (`Core.deriveAddress`, `Core.sol:435`) or a Uniswap V4 pool id per fee tier (`Core.computeV4PoolId`, `:636`), and admits it only after proving it live on-chain. For addressed venues it checks deployed code and, for an address a factory reports rather than one derived, that the venue's two assets match the pair (`Hub._probe`, `Hub.sol:1120`). For V4 it reads `slot0` of many pools in one batched `extsload` call to the PoolManager (`Core.v4Slot0Batch`, `Core.sol:2104`; used by `Hub._probeV4Batch`, `Hub.sol:1282`, `:1295`) and matches the derived id (`Hub._v4IdMatches`, `:1381`).

Closest prior work. Uniswap V2 `pairFor` and V3 `computeAddress` derive addresses with CREATE2 (EIP-1014); the V4 SDK derives pool ids off-chain; V4 `StateLibrary` reads pool state through `extsload` [9].

Status. Known parts for address derivation. On-chain discovery of V4 pools by derived id and batched proof is a new combination.

Claim 8 — Hooks screened by arithmetic, fees read as executed (I-6)

Construction. A Uniswap V4 hook's permissions are encoded in its address. Whether a hook can alter swap deltas is decided by testing two permission bits of that address, with no call to the hook (`Core.hookAltersDeltas`, `Core.sol:2512`). The Solver drops such pools when planning (`Solver.sol:1486`), and the Router refuses them before it unlocks the PoolManager, so before any asset moves (`RouterE(9)`, `Router.sol:1926` ; unlock at `:1983`). A hook that runs during the swap without altering deltas is identified by a second mask (`Core.hookRunsInSwap`, `Core.sol:2530`) and refused if it is paused (`Router.sol:1941`): a hook is pinned to its code hash when it is admitted, and a listed hook whose code has changed, or a revoked one, counts as paused (`Hub.hookPaused`, `Hub.sol:532`). The registry proposes only hooks that are listed and unchanged (`Hub.isHookLive`, `Hub.sol:558`). A dynamic V4 fee is read as it applies at execution (`Core.effV4Fee`, `Core.sol:1763`).

Closest prior work. Uniswap V4 `Hooks.sol` defines the address-bit encoding [9].

Status. Known mechanism in a new combination.

Claim 9 — The split allocator and its believability band (I-10)

Construction. Before allocation, candidates must be believable: the band is centred on the depth-weighted median of the probed marginal rates (`Solver._depthWeightedMedian`, `Solver.sol:1172` ; applied at `:684`, `:691`). Survivors are cut to a leg budget, keeping the heaviest (`Solver._cutByWeight`, `:1042`), and concentrated pools without a usable balance are removed, with the heaviest leg placed last so that it absorbs rounding (`Solver._seatTheSplit`, `:1006`). Two guards follow. A drain clamp caps the output promised from a concentrated pool that holds its own assets at 3,000 bps of its real holdings, and cuts its input in proportion when the quote exceeds those holdings (`MAX_CONC_DRAIN_BPS`, `Solver.sol:189`, applied at `:822`). A split gate keeps the split only if it beats the best single leg by at least 25 parts per million (`MIN_SPLIT_IMPROVEMENT_PPM`, `Solver.sol:91`, applied at `:954`). Allocation itself is one pass, by Claim 1.

When the one-pass split is optimal. §3 proves that for constant-product pools with a common spot price and a common fee, allocation in proportion to reserves is the unique optimum, and the pools then behave as one pool. Claim 1's weight equals reserves up to a constant factor in that case, so the allocator is optimal there over the venues that pass the believability band (Claim 9) and the per-hop leg budget, up to weight rounding of one ten-thousandth of the largest weight; a venue at the common price can still be left out by those filters. Outside that case we claim no optimality; the split gate is what keeps a suboptimal split from replacing a better single leg.

Closest prior work. Angeris, Chitra, Evans and Boyd [2] show that optimal routing equalises marginal prices across used venues, and Diamandis et al. [3] give an efficient algorithm for it. Greedy and iterative allocators are used in production aggregators. Müller, Moumeni and Messaoudi [4] study the cost of manipulating AMM-based oracles.

Status. The optimum is known. The allocator (one pass, a depth-weighted median band, a drain clamp and a split gate, all on-chain) is a new combination.

Claim 10 — The Surplus Rule (I-14)

Construction. The protocol fee is a fixed rate (`PROTOCOL_FEE_BPS`) on one measured base (`Router._chargeHopFee` , `Router.sol:672`). A transient counter records every fee payment in a swap (`Router._payFee` , `:706`). At settlement, a swap that paid no fee reverts (`RouterE(15)`), and in the anchored regime one that paid more than once reverts (`RouterE(16)`) (`Router.sol:1585-1586`). There is no fee term on the gap between the quote and what is delivered.

Closest prior work. CoW Protocol charges a fee on surplus for out-of-market limit orders and discrete TWAP orders: 50 % of surplus, capped at 0.98 % of the order volume [7].

Different. The opposite commitment: surplus over the quote is not a fee base, and the single fee payment is enforced by the contract rather than stated as policy.

Status. A fixed fee on a measured base is known. Its on-chain enforcement (a per-swap count of fee payments that must equal one) is a new combination.

Claim 11 — The Eightfold Dispatcher Ω (I-4)

Construction. One attribute table, four bits per venue kind (`THETA_ATTR` , `Core.sol:257`), drives every surface; a kind outside the table returns no attributes, which fails closed (`Core.sol:270-272`).

Closest prior work. Dispatch by venue type is standard in aggregators.

Status. Known. Named for authorship only.

Claim 12 — Fee-on-transfer assets, routed where natural

Construction. The Router measures what the pool received, re-quotes the leg on that amount (`outV2`), and records the measured transfer ratio (`Router._noteFot` , `Router.sol:1847` , called at `:1888`), which then rescales the protocol floor (`Router.sol:1534`). Input amounts are measured at entry (`Router.sol:453` , `:553` , `:599`).

Closest prior work. Uniswap V2 Router02 [8].

Status. Known, with an extension (the floor follows the measured ratio).

Claim 13 — Uniswap V4 pool-only output computed in-frame, raising the leg bound (I-6)

Construction. Before a V4 leg executes, the Router computes in-frame the output the pool alone would give for that leg (`Router._v4LegQuote` , `Router.sol:960` , called at `:922` from `_hopScaleImpactAndQuote` , `:780`), by walking the pool's liquidity book (`Core.v4LegOut` , `Core.sol:1785`). The leg's bound is then

$$bound = \max\left(attested \cdot \frac{a}{A}, \text{measured} \cdot \frac{a}{A} \cdot \frac{5000}{10^4}\right)$$

for the amount a actually spent out of the planned A (`MIN_QUOTE_COVERAGE_BPS` , `Router.sol:177` ; applied at `:1747-1750`), and the leg must deliver 8,000 bps of it (Claim 4). The measurement can therefore raise the bound and never lower it: a route that deflates its attested quote below half of the measured one is held to half of the measured one, and attesting more never lowers that bound. Where nothing can be measured, the gate does not run and the aggregate floors of Claim 2 still apply.

Closest prior work. None equivalent found. Aggregators that support hooked V4 pools rely on curated lists.

Status. New.

Claim 14 — Named assurance practices (I-18)

Construction. The repository applies a set of named practices to a production contract system, defined in whitepaper §15 and in the repository's assurance documents: a *shared-quantity register*, which lists every quantity that two pieces of evidence share, so that their independence is stated rather than assumed (§15.10); a *calldata-field confirmation matrix*; *projection distance*, measured over every refusal site (§15.9); and a *curated, named mutation guard* (§15.2–15.3).

Closest prior work. Mutation testing tools (Gambit, certoraMutate); combinatorial covering arrays [12]; requirements traceability matrices.

Status. New as named practices. They are offered as methods and as authored text, not as inventions.

3. Derivation: when a depth-proportional split is optimal

Setting. Let pools $i = 1, \dots, n$ follow the constant-product rule with reserves (X_i, Y_i) , $X_i, Y_i > 0$, and a common input fee factor $\gamma \in (0, 1]$. An input $x \geq 0$ to pool i returns

$$f_i(x) = \frac{Y_i \gamma x}{X_i + \gamma x}, \quad f'_i(x) = \frac{\gamma X_i Y_i}{(X_i + \gamma x)^2} > 0, \quad f''_i(x) = -\frac{2\gamma^2 X_i Y_i}{(X_i + \gamma x)^3} < 0.$$

Problem. Maximise $\sum_i f_i(x_i)$ subject to $\sum_i x_i = D$ and $x_i \geq 0$. The objective is a sum of strictly concave functions and the constraints are linear, so the Karush–Kuhn–Tucker conditions are necessary and sufficient, and the maximiser is unique. They read: for some λ , $f'_i(x_i) = \lambda$ whenever $x_i > 0$, and $f'_i(0) \leq \lambda$ whenever $x_i = 0$.

Common price. Suppose $Y_i = p X_i$ for all i . Put $x_i = t X_i$ with $t = D / \sum_j X_j$. Then

$$f'_i(t X_i) = \frac{\gamma p X_i^2}{X_i^2 (1 + \gamma t)^2} = \frac{\gamma p}{(1 + \gamma t)^2},$$

which does not depend on i ; every x_i is positive; so the conditions hold with $\lambda = \gamma p / (1 + \gamma t)^2$. By uniqueness, $x_i \propto X_i$ is **the optimal allocation**. The total output is

$$\sum_i \frac{p X_i \gamma t X_i}{X_i (1 + \gamma t)} = \frac{p \gamma t \sum_i X_i}{1 + \gamma t} = \frac{(\sum_i Y_i) \gamma D}{\sum_i X_i + \gamma D},$$

the output of a single pool with reserves $(\sum_i X_i, \sum_i Y_i)$.

Link to the code. With a common price, the shorter reserve side of pool i after decimal normalisation is $\min(c_x X_i, c_y p X_i) = X_i \min(c_x, c_y p)$, a constant multiple of X_i . For an honest pair the output-side balance is at least the output reserve, so the cap of Claim 1 does not bind. Claim 1's weights are therefore proportional to X_i , up to their rounding to one ten-thousandth of the largest.

Scope. The result needs both a common price and a common fee. With prices that differ, the conditions above fail in general and the proportional split is not optimal. In a numerical check (2,000 random instances, $2 \leq n \leq 6$, trade sizes from 10^{-4} to 10 times the aggregate input reserve, optimum computed independently

by bisection on λ), the proportional split matched the optimum to a relative 1.7×10^{-12} and the single-pool formula to 5.9×10^{-16} under a common price; with prices perturbed by up to 2 % it fell short of the optimum by up to 3.1×10^{-2} . Nothing is claimed here for concentrated-liquidity or stable-swap venues, where the physical-balance cap of Claim 1 can bind and the result above does not apply.

4. Evidence of authorship and dates

Times are UTC. E1, E2, E4, E5 and E6 can be checked without trusting the author. E3 and E7 rest on repository history, whose commit dates are declared by the author and whose repository is not public at the time of writing; the reference commit is deposited with this record, so its content is fixed by this record's own date. The commit dates below are of the same kind.

#	EVIDENCE	DATE	HOW TO CHECK
E1	Predecessor router (V1) on Base at <code>0x2a779f9Be49aac57495A8B6467Cc325a8a47Eb9f</code> : no code at block 48,361,724, code at block 48,361,725. Context only; no construction below is claimed from it.	2026-07-08 12:13:17	Any Base node or explorer (<code>eth_getCode</code> at both blocks).
E2	Internet Archive capture of <code>https://blazephoenix.xyz/llms.txt</code> .	2026-07-27 01:02:48	<code>https://web.archive.org/web/20260727010248/https://blazephoenix.xyz/llms.txt</code>
E3	First commit of the public source repository, <code>c11d615</code> .	2026-08-12 10:39:16	Repository history; the reference commit <code>982c1de</code> (2026-10-12 00:58:28) is deposited with this record.
E4	Whitepaper Version 2.0, Markdown (sha256 <code>e03003d04d87a1ab00e7bfca748e0ccc18cf33adb-cc94a2eccc1ed6d60a78b4</code>) and PDF (sha256 <code>4362c7a3424d001bf64b101d0ec2053a3af7a-fa27411637061dd0bc63c0a412b</code>), each with an Open-Timestamps proof committed in Bitcoin blocks 962,939, 962,951 and 962,979.	Block times 2026-08-17 19:38:02, 2026-08-17 22:06:13, 2026-08-18 03:16:48	Both files and both proofs are deposited with this record. <code>ots verify <proof> -f <file></code> . We checked each block attestation against the Merkle root served by two independent block explorers.
E5	Whitepaper Version 2.2.	2026-09-06	doi:10.5281/zenodo.22529654
E6	Whitepaper Version 2.3.	2026-10-01 (dated 2026-09-24)	doi:10.5281/zenodo.23084091
E7	Authorship fingerprint: a keccak256 value in the header of the Core source (<code>Core.sol:9</code>), present since the first commit.	from 2026-08-12	Source at E3.
E8	This supplement.	2026-10-19	doi:10.5281/zenodo.23460997

Earliest public appearance of each construction. The Version 2.0 text (E4) describes the Φ floor with the maximum of the user's and the protocol's floors, the probe and proportional allocation, a believability band (anchored differently from the current depth-weighted median), the Monoslot, the vitality field and eviction with learning from executed trades, derive-then-prove with batched `extsload`, hook screening by permis-

sion bits with a code-hash pin, the shared dispatcher used by the Solver and the Quoter, the Surplus Rule, and fee-on-transfer handling. The code at E3 implements Claims 1–9 and 12 in the forms then current, together with a fixed protocol fee and a single-range V4 in-frame quote. Later steps, with the commit that introduced each:

- the rule that the V4 in-frame measurement only raises the leg bound (Claim 13): 5c91008 , 2026-08-24, in its current form (the bound raised to half of the measurement, monotone in the attestation) 2b5d79c , 2026-10-11;
- the pair-proof (Claim 6), the depth-weighted median band (Claim 9), the per-hop measured fee base (Claim 10), the attribute table (Claim 11) and the shared-quantity register (Claim 14): 5c91008 , 2026-08-24;
- the current form of the floor function (Claim 2) and the assurance documents (Claim 14): 84d9b7b and c882f4e , 2026-09-04;
- the transient fee counter (Claim 10): 428eac6 , 2026-09-05;
- the in-swap hook mask and the code-hash pause (Claim 8): 9b3a945 and 98c46cb , 2026-09-07;
- the full V4 book walk (Claim 13): 8916545 , 2026-09-24.

Claim 14 is described in E5. Claim 13 is described in an earlier form in E5 and with the book walk in E6; its current form dates from this record.

5. Limitations

The status labels report searches of the scope in §1, which did not include every patent class or every closed-source aggregator. Line references hold at commit 982c1de only. The comparison with ERC-8377 is with its draft text as published; the draft may change. The optimality result of §3 covers constant-product pools with a common price and fee only.

6. Prior-art statement

The constructions of Claims 1–14, as described in §2, in the whitepaper sections cited there and in the deposited source at the functions named, are disclosed with enough detail for a skilled practitioner to reproduce them. They are dedicated to the public state of the art from the dates given in §4. The construction in Appendix A is disclosed as of the date of this record.

7. How to cite

Mitra (2026). *BlazePhoenix-Dex: Related Work and Prior-Art Statement* (Version 1.0). Zenodo. <https://doi.org/10.5281/zenodo.23460997>

```
@techreport{mitra_blazephoenix_priorart,
  author      = {Mitra},
  title       = {{BlazePhoenix-Dex}: Related Work and Prior-Art Statement},
  institution = {BlazePhoenix},
  type        = {Technical note},
  version     = {1.0},
  year        = {2026},
  publisher   = {Zenodo},
  doi         = {10.5281/zenodo.23460997},
  url         = {https://doi.org/10.5281/zenodo.23460997}
}
```

8. Licence

This text is released under the Creative Commons Attribution 4.0 International licence (CC BY 4.0). The deposited source files remain under their own licence, the Business Source License 1.1, which converts to the GNU General Public License v2.0 or later on its Change Date, 2030-06-01. This statement grants no rights in the code beyond that licence. Its purpose is to keep the ideas it describes open to study, citation and independent implementation.

Acknowledgements

The design and the code were developed together with Fable, an instance of Claude (Anthropic), as recorded in the repository's `CITATION.cff`. The claims in this note, and responsibility for them, are the author's.

References

1. Mitra. *BlazePhoenix Technical Whitepaper: an on-chain liquidity aggregator whose quote and execution are one computation*, Version 2.3. Zenodo, 2026. doi:10.5281/zenodo.23084091.
2. G. Angeris, T. Chitra, A. Evans, S. Boyd. *Optimal Routing for Constant Function Market Makers*. arXiv: 2204.05238, 2022.
3. T. Diamandis, M. Resnick, T. Chitra, G. Angeris. *An Efficient Algorithm for Optimal Routing Through Constant Function Market Makers*. arXiv:2302.04938, 2023.
4. Müller, Moumeni, Messaoudi. *Cost of Manipulation in AMM-Based Oracles*. Financial Cryptography 2026, DeFi workshop. arXiv:2606.03548.
5. M. Deible. *System and method for smart order routing and automatic market maker path determination in a decentralized market*. US Patent 12,586,128 B2, granted 2026-03-24; application US 17/665,579, filed 2022-02-06.
6. F. Firdani. *ERC-8377: Reference-Relative Slippage Bounds*. Draft, created 2026-08-05. <https://eips.ethereum.org/EIPS/eip-8377>
7. CoW Protocol documentation. *Fees*. <https://docs.cow.fi/governance/fees>, accessed 2026-10-13.
8. Uniswap. *Uniswap V2 Periphery: UniswapV2Router02*. 2020.

9. Uniswap. *v4-core* (`Hooks.sol` , `StateLibrary`) and *v4-periphery* (`V4Quoter`); *v3-periphery* (`QuoterV2`).
10. OpenZeppelin. *Contracts 5.1* (`ReentrancyGuardTransient`). 2024-10-17.
11. 1inch. *spot-price-aggregator*. <https://github.com/1inch/spot-price-aggregator>
12. D. R. Kuhn, R. N. Kacker, Y. Lei. *Practical Combinatorial Testing*. NIST Special Publication 800-142, 2010.
13. EIP-1014 (CREATE2); EIP-1153 (transient storage opcodes); ERC-7726 (common quote oracle).

Appendix A — Quote-bound hook routing (disclosed construction, not implemented)

This appendix discloses a design. It is not implemented in the code at commit `982c1de` , which refuses delta-altering hooks outright (Claim 8).

A router that admits a Uniswap V4 pool whose hook can alter swap deltas can bound that hook by the quote itself, rather than by a curated list alone:

1. For each V4 leg it separates the pool-only output, computed in-frame as in Claim 13, from the hook's effect on the delta that execution actually returns.
2. The committed or signed quote carries a cap on the hook's take, derived from the quote. Execution reverts if the measured take exceeds the cap.
3. The quote also carries the hook's `extcodehash` . Execution reverts if the code at the hook address differs from the code that was quoted.

The closest mechanism we know is the code-hash pin on an allow-listed hook (Claim 8; whitepaper §14.2). That pin is a property of the registry. Here the cap and the pin are properties of each quote.