delta: Enabling Agentic Commerce with Guardrails

Executive Summary

delta unlocks agent-powered financial workflows with the guardrails required to accelerate a production grade rollout.

- By integrating with delta, platforms can offer their users:
    - Access to digitally native assets (e.g. stablecoins) and payment rails with global-by default reach and 24/7 availability. Users get instant transaction confirmations, and platforms can batch settle outcomes to a shared ledger programmably or on-demand.
    - **The ability to deploy agents that leverage delta's shared state network to facilitate complex, multi-party financial workflows, safely**.
      Agent employers can define and commit to any settlement conditions applied to any program – these serve as guardrails that protect against outlier, negative outcomes. Because they are anchored to delta's shared state network, settlement conditions associated with an agent are **independently verifiable**, allowing counterparties to interoperate with them without requiring trust or mutual audits.
- **Business impact**:
    - With these capabilities, businesses can significantly accelerate the transition from generative AI for research purposes to agentic AI that can manage financial workflows autonomously and interoperate with other businesses, safely. Because the settlement conditions imposed on these agents are independently verifiable**,** each new user that publishes a guardrailed workflow to delta can safely interoperate with other users immediately, creating a compounding network of approved counterparty agents and a compatibility moat that accelerates market-share capture for early adopters.
- **Why now?**
    - AI agents have the potential to add trillions of "virtual hands" to the workforce across the front, middle, and back-offices of businesses. Without

machine-checkable guardrails, adoption will be slow and could stall out at "human-in-the-loop" levels of efficiency gains. Integrating agents with delta provides the missing safety layer to unlock autonomous financial workflows.

AI agents cannot become commercially useful without verifiable guardrails

Problem Statement #1:

Software does not have surface-visible, machine-checkable safeguards.

Every software system consists of an interior and an interface. We can visualize this as a box, where only the surface of the box is visible to the people interacting with it. The interior may be full of tests, asserts, and permissions. But the interface almost never exposes surface-visible, machine-checkable safeguards – guarantees that an untrusting counterparty can rely on in real time. Instead, we substitute trust, contracts, and auditors. In the era of human-administered software, this shortcoming has been costly but largely palatable.

We are now entering the era of AI-administered software.

Without independently verifiable guardrails, software 3.0 will require humans checking and approving every action. It's not safe for companies to outsource high-stakes workflows to agents without guardrails that protect against downside outlier or materially negative outcomes. **Without these guardrails, the rollout of autonomous, agentic commerce will be extremely slow.** At best, it will look much like autonomous vehicles, i.e. a decade-long gap between "this works" and "this is safe to roll-out in production". At worst, we may never break through the human-in-the-loop stage of transformation.

Problem Statement #2:

Blockchains introduced verifiable compute, but they cannot support modern applications, including LLMs:

1. *Capacity-constrained:* blockchains cannot support the size and throughput that LLMs and modern apps require.

2. *Forced determinism:* you cannot program non-deterministic logic into smart contracts. This means no temperature or sampling.

3. *No external connectivity:* It's impossible to fetch external data without using oracles, which are slow and expensive. This means no RAG, no tool use, and no async

workflows.

4. *No autonomous workflows:* smart contracts don't pull or react to events on their own; they rely on keepers.

An example of this is the recently announced AP2 protocol by Google, which leverages blockchains for payments and was developed for the specific problem of agentic commerce. Following protocol, a user can provide its agent with instructions for what to buy and at what price, with a cryptographic commitment sent to be verified by the merchant once the agent requests a purchase.

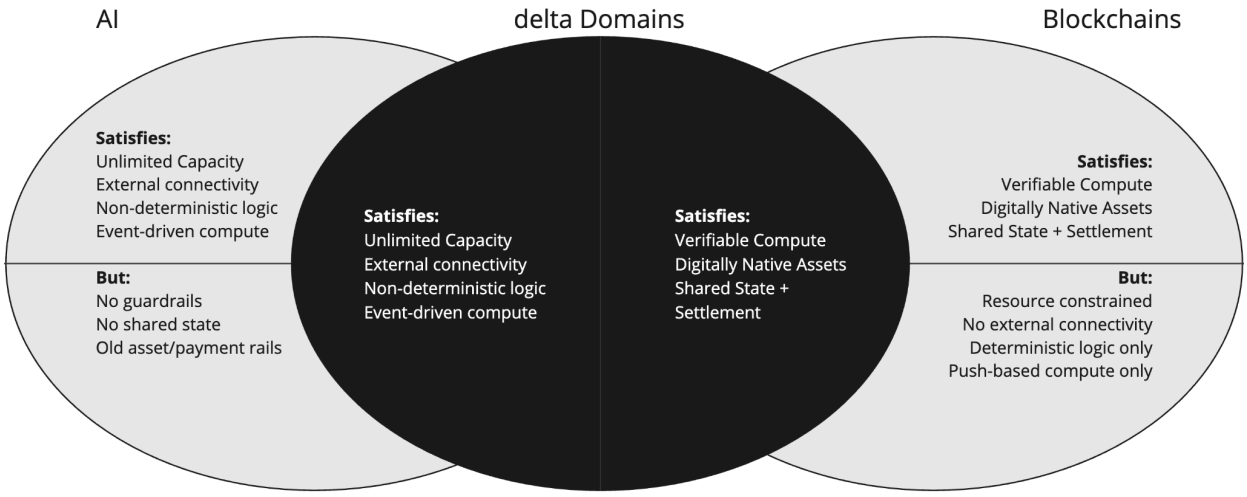
This commitment **does not constitute guardrails,** as it does not impact settlement but rather only exists for record-keeping in cases of dispute between merchant and customer. Worse yet, it's impossible to generalize this protocol to arbitrary agentic interactions (i.e. to provide support for programmable guardrails): in the current world, protection against misbehaving agents is limited to disputes which in any case require human resolution, and only to a very limited set of actions.

Enter delta: A settlement network with the requisite safeguards to enable 24/7 agentic interactions in every sector of the global economy

Every autonomous economic interaction will have to take place on a network like delta for it to be secure. delta is the first such network, and delta domains are the first tool that will allow businesses to implement autonomous financial workflow agents with the requisite, independently verifiable guardrails for safety.

- With delta, arbitrary verifiable guardrails can be added to any agent. These guardrails act as outcome-based settlement rules, and can be independently verified by any prospective counterparty. Any transaction that violates one of these conditions cannot be finalized.

- delta is the only verifiable, shared-state system that can provide these guarantees while being compatible with modern compute stacks and applications.



By writing arbitrary constraint programs over the inputs and outputs of the agent, we can encode cryptographic constraints to prevent misbehavior. This is exactly what delta's "local laws" enable. A delta-based agent binds its local laws to the secure, transparent delta ledger, so everyone else can independently verify the constraints as they interact with the agent.

The combination of a neutral, shared state network with the ability to add arbitrary verifiable guardrails is both necessary and sufficient for secure, autonomously acting AI agents. With it, the *employer* of the AI can provide guarantees against improper behavior — both toward the employer and toward other counterparties. Those counterparties can independently verify that the guarantees are present *before* engaging.

- Note: in the blockchain world, "verifiability" is often used as a standalone term. We find it vague. We use "verifiable guardrails," which captures the purpose: to provide independently checkable guarantees to counterparties.

Why now?

AI agents represent the potential for trillions of new, virtual hands that can enter our workforce. But without safeguards that protect against outlier downside outcomes, they cannot be trusted to operate high-stakes services/workflows autonomously.

Verifiable guardrails provide security guarantees such that the **only** limitation on what an agent can do becomes its capabilities. Given the rate of improvement in AI combined with guardrails provided by delta, we could significantly accelerate the adoption of autonomous, agent-powered workflows that transform business operations across every sector.

Next Steps

delta adds the missing safety layer for autonomous, money-moving workflows. Without delta, the rollout of agents that handle money movements will be a slow, decade-long process, much like autonomous vehicles. With delta, we can safely implement agentic, money-moving workflows in the next 6 months.

We're actively exploring pilots with external design partners to demonstrate these capabilities in production. If you're interested in learning more, please reach out to myles@delta.network or ole@delta.network.

Appendix: Distribution and Implementation Examples

Agent Factory SDK for Partners

We will distribute this capability as a **white-label Agent Factory SDK** that any platform can embed. The SDK provides:

- **Authoring UI + Compiler** from English → Strategy Spec → Local-Law config.
- **Agent Wallets & Rails** for delta-settled actions (with caps/timelocks).
- **Execution/Settlement Hooks** (pre-check, SDL append, proof submit, receipt handling).
- **Receipts API** so customers, counterparties, and auditors can verify outcomes independently.

ESC enables **agents that can listen to instructions and transact safely on behalf of users.** These are superpowers that otherwise wouldn't be available for years, and represent a signficant capability advantage for early adopters. As more partners publish guardrailed workflows, the compatibility moat compounds: integrate once, interoperate safely everywhere on delta.

Explore the Examples

Because delta is compatible with any existing system, any company in the world can integrate agents that transact on behalf of users or their employees. In order to add these capabilities, the only requirements are:

- Integrating their backend with a delta domain
- Integrating support for stablecoins that settle on delta
- Optional: Integrating our English Smart Contract (ESC) platform — this will allow non-technical users or employees to set up and deploy agents with verifiable guardrails

Below are a few examples of agent capabilities integrated into well-known categories, each of which will be possible within the next 6 months. We purposely selected “single player” use case scenarios, as these will be possible by a single company integrating with a domain and stablecoins on delta.

Sports Betting Platforms

1) Overview

Feature:

Power users author persistent, in-play strategies in plain English (ESC). Agents watch licensed feeds and act instantly (cash-outs, in-play bets, futures) under user budgets and risk constraints. This upgrades “tap fast” into “think once, run always,” improving fill quality and discipline.

Security:

- **Error protection:** budget caps, feed freshness/quorum, event windows, and per-market exposure limits prevent fat-finger or model drift.
- **Prompt-injection defense:** scope lock to user-approved markets/teams; ignore free-text in feeds; immutable `strategy_hash` and versioned configs.
- **Settlement safety:** actions pre-checked at execution but **only finalize** if Local Laws (and venue rules) are proven at settlement.

2) Examples

A. English prompt	B. Strategy Spec (derived)	C. Data inputs	D. Local Laws (guardrails)
“If my bet on Lakers live spread loses 30% of potential payout, auto cash-out. If it gains 50%+, lock profit and ping me.”	• Monitor `bet_id` live valuation every 1s • Trigger cash-out when drawdown ≥30% or gain ≥50% • Notify user on execution	• Official odds/valuation feed (allowlist) • Timestamps, provider IDs • User position registry	• `bet_id ∈ user_positions` • `valuation freshness ≤2s` and two-feed quorum • `cash_out ≤ quoted_liquidity` • `per-day cashouts ≤ N` • Budget/time windows; immutable

			`strategy_hash`; ignore feed free-text

A. English prompt	B. Strategy Spec (derived)	C. Data inputs	D. Local Laws (guardrails)
“If an odds boost makes my SGP EV > 5%, stake $30.”	• Poll boosts each minute • Compute EV (model v*) • Place stake $30 when EV≥5%	• Boost catalog feed (allowlist) • Odds legs & correlations • Model version hash	• `offer.source ∈ allowlist` • `EV(model=v*) ≥ 0.05` • `stake == $30` • `per-market exposure ≤ cap` • Anti-tamper: inputs must be structured data; no control flow from free-text

Ticket Sale Platforms

1) Overview

Feature:

Users declare purchase/resale rules (price ceilings, section/row, seller rating). Agents auto-buy on drops and relist per schedule. If assets are tokenized on delta, DvP, royalties, and refunds (on cancellation) are automatic.

Security:

- **Error protection:** per-event caps, seat quality filters, total-cost ceilings (incl. fees/shipping), resale floors.
- **Prompt-injection defense:** event/section whitelist; feed-quorum on price/availability; immutable spec.
- **Settlement:** execution pre-check; final transfer/refund **only** on proven adherence to Local Laws.

2) Examples

A. English prompt	B. Strategy Spec	C. Data inputs	D. Local Laws
“Buy 2 lower-bowl tickets for Friday ≤ $180 each (all-in). Auto-list at $240 if bought.”	• Monitor event E inventory • Filter “lower-bowl” seats • Purchase ≤$180 all-in for qty 2 • Relist at $240	• Inventory/price feed • Fee & tax calculator • Seller rating	• `price_total ≤ 180` • `section ∈ lower_bowl` • `qty ≤ 2` and per-event cap • Seller rating ≥ threshold • Auto-relist rule; refund on event cancel

A. English prompt	B. Strategy Spec	C. Data inputs	D. Local Laws
“If price drops ≥15% vs 7-day average, buy 1; relist at +20%.”	• Compute rolling 7-day avg • Trigger on drop ≥15% • Buy qty 1; relist +20%	• Historical price feed • Current listings • Fee model	• `drop_pct ≥ 15%` (feed quorum) • Budget/day cap • Relist floor ≥ fees + margin • Anti-wash-trade checks

Prediction Markets

1) Overview

Feature:

Users encode conditional trading rules (news thresholds, poll spreads, cross-market signals). Agents place or unwind positions under exposure and EV constraints.

Security:

- **Error protection:** per-market exposure, portfolio VaR caps, time windows, data quorum.
- **Prompt-injection defense:** feed allowlist; deny free-text control; immutable spec versions.
- **Settlement:** actions pre-checked; fills finalize only if proofs of Local Laws pass.

2) Examples

A. English prompt	B. Strategy Spec	C. Data inputs	D. Local Laws
"If candidate spread > +2.5 in polling average, buy YES ≤ 45%."	• Compute poll avg hourly • Bid YES when spread > +2.5 and price ≤45%	• Poll aggregator feed • Market order book	• Feed quorum & freshness • `price ≤ 0.45` • `exposure_market ≤ cap` • Daily trade count ≤ N

A. English prompt	B. Strategy Spec	C. Data inputs	D. Local Laws
"Provide liquidity ±2% around fair with inventory cap $500."	• Quote banded prices • Rebalance when drift >Δ • Cap inventory	• Market price/vol • Inventory tracker	• `inventory_abs ≤ 500` • Quote width ≥ min_spread

			• Rate limits; cancel-on-loss x%