Draft 0.1
November 2026
draft, not peer reviewed

allto Labs
research@allto.trade
allto.trade

q-fin.TR · cs.CR
Trading & Market Microstructure
Cryptography & Security

allto: An Auditable Autonomous Trading Agent with a Hard Risk Ceiling

allto Labs · t. yebtimothy · Maximus Maxis

Correspondence: research@allto.trade · *Subjects:* Trading and Market Microstructure (q-fin.TR); Cryptography and Security (cs.CR)
Classification: Technical whitepaper (system design and security analysis). Not peer reviewed; comments welcome.

ABSTRACT

Trading bots conflate the **proposal** of a trade, the **refusal** of a trade and the **record** of both. *allto* separates them. The agent scores each observation against a cost model and submits it to a *risk kernel*, a pure function over a versioned limit set and a caller-supplied state snapshot, which approves the order or returns a veto naming the gate that fired. Nothing in the decision path holds a clock, a socket or a database handle, so the path is a deterministic map from recorded inputs to recorded outputs, and every observation, score, veto and order lands in a hash-chained ledger where each record commits to its predecessor by SHA-256. We formalise the kernel as five indicator gates in a fixed attribution order, prove approval is invariant under permutation while attribution is only equivariant, and show the ledger's length-prefixed encoding is prefix-free. Size follows a constrained Kelly program whose constraints are the kernel's own caps, so limit and sizing rule cannot diverge; the approved set is S_5-invariant; venue exposure is a coherent tail functional rather than a variance proxy. The non-custodial session-key boundary is specified, not yet deployed, and the paper says so where it matters.

Keywords: autonomous agents, risk kernels, deterministic replay, hash-chained logs, session keys, Kelly sizing, coherent risk measures

---

1 Introduction
2 Notation and System Model
3 Architecture
4 The Decision Function
5 The Risk Kernel
6 The Ledger
7 Deterministic Replay
8 Performance Functionals
9 Custody and the Session-Key Boundary
10 Launch Discovery Is Not a Price Source
11 Implementation, Limitations, Conclusion
References

---

1 Introduction

A trading bot is conventionally a pipeline: read a price, compute a signal, send an order. It is easy to build and hard to trust, for three structural reasons. **The decision is not inspectable after the fact.** By the time a loss is understood the snapshot has been overwritten, the model retrained, the threshold tuned, and what remains is a fill and a narrative. **The limit is advisory.** Production risk checks are monitoring services that page a human while the order path continues, so any bug reaching the signer bypasses the check. **The number on the marketing page is typed in.** A figure not derived from the records that drove the trades is at best stale and at worst fabricated.

allto removes all three architecturally. The decision path is a **pure function**: every input that could vary (time, market data, limits, realised performance) is passed in, so a replay of recorded inputs reproduces recorded outputs byte for byte. The limit is **evaluated inside the signing path**, in a package that cannot reach the network, the clock or a database, and the same limit is enforced onchain by a session key the operator cannot widen. Every displayed figure is **derived from the ledger** by a stated functional; where the product still quotes a constant that is an open problem (§11f), not a derivation.

2 Notation and System Model

2.1 Notation

t_i, v_i, m_i, p_i, $\hat{e}_i$, N_i	time, venue, market, price, fair-value estimate, notional
λ, δ	versioned limit set; drawdown kill threshold (a kernel constant, §11a)
$s = (D, d, \{V_v\})$	day loss, normalised drawdown, approved notional per venue
e_i, $\tilde{e}_i$, θ	net edge, correlation-discounted edge, score threshold (points)
γ_i, Γ	candidate/book correlation; correlation ceiling
H_n, seq_n	n-th ledger digest and sequence number
E_j, P_j, r_j	equity, running peak, day return
Λ, η, v	launch liquidity, holder count, 24-hour volume (§10)

A *point* is one hundredth of a unit of notional: on a $1 contract, one point is $0.01. Points are the common unit of edge, cost and threshold throughout.

2.2 Actors

- **User.** Holds funds in a smart account and grants the agent a scoped session key.
- **Agent.** Proposes trades and holds no custody.
- **Signer.** Signs only orders carrying a valid risk-approval token.
- **Chain.** Enforces the session key's permission set, the only boundary an attacker cannot edit.

2.3 Assumptions

- **A1 (Determinism).** The engine reads no global clock and draws no randomness.
- **A2.** SHA-256 [1] is collision- and preimage-resistant.
- **A3.** Transactions confirmed beyond the router's reorg depth are final.
- **A4 (Honest measurement).** The caller measures day loss, drawdown and venue notional correctly; the kernel detects a *breach* of a limit, not a *wrong measurement* of one.

2.4 Adversary

We allow compromise of the web tier, of the engine and signer, of the ledger at rest, and attempts to induce mandate-breaching trades; the adversary does not control the user's keys, chain consensus or SHA-256. It cannot then (i) move user funds, (ii) approve token spend, (iii) place an order exceeding the on-chain permission set, or (iv) alter a historical record undetected. Claims (i)–(iii) are conditional on the boundary of §9 being deployed; in the current build only (iv) is testable against shipped code, and (i)–(iii) hold because there is as yet no signer and no capital at risk.

3 Architecture

One Go binary today, with seams for the multi-service form the plan targets. HTTP pages, the SSE stream, the tick loop, the kernel and the ledger store run inside `cmd/web`'s process; kernel and engine are separate packages behind an import boundary CI enforces, so splitting them into processes is a deployment change rather than a redesign.

The rule that gives the architecture its security property: **the engine is the only component that may ask the signer to act, and the signer acts only on orders carrying a risk-approval token it verifies independently.** A defect in the web tier cannot place an order, because the web tier has no route to the signer. In today's single binary the rule holds in the weaker form the build supports: there is no signer to reach, and the web tier reads verdicts for display, never producing one.

Three choices follow from the auditability goal. **Server-sent events, not WebSockets**: push traffic is one-way, ordering is per connection, and `Last-Event-ID` reconnect is served from an in-process ring rather than a broker we would operate [10]; the write path stays an ordinary HTTP request that CSRF middleware and rate limits already cover. **Templates compiled to Go**: `templ` [12] type-checks markup at build time, so a page referencing a field the engine no longer emits fails the build, not the browser. **The ledger is the system of record**: the broker is a bounded-ring transport; a subscriber that cannot keep up is dropped and replayed on reconnect, never buffered without bound.

4 The Decision Function

An observation is

$$o_i = (t_i, v_i, m_i, p_i, \hat{e}_i, N_i).$$

The raw signal $| \hat{e}_i - p_i |$ is optimistic: it ignores the cost of holding and of crossing. The engine scores **net** edge,

$$e_i = | \hat{e}_i - p_i | - 100(\rho h/365 + \sigma/10^4),$$

with ρ the annualised borrow rate, h holding period in days and σ round-trip slippage in basis points; the factor 100 converts notional fractions into points so both terms are commensurable. The reference model is deliberately pessimistic (ρ = 0.06, h = 3, σ = 25 bp gives $100(0.06 \cdot 3/365 + 0.0025) \approx 0.30$ points), chosen against the stylised facts of intraday returns (heavy tails, volatility clustering, persistent spread costs [7]) rather than fitted to a favourable sample. A strategy that works only at zero borrow and zero slippage does not work.

Two markets that move together are one position, not two, so the score is discounted by correlation with existing book exposure,

$$\tilde{e}_i = e_i(1 - \min(\gamma_i, 1)), \quad \gamma_i \in [0, 1].$$

The discount is implemented and tested but not yet wired in: the engine passes $\gamma_i = 0$, so $\tilde{e}_i = e_i$ today and the gate consuming γ_i is dormant. Finally, if $\tilde{e}_i < \theta$ the order never reaches the kernel; the engine records a veto under a pseudo-gate `score_threshold` deliberately **not** one of the five limits, because a low score is the strategy declining, not the risk system refusing. With $\theta = 5$ points,

$$\mathrm{submit}(o_i) = \mathbb{1}[\tilde{e}_i \geq \theta].$$

5 The Risk Kernel

5.1 Definition and gates

The kernel is a pure function

$$\mathcal{K} : \mathcal{O} \times \Lambda \times \mathcal{S} \longrightarrow \{ \mathrm{approve} \} \cup \{ \mathrm{veto}(g, r) \},$$

importing no network, filesystem, clock, database or context package (enforced by a CI check on direct imports). The order carries only what the caller measured, $o = (\mathrm{id}, v, N, \gamma)$; the limit set is $\lambda = (C_{\mathrm{trade}}, L_{\mathrm{day}}, \{ C_v \}_{v \in V}, \Gamma)$; the state is $s = (D, d, \{ V_v \})$ with $D \geq 0$ the realised day loss, d drawdown as a fraction of peak equity, and V_v notional already approved at venue v. Each gate is an indicator:

$$g_1 = \mathbb{1}[\, N > C_{trade} \,],$$

$$g_2 = \mathbb{1}[\, D \geq L_{day} \,],$$

$$g_3 = \mathbb{1}[\, d \geq \delta \,],$$

$$g_4 = \mathbb{1}[\, C_v \text{ defined} \wedge V_v + N > C_v \,],$$

$$g_5 = \mathbb{1}[\, \gamma > \Gamma \,].$$

The approval predicate is their conjunction

$$A(o, \lambda, s) = \prod_{k=1}^{5} (1 - g_k) \in \{0, 1\},$$

and attribution is the first firing gate, $g^* = \min \{ k : g_k = 1 \}$.

> **Proposition 1 (Order-invariance of approval, equivariance of attribution).**
> For any $\pi \in S_5$, $\prod_k (1 - g_k) = \prod_k (1 - g_{\pi(k)})$; but if $g_1 = g_3 = 1$ then $g^* = 1$ canonically and $g^* = 3$ under the transposition $(1\ 3)$.
>
> *Proof.* Multiplication of $\{0,1\}$ factors is commutative, giving the first claim. For the second, min over the *set* of firing indices is invariant, but the reported element is the least under the chosen order; a transposition changes which element is least. ∎

Safety is therefore robust to implementation order, while explanation is a function of a convention, so the gate order is part of the contract, and changing it is a breaking change requiring a rules-version bump.

5.2 The approved set is S_5-invariant

Let S_5 act on $\mathcal{G} = \{ g_1, \ldots, g_5 \}$ by relabelling. Because A is symmetric in the gate vector it is constant on S_5-orbits, $A(\pi \cdot o) = A(o)$ for all π, so the approved set $\mathcal{A}^+ = \{ o : A(o) = 1 \}$ is a union of orbits and so is its complement: the kernel is an intersection of five permutation-invariant half-spaces, and no gate is

privileged in *whether* an order is approved. Attribution is the equivariant counterpart, g*(π · o) = π(g*(o)), Proposition 1 read as a group action rather than a list of products. Two consequences: adding a sixth gate cannot silently relax the existing five, since the approved set can only shrink under intersection, and the state space is a quotient, so the ledger need record only the canonical representative plus the rules version, not the permutation used at evaluation time.

5.3 The mandate is data, the kernel is code

The kernel never reads limits from a file, an environment variable or a database; it receives λ as an argument. A rules engine configured from data would make the decision path *data*, and a configuration change could silently alter the behaviour of a past decision. Versioning λ and stamping every record with the rules version keeps the two separable. The reference mandate is

$$C_{trade} = 1000, \quad L_{day} = 500, \quad C_v = 5000 \text{ for } v \in \{ \text{uniswap, curve, kalshi} \}, \quad \Gamma = 0.8.$$

The single exception is δ, still a kernel constant: precisely the defect in §11(a).

5.4 Constrained Kelly sizing

The caps are not merely veto conditions; they are the feasible set of the sizing problem. Let f be the fraction of equity committed and R the trade's return. The growth-optimal size solves

$$\max_{f \geq 0} \mathbb{E}[\log(1 + fR)] \quad s.t. \quad fN \leq C_{trade}, \quad V_v + fN \leq C_v \ \forall v.$$

With multipliers μ, { ν_v } ≥ 0 the Lagrangian is $\mathbb{E}[\log(1 + fR)] - \mu(fN - C_{trade}) - \Sigma_v \nu_v(V_v + fN - C_v)$, and stationarity gives

$$\mathbb{E}[R / (1 + fR)] = N(\mu + \Sigma_v \nu_v),$$

with complementary slackness $\mu(fN - C_{trade}) = 0$ and $\nu_v(V_v + fN - C_v) = 0$. Where no cap binds, $\mu = \nu_v = 0$ and the condition reduces to the unconstrained Kelly criterion $\mathbb{E}[R/(1 + fR)] = 0$. Where caps bind, the right-hand side is the shadow price of the tightest constraint and the solution is water-filling:

$$f^* = \min \{ f_{Kelly}, \quad C_{trade}/N, \quad \min_v (C_v - V_v)/N \}.$$

The sizing rule and the risk ceiling are thus the *same object*: the caps that veto an order are the caps that bound its size, so no configuration exists in which the agent sizes to a limit the kernel does not enforce. The shadow price is an interpretable diagnostic: a small μ means the trade cap is slack and size is set by

edge, a large μ means the mandate rather than the model is choosing the position, and the utilisation u_k of §8 reads off which regime applies.

5.5 Fail-closed composition

Approval is necessary but not sufficient. The signer requires and independently verifies a risk-approval token, and the chain requires the transaction to fall inside the session key's permission set. The three compose in series,

$$settle \iff A(o, \lambda, s) \wedge tokenValid(o) \wedge o \in \mathcal{A}(P),$$

where $\mathcal{A}(P)$ is the set admitted by permission set P. Failure of any term yields no transaction: if the risk service is unavailable, A cannot be established and nothing is placed. The system fails closed. Only the first term is implemented today: the paper loop stops at the kernel's verdict, and the later terms are properties of the undeployed boundary of §9.

6 The Ledger

Let $\ell(x)$ be the length-prefixed encoding of x (an 8-byte big-endian length followed by x) and $\nu(z)$ the 8-byte big-endian encoding of an integer. For record n with sequence seq_n, kind k_n, timestamp τ_n (ns since epoch), rules version r_n and payload π_n:

$$H_n = SHA256(\nu(seq_n) \,\|\, \ell(k_n) \,\|\, \nu(\tau_n) \,\|\, \ell(r_n) \,\|\, H_{n-1} \,\|\, \ell(\pi_n)), \quad H_0 = 0^{256}.$$

Each record stores H_{n-1} alongside H_n, so the chain is self-describing: a reader recomputes it from the records alone, without trusting the store that served them. The construction is the hash-linked log of Nakamoto [2] with Merkle's tree [3] collapsed to one leaf per level.

> **Proposition 2 (Prefix-freedom).**
>
> ℓ is prefix-free.
>
> *Proof.* Each codeword begins with an 8-byte length field declaring the length of what follows. For distinct codewords either the length fields differ, and then neither is a prefix, since a prefix must share the first eight bytes, or the lengths agree and the contents differ inside the declared window. ∎

Corollary. Concatenations of ℓ-encoded fields are unambiguous, so two distinct field tuples cannot yield the same byte string: any single-field modification changes the digest input, and detection reduces to A2.

Proposition 3 (Local tamper propagation).

If record j is modified, dropped or reordered, then $H_n \neq H_n^{\text{orig}}$ for every $n \geq j$.

Proof. By the corollary, modifying j changes its digest input, so H_j changes unless SHA-256 collides; since H_{j+1} includes H_j, induction gives the claim. Dropping j shifts later sequence numbers, changing their ν(seq) fields; reordering changes at least one ν(seq) and one H_{prev}. ∎

Verification is one forward pass comparing recomputed digests against stored ones and checking $\text{seq}_n = n$ and H_{n-1} agreement; the first mismatch localises the tamper point. The kind set is closed and versioned, and a new kind is a schema change. Corrections are *new* records, never updates, so the history remains a faithful account of what the system believed at each moment.

7 Deterministic Replay

Let $\mathscr{R}$ be a recorded observation sequence, λ a limit set, θ a threshold, c a cost model and ψ a realised-performance source. The engine is $\mathscr{E} : (\mathscr{R}, \lambda, \theta, c, \psi) \longrightarrow (\mathcal{D}, \mathcal{L})$ and the harness is

$$\mathscr{H}(\mathscr{R}, \lambda, \theta, c, \psi) = (\mathscr{E}(\cdot), \Delta(\mathcal{L})), \quad \Delta(\mathcal{L}) = \mathrm{SHA256}(\mathrm{encode}(\mathcal{L})),$$

where `encode` serialises one JSON object per line with HTML escaping disabled, so bytes are stable across runs and machines.

Theorem 1 (Replay determinism).

Under A1, for fixed $(\mathscr{R}, \lambda, \theta, c, \psi)$, $\Delta(\mathcal{L}_1) = \Delta(\mathcal{L}_2)$ for any two independent runs.

Proof. $\mathscr{E}$ is a sequential fold over $\mathscr{R}$: at step i it appends an observation record, computes e_i by arithmetic on $(p_i, \hat{e}_i, N_i, c)$, queries ψ at t_i, evaluates $\mathcal{K}$, and appends exactly one of an order or a veto. Each step depends only on the accumulator and o_i; by A1 no step reads a clock or a random source, so by induction each record is determined by the inputs. Records are produced in a fixed order and `encode` iterates a slice rather than a map, so the byte stream is determined, and Δ of a determined stream is determined. ∎

The theorem is asserted of the fold and enforced by a golden digest over a recorded fixture, not mechanically proved (§11g). It converts a behavioural change into a bit-level observable: any change to scoring, cost or the kernel must either reproduce the golden digest or have its diff explained in review. The harness

runs on a fixed arithmetic wave rather than a seeded walk: a seeded generator is one more dependency between CI and a reproducible digest.

Alongside replay, the kernel is property-tested [13] over random order sequences:

(I1) $A = 1 \Longrightarrow N \leq C_{trade}$;

(I2) $A = 1 \Longrightarrow V_v + N \leq C_v$ *for all* v;

(I3) $A = 0 \Longrightarrow g^* < \infty \wedge r \neq \varepsilon$;

(I4) $\mathcal{K}(o, \lambda, s) = \mathcal{K}(o, \lambda, s)$.

(I1)–(I2) are safety, (I3) explainability (a veto that cannot name its gate is a bug, not a rejection), and (I4) is what makes the other three testable at all.

8 Performance Functionals

Realised edge is reconstructed from `order` records; there is no fill record until the execution router exists. For UTC day j, $X_j = \Sigma_{i \in \text{day } j} e_i N_i$, day return $r_j = X_j / E_0$, and equity recursion $E_j = E_{j-1} + X_j$.

A day joins the series when the ledger holds an `order` or a kernel `veto` for it. A day whose only records are score-threshold skips is not a day: the strategy declined before the risk system was consulted, and counting it would dilute the dispersion estimate with inactivity. A day with a veto and no order does appear, at $r_j = 0$: the system was awake, submitted, and was refused, which is a real observation about the mandate.

With running peak $P_j = \max_{k \leq j} E_k$, maximum drawdown is $\text{MDD} = \max_j (P_j - E_j)/P_j$ [6], walked in the same day order the page draws. The annualised Sharpe ratio is

$$SR = (\mu/\sigma)\sqrt{252}, \quad \mu = (1/n)\Sigma_{j=1}^{n} r_j, \quad \sigma = \sqrt{((1/(n-1))\,\Sigma_{j=1}^{n} (r_j - \mu)^2},$$

returning 0 for $n < 2$, a deliberate refusal, since one observation has no deviation and a ratio invented from it is exactly the class of number this paper exists to eliminate [4, 5]. Win rate is $WR = (1/n)\Sigma_j \mathbb{1}[\, r_j > 0 \,]$ over the same days. Fills, vetoes and skips are counted in three registers: a high veto rate means the mandate binds too tightly, a high skip rate means the strategy has no edge, and conflating them makes the two indistinguishable.

For each gate the page renders $u_k = \mathrm{clamp}_{[0,1]}(\, \mathrm{used}_k / \mathrm{limit}_k \,)$; $u_k > 1$ before clamping means the limit is breached, which is a veto rather than a warning. Venue utilisation is the *worst* venue, $u_{\mathrm{venue}} = \max_v V_v / C_v$, since one exhausted venue blocks the next order while a mean would conceal it.

8.1 Tail functionals

A cap is a statement about the tail, so it should be expressed as one. Let L be the loss of the book over a fixed horizon. A functional ρ is **coherent** [16] when it is monotone ($L_1 \le L_2 \Rightarrow \rho(L_1) \le \rho(L_2)$), translation-invariant ($\rho(L + c) = \rho(L) + c$), positively homogeneous ($\rho(tL) = t\rho(L)$, $t \ge 0$) and subadditive ($\rho(L_1 + L_2) \le \rho(L_1) + \rho(L_2)$). Subadditivity is the one that matters: consolidating two venues never looks riskier than reporting them separately, which is what makes a *single* venue cap sound rather than merely conservative. Value-at-risk fails it; expected shortfall does not. Venue exposure is therefore defined by the Rockafellar–Uryasev representation

$$\mathit{CVaR}_{\alpha}(L) = \min_{t \in \mathbb{R}} \{\, t + (1/(1-\alpha))\, \mathbb{E}[\,(L - t)^{+}\,] \,\},$$

coherent for $\alpha \in (0,1)$ and equal to the mean of the worst $1-\alpha$ fraction of outcomes. The venue cap then constrains the functional rather than a notional, $\mathrm{CVaR}_{\alpha}(L_v) \le \kappa_v$ with $u^{\mathrm{tail}}_v = \mathrm{CVaR}_{\alpha}(L_v)/\kappa_v$, and the gate that fires is the worst venue's u^{tail}_v, exactly as in the notional case, the notional cap being the degenerate $\alpha \to 1$ instance of the same rule. Two properties follow without further assumption: subadditivity means enforcing κ_v per venue bounds the book's total tail by $\Sigma_v \kappa_v$, and positive homogeneity means doubling every position doubles the measured tail, so the diagnostic scales with exposure instead of saturating like a probability.

9 Custody and the Session-Key Boundary

This section specifies the boundary the design relies on. As recorded in §11(i) it is a specification: the session key, the permission module and the signer are not deployed.

The user's smart account grants the agent a session key with permission set P covering trading venues and a maximum order size, and nothing else, in the manner of the Safe smart account and its Zodiac Roles modifier [14]. The admitted set is

$$\mathcal{A}(P) = \{\, \mathrm{order}(v, N) : v \in V_{\mathrm{trade}},\ N \le C_{\max} \,\}, \quad \{\, \mathrm{withdraw}, \mathrm{approve}, \mathrm{transfer} \,\} \cap \mathcal{A}(P) = \varnothing.$$

> **Proposition 4 (Non-custody).**
>
> Under the adversary of §2, no sequence of calls available to the agent transfers value out of the user's account to an address the user has not authorised.
>
> *Proof.* Any state-changing call the agent can make must be signed by the session key and must fall in $\mathcal{A}(P)$, which by construction contains only order placements at permitted venues with bounded notional. Order placement moves funds between the account and a venue's settlement contract through the router, so the *balance* is not invariant, the *set of counterparties* is. A settlement contract receives funds and returns them on fill or refund; it is not an address the agent chose, and the agent cannot compose a call paying an arbitrary address, because such a call is not in $\mathcal{A}(P)$ and the permission module rejects it before execution. ∎

The proposition bounds what the agent can do; it says nothing about what a venue's contract does with funds while it holds them (protocol risk, accepted by routing there) or about the fill price (market risk). Revocation is a user-signed transaction, not an API call: after it confirms, $\mathcal{A}(P) = \varnothing$ and any in-flight order fails onchain. The engine's cooperation is a latency optimisation, not a security property. The signed order is EIP-712 typed data [9], so what the account authorises and what the engine intended are the same object rather than two renderings of it.

10 Launch Discovery Is Not a Price Source

The system monitors a token launchpad for new deployments. That is a *discovery* feed: its adapter emits launch events, not probabilities, and it has no route to the signer. A launch becomes a candidate only by clearing a screening score,

$$S(L) = 55 \min(1, \Lambda/10^6) + 25 \min(1, \eta/500) + 20 \min(1, v/(2\cdot 10^6)) \ge 70,$$

with Λ first-pool liquidity, η holder count and v 24-hour volume. The weighting encodes one opinion: liquidity cannot be faked cheaply at activation, holders can be airdropped, and volume can be washed. The score is a pure function of the launch record, so a replay reproduces it. Activations are *candidates*, and a

candidate still passes §4 and §5 in full: discovery cannot widen the agent's authority because it has no path to the signer. An unconfigured adapter returns a distinguishable error rather than an empty list, because a broken feed and a quiet feed must not look the same.

11 Implementation, Limitations, Conclusion

11.1 Implementation

The implementation is Go: one language for the web tier, engine and kernel, removing a class of serialisation mismatch. Dependencies beyond the standard library are `chi`, `templ`, `pgx`, `go-oidc` / `siwe-go`, `rapid` and `embedded-postgres` (integration tests only). The packages carrying the paper's claims are `internal/risk` (§5: property tests plus an import-purity check), `internal/engine` (§4), `internal/ledger` (§6), `internal/replay` (§7: golden digest on a 30-day synthetic recording), `internal/performance` (§8), `internal/loop` (§4–§8 with an injected clock), `internal/ingest` (staleness and divergence), `internal/launchpad` (§10) and `internal/events` (§3 backpressure).

Three details are load-bearing, each an architecture decision [15]: the **ledger has no clock** (timestamps are supplied, or the digest would differ every run); the **paper account holds a position for a fixed number of ticks and realises at the model's own estimate**, because the engine has no exit rule and the account must not invent one; and the **signer is reachable from the engine only**: a network policy outside the code in the deployed form, and in the single binary a property of the import graph, where `risk.Evaluate` has exactly one caller outside its own tests.

11.2 Limitations

(a) Drawdown semantics.

The kernel compares d against $\delta = 1$ (total loss of peak) while product copy states a 6.00% halt; until they are reconciled the user-facing figure is the mandate's, not the kernel's: a correctness gap, not a presentation choice.

(b) Display determinism.

Worst-venue utilisation iterates a Go map, so among tied venues the displayed name can vary across runs; values and ledger are unaffected.

(c) Input integrity.

By A4 the kernel cannot validate its own measurements; purity buys reproducibility, not integrity.

(d) Cost model.

Borrow, holding period and slippage are assumptions the score inherits.

(e) No exit rule.

PnL describes the model's fair value, not achieved fills.

(f) Production figures are not yet ledger-derived.

The landing surface quotes static values and limit read-outs inconsistent with the mandate of §5.3: an explicit pre-launch gate.

(g) No formal verification.

Property tests sample 100,000 cases each; the five-gate conjunction is small enough for a machine-checked proof.

(h) Jurisdiction.

Prediction-market access depends on the user's region and the venue's terms.

(i) The custody boundary is specified, not deployed.

There is no signer process and no permission module in the repository, so the honest reading of the current build is a paper-trading system with a pure risk kernel and an append-only ledger.

11.3 Conclusion

The three trust-critical claims are architectural rather than promissory. The decision path is a pure function, so any trade replays to a byte-identical ledger. The kernel is five indicator gates whose approved set is S_5-invariant, so approval is order-invariant while attribution is explicit, versioned and reproducible. Sizing is the constrained Kelly program over those same caps, so limit and size rule cannot diverge. The ledger is a prefix-free length-encoded SHA-256 chain, so any edit propagates to every later digest. Performance figures are functionals over that ledger, venue exposure stated as a coherent tail functional, so a displayed number and its derivation cannot diverge without the divergence being a detectable bug. The custody boundary, once deployed, bounds the blast radius of a compromised engine by a rule the attacker cannot edit. The open problems are small: a constant to reconcile, a tie-break to fix, a proof to mechanise, a boundary to deploy.

References

[1] NIST. *FIPS PUB 180-4: Secure Hash Standard*. 2015.

[2] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008.

[3] R. C. Merkle. A certified digital signature. *CRYPTO '89*, LNCS 435, 1990.

[4] W. F. Sharpe. Mutual fund performance. *Journal of Business* 39(1):119–138, 1966.

[5] A. W. Lo. The statistics of Sharpe ratios. *Financial Analysts Journal* 58(4):36–52, 2002.

[6] M. Magdon-Ismail, A. Atiya. Maximum drawdown. *Risk* 17(10):99–102, 2004.

[7] R. Cont. Empirical properties of asset returns. *Quantitative Finance* 1(2):223–236, 2001.

[8] Spruce Systems. *EIP-4361: Sign-In with Ethereum*. 2021.

[9] Ethereum Improvement Proposals. *EIP-712: Typed Structured Data Hashing and Signing*.

[10] WHATWG. *HTML Living Standard*, § Server-sent events.

[11] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, UC Irvine, 2000.

[12] a91. *templ: a language for writing HTML user interfaces in Go*.

[13] pgregory. *rapid: property testing for Go*.

[14] Safe Ecosystem Foundation. *Safe smart account* and *Zodiac Roles modifier*.

[15] allto Labs. *Architecture Decision Records 0001–0005*. `docs/adr/`.

[16] P. Artzner, F. Delbaen, J.-M. Eber, D. Heath. Coherent measures of risk. *Mathematical Finance* 9(3):203–228, 1999.

This document describes a system under development. Figures quoted for returns, drawdown and Sharpe ratio are illustrative and simulated until the ledger switch in §11(f) is complete. Nothing here is investment advice. Trading digital assets involves substantial risk of loss, including total loss of capital.