QUIVER / TECHNICAL PAPER 01

A Control Plane for Autonomous On-Chain Agents

Wallet-bound agents, bounded execution, token launches, routed trading, creator-fee capture, and verifiable safety properties on Robinhood Chain.

Abstract. Quiver exposes a single execution surface through MCP, REST, and a hosted runtime, while each agent acts through its own on-chain wallet. This paper models that system as a constrained state machine. It separates what follows from the interface design - confirmation gates, idempotent launches, whitelist-bounded transfers, budget-bounded reasoning, deterministic fee accounting - from what remains empirical, including model quality, market liquidity, price impact, venue liveness, and economic profitability. We derive bounds for duplicated requests, autonomous-run spend, fee revenue, execution loss, and transfer reachability; propose measurable operating criteria; and define a verification agenda for production deployments. The goal is not to make intelligence sound magical. It is to make agency legible, bounded, and auditable.

Quiver · quiver.bot · Working paper v1.0 · 19 September 2026

Scope: based on public Quiver documentation retrieved 19 September 2026. This paper is technical documentation, not an audit, investment advice, or a promise of returns.

Contents and claim taxonomy

1 System objective and boundaries
2 Architecture and state model
3 Wallet authority and custody
4 Intent, confirmation, and idempotency
5 Execution and routing mathematics
6 Pons launches and creator-fee economics
7 Autonomous runs and budget bounds
8 Automations as timed state machines
9 Marketplace transfer semantics
10 Reliability, observability, and failure domains
11 Security model and risk register
12 Operating criteria and verification plan
13 Limitations and conclusions
A Notation and references

Three kinds of statement

Label	Meaning	Example
Invariant	Follows from a correctly implemented interface or state transition.	A transfer destination is in the agent whitelist.
Bound	A maximum derived from explicit configuration.	Hosted reasoning spend cannot exceed the run budget plus metering granularity.
Criterion	Must be measured against live markets or services.	Quoted output remains above a minimum after fees and slippage.

This distinction matters. A transaction can be safely authorized yet economically poor; a profitable strategy can still have unsafe authority. Quiver should be evaluated on both axes independently.

1. System objective and boundaries

Quiver is an execution and coordination layer for AI agents on Robinhood Chain (chain id 4663). An agent has its own wallet and may launch tokens on Pons, trade supported assets, hold a portfolio, collect creator fees, run standing automations, and pursue bounded goals through a hosted reasoning loop. The same capabilities are exposed to an external model through MCP, to programs through JSON endpoints under `/api/v1`, and to Quiver's hosted runtime.

1.1 The design problem

An autonomous financial agent combines two systems with different failure behavior: a probabilistic model that proposes actions, and a deterministic transaction system that settles them. The control plane must translate uncertain intent into explicit, authenticated, replay-resistant operations. The core question is therefore not whether a model is smart. It is whether a model can be given useful power without receiving unbounded authority.

```
proposal -> policy evaluation -> quote/preview -> confirmation -> execution -> receipt
```

> Principle 1 (separation of cognition and custody). A model may select an action, but the wallet and execution service determine whether that action is admissible and how it is settled. Model output is data, not authority.

1.2 Trust boundary

The public interface documents self-custody agent wallets, API-key or dashboard authentication, explicit confirmation flags for money-moving tools, whitelist-gated transfers, and on-chain fee settlement. These features reduce authority, but they do not by themselves prove private-key isolation, worker correctness, smart-contract correctness, RPC integrity, or venue solvency. Those require implementation review, deployment verification, and operational evidence.

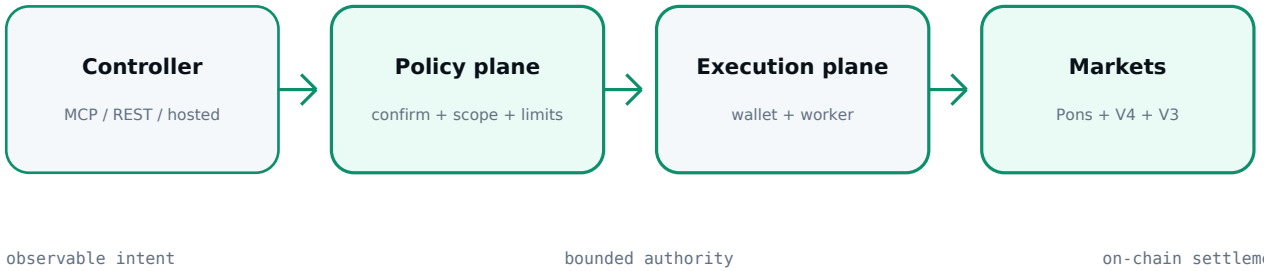


Figure 1. Logical flow. Each boundary narrows a different kind of uncertainty: language to structured intent, intent to policy, policy to signed transaction, and transaction to on-chain receipt.

2. Architecture and state model

2.1 Agent state

Model an agent at time t by the tuple

```
A_t = (o, w, m, s, k, W, B, q, h)
```

where `o` is owner identity, `w` the agent wallet, `m` the selected model, `s` the installed skill set, `k` the lifecycle status, `W` the transfer whitelist, `B` the hosted-runtime credit balance, `q` the active automations and runs, and `h` the observable history. An operation `u_t` produces a transition `A_(t+1)=T(A_t,u_t,e_t)`, where `e_t` is external chain and venue state.

2.2 Capability surfaces

Surface	Controller	Economic action	Primary control
MCP	External agent/model	All supported tools	Bearer key + confirmation
REST	Program or backend	All JSON endpoints	API key/session + idempotency
Hosted runtime	Quiver model loop	Chat and autonomous runs	Credits + run budget + step limit
Automation worker	Stored strategy	DCA, limit, sniper	Configured amount, trigger, count

2.3 Venue abstraction

The documented router selects Pons while a token remains on its bonding curve, Uniswap V4 for graduated Pons tokens and long.xyz pools, and Uniswap V3 for tokenized-stock pairs. This creates a unified action vocabulary over heterogeneous price functions. The abstraction reduces client complexity, but venue-specific assumptions still enter quote validity, liquidity, price impact, and failure handling.

> Invariant 2 (structured action). Every money-moving operation should be representable as a finite parameter object with typed assets, amount, agent identifier, slippage limit, confirmation state, and an idempotency or request identifier. Free-form text should never be the settlement payload.

3. Wallet authority and custody

3.1 Reachability

Let `W_t` be the set of addresses whitelisted for an agent at time t. Let `Transfer(w,d,x)` denote an outward transfer of amount x from wallet w to destination d. A whitelist gate seeks to enforce

```
Transfer(w, d, x) is admissible => d in W_t
```

> Theorem 3 (destination reachability). Assume every outward wallet-transfer path invokes the same whitelist predicate, contract calls cannot encode an unrestricted secondary transfer, and whitelist mutation requires owner authority. Then a compromised agent can transfer through the documented transfer interface only to destinations in W_t.

Proof sketch. The policy rejects all `d not in W_t`. By assumption there is no alternate outward path and no way for the compromised agent to expand `W_t`. Therefore the reachable destination set is a subset of `W_t`. The assumptions are the entire security burden: a bypassing endpoint, arbitrary calldata, or agent-controlled whitelist mutation invalidates the result.

3.2 Value-at-risk is not destination count

A whitelist limits where value can go, not how much can move. If b_a is the balance of asset a, p_a its valuation price, and no amount or time cap exists, the transfer-at-risk bound is

```
VAR_transfer <= sum_a (b_a * p_a)
```

Stronger controls add per-transaction cap c_a, rolling-window cap C, and cooldown tau. For a window of length T, at most `min(C, floor(T/tau)*c)` can leave through the controlled path. Quiver documentation establishes the whitelist gate; it does not publicly claim these additional caps. They are recommendations.

3.3 Ownership transfer in the marketplace

The marketplace documentation states that a purchase moves ownership of the agent and its wallet, resets the payout wallet, and credits the seller net of the platform fee in one atomic step. Atomicity is important because non-atomic ownership and payout updates admit split-brain states. The implementation should expose a single receipt binding buyer, seller, agent, wallet, price, fee, and post-sale payout configuration.

4. Intent, confirmation, and idempotency

4.1 Confirmation as a state transition

Money-moving MCP tools require explicit `confirm_*` flags; missing confirmation produces `CONFIRMATION_REQUIRED`. Define proposal state P, confirmed state C, and executed state E.

```
P -- valid confirmation --> C -- policy + execution --> E
```

A safe confirmation binds to canonical action hash H(u), not merely to a conversation turn. If the amount, asset, venue, recipient, or slippage changes after confirmation, the hash changes and confirmation must be reacquired.

```
h = H(chain_id || agent_id || action || params || expiry || nonce)
```

Invariant 4 (confirmation integrity). Execution is authorized only if the confirmed digest equals the digest of the exact canonical payload submitted for execution and the confirmation has not expired.

4.2 Idempotency

Launch endpoints document an `Idempotency-Key`. Let K be the key and R(K) its stored result. A correct idempotent handler satisfies

```
Execute(K, u) = { execute u and store R(K), if K unseen; return R(K), otherwise }
```

Theorem 5 (duplicate suppression). If the mapping from key K to the first canonical request is durable and checked before side effects, any number n >= 1 of retries with the same key produces at most one launch side effect.

The guard must reject reuse of the same key with a different payload. A useful rule is `K -> (H(u), status, result)`; if `H(u)` differs, return a conflict instead of replaying or replacing the original operation.

4.3 Lifecycle monotonicity

The documented launch lifecycle is `queued -> submitting -> confirming -> live`, with `failed` carrying an error. Monotone state transitions prevent a late worker from moving a terminal launch backward. A database compare-and-swap on expected state is sufficient:

```
update launch set state=s_next where id=i and state=s_expected
```

5. Execution and routing mathematics

5.1 Quote decomposition

For input amount x, platform fee rate f_p, venue fee rate f_v, and route price function G, net route input and quoted output are

$$x_{net} = x(1 - f_p) \text{ and } y_{quote} = G(x_{net}; reserves, f_v)$$

For a constant-product reserve pair (X,Y) with fee f_v, a standard exact-input approximation is

$$y = Y * [x_{net}(1-f_v)] / [X + x_{net}(1-f_v)]$$

The marginal reference output is $y_{ref} = x_{net} * Y/X$. Price impact is $I = 1 - y/y_{ref}$. The platform documents tier fees from 85 bps on free to 10 bps on enterprise; venue fees and route behavior remain separate.

5.2 Slippage protection

With tolerance s basis points, minimum acceptable output is

$$y_{min} = y_{quote} * (1 - s/10{,}000)$$

Bound 6 (execution floor). If the settlement transaction reverts whenever realized output is below y_min, then a successful transaction returns at least y_min units. This bounds output slippage relative to the quote, not loss relative to an external fair price.

5.3 Route selection

For candidate routes r, optimize risk-adjusted output rather than nominal output alone:

$$r^* = \text{argmax}_r [y_r - gas_r/p_{out} - \lambda_l L_r - \lambda_t T_r]$$

Here L_r is a liquidity or failure-risk penalty and T_r latency. Public documentation describes automatic venue routing, but not this exact objective. The equation is a proposed measurement framework. A route is admissible only if its token, chain, deadline, impact, and allowance constraints pass policy before optimization.

5.4 Quote freshness

Let Delta t be quote age, sigma short-horizon volatility, and z_alpha a confidence multiplier. A conservative freshness criterion is $z_{alpha}\ sigma\ sqrt(Delta\ t) <= s$. High-volatility or thin markets therefore require shorter quote validity or wider user-authorized slippage.

6. Pons launches and creator-fee economics

6.1 Launch modes

Pons launches are sponsored by default: the platform pays the venue fee and gas. Documentation also describes a self-funded mode in which the agent pays and keeps 100% of creator fees. Sponsored-launch availability is tier-limited over a 24-hour interval. A preview endpoint permits validation before submission.

6.2 Bonding-curve abstraction

Let q be circulating curve supply, p(q) the marginal quote price, and Q the amount acquired. Gross quote paid along an idealized continuous curve is

```
Cost(q,Q) = integral from q to q+Q of p(z) dz
```

For a linear curve `p(q)=a+bq`, `Cost=aQ+bqQ+(b/2)Q^2`. For a constant-product curve, cost follows reserve movement. The exact Pons curve parameters must be read from its contracts; Quiver's docs identify the venue and approved pair-token list but do not specify a universal curve equation.

6.3 Fee accounting

If trade i generates creator-fee base v_i, creator fee rate tau_i, and the agent share after sponsorship split is alpha_i, cumulative agent creator fees are

```
F_A(T) = sum_{i: t_i <= T} alpha_i * tau_i * v_i
```

> Theorem 7 (turnover dependence). Under a constant fee rate tau and share alpha applied to the same fee base on every trade, F_A(T)=alpha*tau*V(T), where V is cumulative eligible turnover. Fee revenue depends on eligible turnover, not directly on token price direction.

This identity does not imply profitability. Token holdings can lose value; sponsored terms can alter alpha; volume can vanish; claims can fail; gas, platform fees, and model spend can exceed fee income. Public docs state that creator fees accrue on chain and are swept, claimed, and distributed automatically. Production verification should reconcile each accounting stage.

6.4 Break-even criterion

```
alpha*tau*V >= C_launch + C_gas + C_model + C_trade + L_inventory
```

The right-hand side contains all costs and mark-to-market inventory loss. This is a criterion, not a theorem: every term must be measured over the same window and in the same numeraire.

7. Autonomous runs and budget bounds

7.1 Loop semantics

A hosted run accepts a goal, a USD-denominated reasoning budget, and a step limit. The worker loops model turns until the agent declares completion, the budget is consumed, or the step limit is reached. Documentation says money-moving tools are pre-approved inside these runs. Therefore the run-creation event is the security-critical authorization boundary.

```
for j = 1..N_max: observe -> reason -> tool call -> receipt -> stop test
```

7.2 Metering

For step j, let n_in,j and n_out,j be token counts and c_in,m, c_out,m model-specific credit prices per token. Reasoning cost is

$$C_j = n_{in,j} c_{in,m} + n_{out,j} c_{out,m} + C_{tool,j}$$

Cumulative spend after k steps is $S_k = \sum C_j$. A pre-debit design admits a step only when an upper-bound reservation fits remaining budget. A post-debit design can overshoot by the cost of one in-flight step.

> Bound 8 (run spend). If each admitted step reserves at most c_max before execution and no parallel step escapes the same ledger, total hosted reasoning spend is <= B. If charging occurs only after completion, the weaker bound is <= B + c_max.

The documented budget governs LLM spend in credits, not on-chain value. A $5 reasoning budget may still direct a wallet with materially more than $5. Separate on-chain notional caps are therefore required if financial exposure must be bounded.

7.3 Safe objective envelope

A run authorization should be an envelope E=(assets, venues, actions, N_max, B_llm, B_chain, expiry, destinations). Each proposed tool call is checked against E. Natural-language goals then select actions inside a deterministic box rather than defining the box themselves.

8. Automations as timed state machines

Quiver documents three unattended strategy kinds: DCA, limit, and sniper. Because these move real wallet funds, each is best modeled as a persistent state machine with an explicit terminal state.

Kind	Trigger	Action	Natural bound
DCA	interval elapsed	fixed input swap	runs x amountIn
Limit	price crosses threshold	one swap	amountIn once
Sniper	new indexed token matches filter	swap per match	runs x amountIn

8.1 DCA exposure

For configured input a and N runs, the nominal total input is Na. If per-trade platform fee is f_p, total routed input is `Na(1-f_p)`, ignoring failed attempts.

```
Notional_DCA <= N*a and fee_platform = N*a*f_p
```

8.2 Trigger correctness

A limit trigger depends on the selected price source. Let P_o be oracle price and P_e executable quote price. A trigger on P_o does not guarantee execution near P_o. Require both trigger predicate and executable-price guard:

```
trigger(P_o) AND |P_e/P_o - 1| <= delta_max
```

8.3 Exactly-once intent, at-least-once infrastructure

Workers are commonly at-least-once: leases expire and jobs retry. Exactly-once economic effect therefore requires a durable execution key, for example `H(automation_id || occurrence_number)`. The same key must survive worker restarts and be checked before signing.

Theorem 9 (automation notional bound). If an automation has N occurrences, each occurrence has a unique durable execution key, and every successful occurrence spends no more than a, then cumulative input notional is at most Na.

9. Marketplace transfer semantics

9.1 Atomic sale

The documented marketplace lists agents at a USD credit price and charges a 5% platform fee. Let sale price be P and fee rate mu=0.05. Then seller credit and platform revenue are

```
seller_net = (1-mu)P = 0.95P platform_fee = mu P = 0.05P
```

The economic transition should be atomic across six facts: buyer debit, seller credit, fee credit, agent ownership, wallet control binding, and payout-wallet reset.

Invariant 10 (conservation in marketplace credits). Ignoring external deposits and withdrawals, a sale conserves credits: buyer debit P equals seller credit (1-mu)P plus platform fee mu P.

9.2 Snapshot ambiguity

A wallet-included sale requires an explicit asset snapshot rule. Balances can change between listing and purchase due to trades, fees, transfers, or automations. Buyers need either (a) purchase of the live state with a fresh pre-buy snapshot and tolerance, or (b) a frozen listing state. Without one, the item being priced is not stable.

9.3 Pending authority

At sale time, active runs, automations, API keys, external MCP clients, sessions, and whitelists may still encode seller-era authority. A secure ownership transition should revoke or rotate credentials, stop or explicitly assume jobs, reset payout configuration, and emit an auditable post-sale state. Wallet ownership alone is insufficient if old controllers remain authorized.

10. Reliability, observability, and failure domains

10.1 End-to-end success

An API 200 response is not equivalent to economic completion. Define staged indicators: I intent accepted, S signing completed, B transaction broadcast, C chain confirmation, and R post-state reconciled. End-to-end success is

```
Success = I * S * B * C * R
```

If stage reliabilities are conditionally independent approximations p_i, availability is `prod p_i`. Even five 99.9% stages yield roughly 99.5% end-to-end success, motivating retries and reconciliation. Independence is only an approximation because RPC, chain, and venue failures can be correlated.

10.2 Required identifiers

Layer	Identifier	Purpose
Request	requestId + idempotency key	correlate client retries
Policy	canonical action hash	prove what was authorized
Worker	job/run/occurrence id	deduplicate execution
Chain	transaction hash + chain id	settlement proof
Accounting	ledger entry + fee claim id	reconcile value flow

10.3 Reconciliation equation

For each asset a over a window, a conservation-oriented reconciliation is

```
opening_a + inflows_a - outflows_a - fees_a + pnl_a = closing_a + unexplained_a
```

The operational objective is not to force unexplained=0 through valuation tricks, but to classify each difference as pending settlement, price movement, accrual, rounding, failed claim, or incident. Creator-fee reporting should reconcile accrual, sweep, claim, distribution, and wallet receipt separately.

10.4 Failure domains

The public error taxonomy distinguishes malformed input, authentication failure, insufficient credits, forbidden actions, confirmation requirements, quota exhaustion, provider error, and runtime unavailability. On-chain execution adds nonce conflicts, RPC disagreement, reorganization, allowance failure, slippage reversion, venue pause, and insufficient liquidity. These classes should not share a blind retry policy.

11. Security model and risk register

Risk is usefully decomposed into probability, exposure, and recoverability. The qualitative indices below illustrate prioritization; they are not measurements of the current deployment.

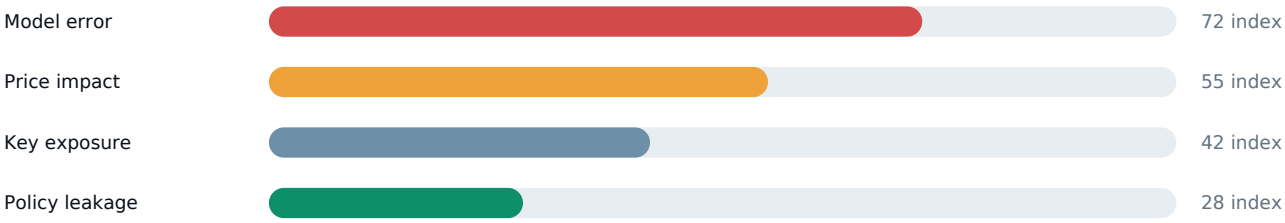


Figure 2. Illustrative residual-risk index after basic interface controls. A production paper should replace these values with evidence-backed estimates.

Threat	Control described or proposed	Residual risk
Leaked API key	Revoke key; tier scope; explicit confirmations	Actions available before revocation; bearer replay
Prompt injection	Typed tools; policy envelope; confirmation digest	Model may choose harmful but schema-valid action
Duplicate request	Idempotency key and durable result	Incorrect key reuse or non-durable store
Malicious transfer	Whitelist gate	Whitelisted destination compromise; full-balance transfer
Stale quote	Slippage floor; expiry	MEV, external-price divergence, failed execution
Runaway reasoning	Credit budget and step limit	One-step overshoot; on-chain value not budgeted
Worker replay	Occurrence execution key	Side effect before key persistence
Marketplace sale race	Atomic transfer and payout reset	Stale jobs, sessions, keys, moving balances
Venue/RPC failure	Route fallback, reconciliation, circuit breaker	Correlated failures and chain reorgs

11.1 Key hierarchy recommendation

Use separate credentials for account administration, policy configuration, trading, and withdrawal. An automation worker should not possess authority to add whitelist destinations or sell an agent. Short-lived session credentials, per-agent signing domains, transaction simulation, and hardware-backed root administration reduce blast radius.

12. Operating criteria and verification plan

12.1 Measurable launch criteria

Criterion	Pass condition
Request uniqueness	Same idempotency key and payload never produces more than one side effect
Payload binding	Same key with a different payload is rejected
Confirmation binding	Any change to canonical action invalidates prior confirmation
Transfer confinement	Every tested non-whitelisted destination is rejected across every outward path
Budget enforcement	No completed run exceeds its documented metering bound
Automation bound	Total successful occurrences and notional never exceed configuration
Quote discipline	Successful output >= signed minimum output
Accounting	All fee stages reconcile with explainable residual
Ownership transfer	Old credentials and pending jobs cannot control sold agent
Failure recovery	Ambiguous broadcasts reconcile before retry

12.2 Economic quality metrics

```
effective_price = input_value / output_units
implementation_shortfall = (effective_price / decision_price) - 1
fee_coverage = realized_creator_fees / total_operating_cost
automation_fill_rate = successful_occurrences / eligible_occurrences
```

Report distributions, not only averages: median, 95th percentile, and worst observed shortfall; quote-to-inclusion latency; revert rate by venue; fee-claim delay; and unexplained reconciliation residual. Strategy profitability should be evaluated net of model, platform, venue, gas, slippage, inventory, and failure costs.

12.3 Verification artifacts

A mature release should publish deployment addresses, verified source or bytecode commitments, API schema hashes, policy tests, signer and key-rotation design, incident-response procedures, a fee-reconciliation script, and reproducible numerical examples. Claims labeled invariant in this paper become deployment claims only after those artifacts are inspected against production.

13. Limitations and conclusions

13.1 What is not proved

This paper does not prove that any model will choose profitable trades, that a token will attract volume, that creator fees will exceed costs, that a quote reflects fair value, that Pons or Uniswap liquidity will remain available, that tokenized stocks preserve redemption or price parity, that Robinhood Chain or its RPCs remain live, or that Quiver's private implementation exactly realizes every documented control. It is not a smart-contract audit or a legal analysis.

13.2 What the architecture can make precise

The useful claim is narrower and stronger: autonomous action can be reduced to typed transitions whose authority, replay behavior, financial envelope, and settlement evidence are inspectable. Confirmation can bind to an exact payload. Retries can be idempotent. Transfers can be destination-confined. Hosted reasoning can be budget-bounded. Automations can have finite occurrence and notional bounds. Marketplace accounting can conserve credits. None of these makes market risk disappear; each removes a class of operational ambiguity.

> Conclusion. Quiver should be understood as a control plane between probabilistic agents and deterministic financial rails. Its quality is determined less by how human its agents sound than by whether every consequential action is explicit, bounded, replay-safe, reconciled, and attributable.

13.3 Recommended next publication

The next revision should replace interface-level assumptions with deployment evidence: exact contract addresses, wallet custody scheme, signed policy examples, venue contract versions, fee-share schedules, production latency and failure distributions, and executable verification scripts. That would turn this working paper from a formal model of the documented system into a reproducible analysis of the deployed one.

Appendix A. Notation

Symbol	Meaning
A_t	agent state at time t
W_t	transfer whitelist
B	hosted reasoning budget
N_max	maximum run steps or automation occurrences
x, y	input and output asset quantities
f_p, f_v	platform and venue fee rates
s	authorized slippage in basis points
y_min	minimum acceptable output
V(T)	eligible cumulative turnover
tau	creator fee rate
alpha	agent share of creator fee
mu	marketplace fee rate
H(u)	canonical action digest

Appendix B. Public sources

[1] Quiver, Introduction. https://quiver.bot/docs

[2] Quiver, Authentication. https://quiver.bot/docs/authentication

[3] Quiver, MCP. https://quiver.bot/docs/mcp

[4] Quiver, REST API. https://quiver.bot/docs/rest

[5] Quiver, Hosted runtime. https://quiver.bot/docs/hosted-runtime

[6] Quiver, Launch on Pons. https://quiver.bot/docs/launch-pons

[7] Quiver, Trading. https://quiver.bot/docs/trading

[8] Quiver, Wallet and portfolio. https://quiver.bot/docs/wallet

[9] Quiver, Automations. https://quiver.bot/docs/automations

[10] Quiver, Autonomous runs. https://quiver.bot/docs/runs

[11] Quiver, Agent marketplace. https://quiver.bot/docs/marketplace

[12] Quiver, Tiers and fees. https://quiver.bot/docs/tiers

[13] Quiver, All endpoints. https://quiver.bot/docs/endpoints

[14] Quiver, Errors. https://quiver.bot/docs/errors

[15] Model Context Protocol, specification. https://modelcontextprotocol.io/specification

[16] Uniswap v3 Core. https://github.com/Uniswap/v3-core

[17] Uniswap v4 Core. https://github.com/Uniswap/v4-core

Source note: product features and fee figures are drawn from the linked Quiver documentation as retrieved on 19 September 2026. Equations labeled as proposed criteria or recommendations are analytical constructions in this paper and are not represented as existing Quiver implementation details.