AEX-402

Agentic Exchange Protocol

Toward Autonomous Liquidity Infrastructure
for the x402 Economy

A Theoretical Framework and Research Agenda for
Adaptive Market Making with Reinforcement Learning

Rin Fhenzig

OpenSVM Research

research@opensvm.com

December 25, 2025

Note: This paper distinguishes proven results (Theorems, Propositions) from research directions (Conjectures, Open Problems). Claims about the x402 integration describe proposed designs, not implemented systems.

Abstract

We present AEX-402, a theoretical framework for autonomous market making designed for the emerging agentic economy. As Autonomous Economic Agents (AEAs) increasingly transact via protocols like A2A and x402, they require liquidity infrastructure with properties current AMMs cannot provide: self-optimization, formal guarantees, programmatic discovery, and agent-to-agent negotiation.

This paper makes contributions at three levels:

Rigorous contributions. We introduce the *Generalized Bonding Curve* (GBC) family—a mathematically valid parameterization that interpolates between constant product and concentrated liquidity through a single curvature parameter $\kappa \in [0, 1]$. We prove efficient swap computation via Newton-Raphson iteration with $O(\log \log(1/\epsilon))$ convergence. We establish a tight $\Omega(T)$ welfare loss lower bound for static curves under regime-switching markets.

Agent-theoretic framework. Drawing on the AEA literature, we model how autonomous agents discover pools via on-chain registries (AEAMCP), evaluate trades through marginal utility computation, and communicate via A2A protocols. We analyze agent decision-making under uncertainty and characterize the game-theoretic equilibrium of multi-agent trading.

Research directions. We propose Q-learning for on-chain parameter adaptation, identifying the *tracking-vs-learning tradeoff* as the central theoretical obstacle in non-stationary environments. We outline mechanism design for tiered access with incentive compatibility proofs under price-taking assumptions.

Virtual Pool Graduation. We introduce a zero-rent token launch mechanism using bonding curves with anti-manipulation protections. Tokens exist virtually until reaching a *dynamic graduation target* that penalizes pump-and-dump behavior through churn detection. We prove the graduation game has a unique subgame-perfect equilibrium where manipulation is unprofitable, enabling 1.49 million holders per pool with only 7 bytes per holder through novel hash compression.

We are explicit about limitations: convergence conjectures await rigorous proof; the agent registry integration is at design stage; equilibrium characterization remains open. This paper is both a technical contribution and a research agenda for adaptive, agent-native decentralized finance.

Keywords: Automated Market Makers, Autonomous Economic Agents, Reinforcement Learning, Mechanism Design, A2A Protocol, x402 Protocol, Bonding Curves, Multi-Agent Systems

JEL Classification: D47, G14, C73

MSC Classification: 91B26, 68T05, 91A25

Contents

1 Introduction

1.1 The Thesis

When Uniswap launched in 2018, its creators faced a fundamental question: what exchange rate should a decentralized pool offer? Their answer—the constant product formula $xy = k$—was elegant, simple, and consequential. It worked remarkably well for general-purpose trading. But it also locked in an assumption: that all token pairs should trade on the same curve, regardless of whether they're volatile cryptocurrencies or stablecoins that should hover near parity.

Curve Finance arrived two years later with a different answer for stablecoins: a flatter curve optimized for pairs expected to trade 1:1. This was better for stablecoins but worse for assets that might diverge. Then came Uniswap v3 with concentrated liquidity, Balancer with weighted pools, and a proliferation of specialized AMM designs. Each optimizes for different assumptions about market behavior.

Here lies the problem: **markets change, but curves don't**. A pool launched during stable conditions might face a depeg event. A pool designed for correlated assets might see correlation break down. The curve that was optimal yesterday may be suboptimal tomorrow. And yet, changing a pool's fundamental parameters requires governance proposals, community votes, and time—luxuries unavailable during a crisis.

This paper advances a simple thesis: **automated market makers should learn from experience**. Current AMMs use fixed exchange functions chosen at deployment—constant product for Uniswap, StableSwap for Curve, concentrated liquidity for Uniswap v3. These choices embed assumptions about asset behavior that may not hold as markets evolve. We argue that AMMs should adapt their exchange functions in response to observed market conditions, and we provide theoretical foundations for this adaptation.

The thesis is not that adaptation is easy or that we have solved all problems. Rather, we aim to:

1. Formalize what "adaptive AMM" means mathematically
2. Prove that adaptation can, in principle, outperform any static design
3. Identify the key technical challenges in achieving practical adaptation
4. Propose concrete mechanisms and analyze their properties
5. Be explicit about what remains unproven or speculative

1.2 Why Now: The Agentic Economy

Our interest in adaptive AMMs is motivated by a broader transformation: the emergence of autonomous agents as economic actors. Consider a future—perhaps nearer than we think—where millions of AI agents operate continuously, making thousands of economic decisions per day. An agent monitoring DeFi yields might rebalance a portfolio hourly. A content-generation agent might pay for compute resources every few seconds. A trading agent might seek the best exchange rate across dozens of venues in milliseconds.

These agents don't sleep. They don't take weekends off. They don't wait for governance proposals to pass. They operate at machine speed, and they need infrastructure that operates at machine speed too.

1.3 Autonomous Economic Agents: A Framework

To understand what adaptive AMMs must serve, we must understand what autonomous economic agents *are*. The concept has been formalized in the multi-agent systems literature [Minarsch et al., 2020], where an Autonomous Economic Agent (AEA) is defined as:

> *An intelligent agent that acts on its owner's behalf, with limited or no interference, and whose goal is to generate economic value for its owner.*

This definition is precise about what AEAs are—and what they are not. They are *not* general-purpose assistants handling any owner need. They are *not* digital twins mirroring personality. They are *not* passive APIs or smart contracts (which lack autonomous agency). And they are certainly *not* AGI systems. Rather, AEAs are narrowly scoped economic actors: software that pursues wealth generation within well-defined parameters.

What makes AEAs distinctive as agents? Three properties:

- **Autonomy**: They operate independently of constant owner input, making decisions aligned with prescribed goals
- **Proactivity**: They take initiative rather than merely responding to stimuli, actively seeking opportunities
- **Self-interest**: They prioritize owner interests over broader system welfare—a feature, not a bug, that enables incentive-aligned design

The AEA framework [Minarsch et al., 2020] operationalizes these properties through a modular architecture. An AEA comprises *Skills* (business logic), *Connections* (interfaces to networks and APIs), *Protocols* (message formats and dialogue rules), and *Contracts* (smart contract wrappers). The agent's "economic brain"—called the DecisionMaker—maintains a wallet, evaluates transactions against owner preferences, and computes marginal utility scores to decide which trades to accept.

This architecture has implications for AMM design. When an AEA approaches a liquidity pool, it doesn't "click swap" like a human. It:

1. Queries the pool's current state via a Connection
2. Evaluates the proposed trade against its Preferences (expected utility vs. cost)
3. If favorable, constructs a transaction using the appropriate Protocol
4. Submits via its blockchain Connection and records the outcome

Each step happens programmatically, in milliseconds. Multiply this by millions of agents making thousands of decisions daily, and you see why AMM infrastructure matters: it's the substrate on which an entire economy runs.

1.4 Multi-Agent Systems and Trust

AEAs don't operate in isolation. They interact with other agents in what computer scientists call a Multi-Agent System (MAS)—a decentralized system where autonomous agents coordinate to achieve goals. MAS are inherently complex: each agent pursues its own objectives, conflicts of interest arise, agents operate asynchronously, and without central coordination, uncertainty pervades.

How do we build reliable infrastructure in such an environment? The answer from distributed systems theory is *Byzantine fault tolerance* (BFT) [Lamport et al., 1982]. Rather than trusting any single agent, we design protocols where correctness is guaranteed as long as fewer than one-third of participants are faulty or malicious. Specifically, with $N = 3f + 1$ agents, we can tolerate up to f failures while maintaining consensus.

This creates what's called a *trust-minimized system*: trust isn't eliminated (some assumptions are always necessary), but it's distributed across many independent parties. No single agent—or small coalition—can corrupt the system.

How does this relate to AMMs? Current AMMs are already trust-minimized in one sense: the smart contract enforces the exchange invariant regardless of who submits transactions. But they're *not* adaptive—the invariant is fixed at deployment. Our proposal extends trust-minimization to the adaptation layer: the Q-learning mechanism that adjusts curve parameters should itself be resistant to manipulation by any small subset of traders.

1.5 On-Chain Agent Discovery

Before agents can trade, they must *find* each other. In traditional systems, this means centralized directories—single points of failure and censorship. The decentralized alternative is on-chain agent registries.

The AEAMCP project [OpenSVM, 2025] demonstrates this pattern on Solana. It provides two interconnected registries: an **Agent Registry** for autonomous agents following AEA and A2A paradigms, and an **MCP Server Registry** for Model Context Protocol servers providing tools and resources.

The architecture uses a *hybrid storage model*. Essential data—identifiers, ownership, status, service endpoints—lives on-chain in Program Derived Addresses (PDAs). Extended metadata—full capability schemas, documentation—lives off-chain with cryptographic hashes stored on-chain for integrity verification. This balances cost (Solana rent is expensive for large data) with verifiability (hashes prove off-chain data hasn't been tampered with).

Each registry entry includes an owner authority (controlling modifications), service endpoints (where the agent can be contacted), capability flags, and an extended metadata URI linking to full specifications. Discovery happens through event-driven off-chain indexing: the registry emits events for all state changes, which indexers process into queryable databases.

How does this relate to AEX-402? Our adaptive AMM pools could themselves be registered as agents—discoverable by any AEA seeking liquidity services. The pool's service endpoint would be the X402-enabled HTTP API. Its capabilities would include "swap", "add_liquidity", "oracle_query". Other agents could discover it, query its current κ, and decide whether to trade.

1.6 The A2A and x402 Protocols

Two protocols form the communication backbone for agent commerce.

The **A2A Protocol** (Agent-to-Agent) [Linux Foundation, 2025], originally developed by Google and donated to the Linux Foundation, provides a standard for agent-to-agent communication. It enables agents to interact without exposing internal memory, tools, or proprietary logic—maintaining security while enabling collaboration. A2A complements MCP (Model Context Protocol): MCP standardizes agent-to-tool communication; A2A standardizes agent-to-agent communication. Together they form a complete interoperability stack.

The X402 protocol [x402 Foundation, 2025], launched by Coinbase and partners in 2025, activates HTTP status code 402 ("Payment Required") for internet-native payments. This enables software agents to pay for resources—compute, data, services—directly over HTTP without accounts or human intervention. It's the plumbing that makes agent-to-agent commerce possible.

When agents can pay, they will also need to exchange. An agent earning revenue in USDC may need to pay for services in SOL. An arbitrage agent may need to swap across multiple venues. A portfolio management agent may need to rebalance positions. These agents will demand exchange infrastructure matching their characteristics:

- **Programmatic access**: HTTP APIs rather than wallet connections
- **Predictable execution**: Bounded slippage and formal guarantees
- **Self-optimization**: No reliance on human governance
- **Tiered service**: Ability to pay for priority execution

Current AMMs were designed for human users clicking buttons in wallet interfaces. Adaptive AMMs could be designed for agents making API calls at millisecond intervals. The difference matters: agents can process information faster, but they also have less tolerance for unpredictability. An agent that expects 0.05% slippage but receives 0.5% has wasted 10x its expected cost—and unlike a human, it can't shrug and try again later.

1.7 Paper Structure and Honesty Policy

This paper contains both rigorous results and speculative proposals. We maintain strict separation:

Proven results are labeled as Theorems, Propositions, or Lemmas with complete proofs or citations to the literature.

Research directions are labeled as Conjectures or Open Problems. These represent our best current thinking but require further work.

Design proposals (particularly for X402 integration) describe systems we believe should be built, not systems that currently exist.

We believe this honesty policy serves readers better than the common practice of presenting conjectures as theorems or designs as implementations.

1.8 Summary of Contributions

Section 2: We prove that static AMMs suffer fundamental limitations. Under regime-switching market conditions, any fixed curve accumulates $\Omega(T)$ welfare loss over time horizon T. This is tight: we construct markets where every static choice is suboptimal half the time.

Section 3: We introduce the Generalized Bonding Curve (GBC) family, a mathematically valid parameterization interpolating between curve types. Unlike naive blending (which is ill-defined), GBC uses a single curvature parameter κ to control the tradeoff between capital efficiency and robustness. We prove swap computation is efficient.

Section 4: We formulate adaptive curve selection as reinforcement learning. We identify the tracking-vs-learning tradeoff as the central challenge: learning requires stability (to converge) while adaptation requires responsiveness (to track changing optima). We state convergence conjectures and identify what would be needed to prove them.

Section 5: We analyze mechanism design for tiered access. Agents can pay for execution guarantees via X402. We prove incentive compatibility under price-taking assumptions and identify open problems for strategic settings.

Section 7: We describe the proposed X402 integration architecture. This section is explicitly a design proposal, not an implementation report.

Section 8: We analyze how autonomous economic agents make trading decisions, drawing on the AEA framework [Minarsch et al., 2020]. We model agent utility, preferences, communication protocols, and game-theoretic interactions.

Section 9: We candidly discuss limitations, open problems, and what could go wrong.

1.9 System Architecture Overview

Figure 1 shows the complete AEX-402 architecture, integrating on-chain and off-chain components with the agent ecosystem.

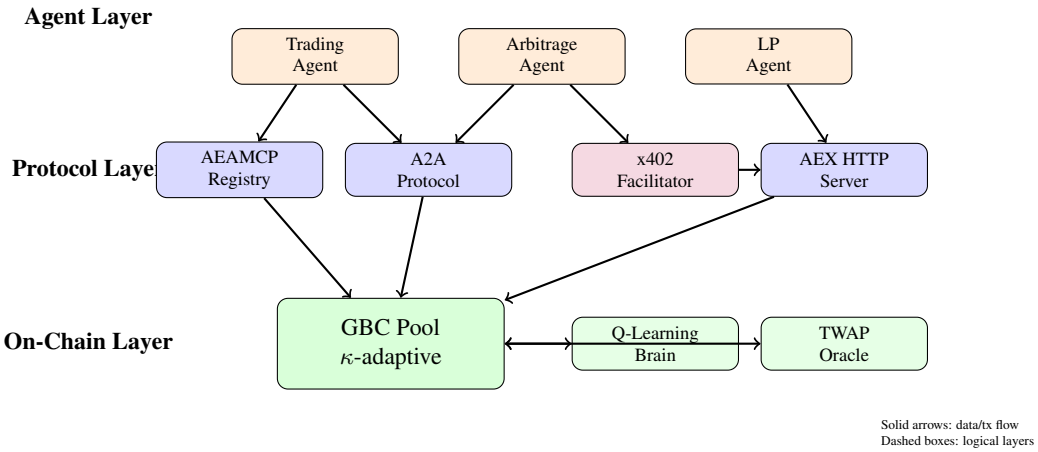

Figure 1: Complete AEX-402 system architecture. Agents discover pools via AEAMCP, communicate via A2A, pay via x402, and interact with adaptive GBC pools that learn from trading patterns.

1.10 Comparison with Existing AMMs

Table 1 compares AEX-402 with existing AMM designs across key dimensions.

Table 1: Comparison of AMM designs. AEX-402 is the only design combining adaptive curves with agent-native protocols.

Feature	Uniswap v2	Curve v1	Uniswap v3	Curve v2	AEX-402
Curve type	Constant product	StableSwap	Concentrated	Dynamic peg	GBC family
Adaptation	None	None	Manual LP	Heuristic	Q-learning
Parameters	Fixed	Fixed A	LP positions	Auto-rebalance	κ learned
Agent discovery	None	None	None	None	AEAMCP
Agent protocol	None	None	None	None	A2A
HTTP-native	No	No	No	No	x402
Tiered access	No	No	No	No	Yes
On-chain oracle	No	Limited	No	EMA	TWAP
Capital efficiency	Low	High (peg)	High (range)	Medium	Adaptive
Robustness	High	Low (depeg)	Medium	Medium	Adaptive
Compute cost	$\sim$1K CU	$\sim$3K CU	$\sim$2K CU	$\sim$5K CU	$\sim$3K CU

The key differentiator is the combination of adaptive curve selection with agent-native infrastructure. Curve v2 adapts but uses heuristics rather than learning, and no existing AMM integrates with agent discovery or communication protocols.

2 The Limits of Static AMMs

Before proposing adaptive mechanisms, we establish that adaptation is necessary. This section proves that static AMMs—those with fixed exchange functions—suffer fundamental limitations in changing markets.

The argument is intuitive: if markets alternate between two different states, and each state has a different optimal curve, then any single fixed curve must be wrong at least half the time. We formalize this intuition into a theorem showing that static curves accumulate welfare loss *linearly* over time.

2.1 Market Model

To reason precisely about adaptation, we need a model of how markets change. The simplest useful model is one where markets alternate between distinct "regimes"—periods with different statistical properties.

Think of a stablecoin pair like USDC/USDT. Most of the time, both trade near \$1.00, with small fluctuations. This is the "calm" regime: low volatility, high correlation. Occasionally, stress events cause one stablecoin to depeg—perhaps briefly during a bank run, or more persistently during a protocol failure. This is the "volatile" regime: high volatility, broken correlation.

The optimal AMM curve differs dramatically between these regimes. In calm times, liquidity providers want a flat curve (like StableSwap) that concentrates capital near the peg, earning fees without suffering impermanent loss. In volatile times, they want a robust curve (like constant product) that doesn't catastrophically misprice as the peg breaks.

We model market conditions as an exogenous stochastic process. This is a simplification: in reality, AMM behavior affects market conditions. We discuss this limitation in Section 9.

Definition 2.1 (Market Regime)**.** *A market regime $\sigma \in \Sigma$ specifies:*

- *Volatility $v(\sigma)$: expected magnitude of price changes*
- *Correlation $\rho(\sigma)$: co-movement of assets in the pool*

- *Volume $V(\sigma)$: expected trading activity*

Definition 2.2 (Regime-Switching Process)**.** *A regime-switching market alternates between regimes σ_H (high volatility, low correlation) and σ_L (low volatility, high correlation) according to a Markov chain with transition matrix:*

$$P = \begin{pmatrix} 1-p & p \\ q & 1-q \end{pmatrix} \tag{1}$$

where $p, q \in (0, 1)$ are switching probabilities.

This model captures the key phenomenon: markets alternate between "calm" periods (where stablecoins maintain peg, correlated assets move together) and "volatile" periods (where correlations break down and prices move sharply). The switching probabilities p and q control how frequently regimes change—low values mean long periods of stability, high values mean rapid alternation.

2.2 Welfare and Optimality

How do we know which curve is "better"? We need a metric that captures the interests of all participants. Traders want low slippage. Liquidity providers want high fees and low impermanent loss. Sometimes these conflict—a flat curve gives traders low slippage but exposes LPs to impermanent loss during volatility.

We resolve this by measuring *social welfare*: the sum of all participants' utilities. This is a standard approach in mechanism design, though we acknowledge it papers over distributional concerns. A pool that's excellent for traders but terrible for LPs might have high social welfare but fail to attract liquidity.

Different curves perform differently under different regimes. We measure performance by social welfare: the sum of LP returns and trader surplus.

Definition 2.3 (Social Welfare)**.** *For curve c under regime σ, the instantaneous social welfare is:*

$$W(c, \sigma) = \underbrace{f \cdot V(\sigma) - IL(c, \sigma)}_{\text{LP utility}} + \underbrace{V(\sigma) \cdot (1 - s(c, \sigma))}_{\text{trader surplus}} \tag{2}$$

where f is the fee rate, IL is impermanent loss, and s is average slippage.

Definition 2.4 (Regime-Optimal Curve)**.** *The optimal curve for regime σ is:*

$$c^*(\sigma) = \arg\max_c W(c, \sigma) \tag{3}$$

The key observation is that $c^*(\sigma_H) \neq c^*(\sigma_L)$ in general. High-volatility regimes favor robust curves (like constant product) that handle large price movements. Low-volatility regimes favor concentrated curves (like StableSwap) that provide capital efficiency near the current price.

This is not merely theoretical. Consider the UST depeg of May 2022: stablecoin pools optimized for pegged behavior experienced massive impermanent loss as UST collapsed. A pool that had switched to a more robust curve before the depeg would have protected its LPs. Of course, predicting such events is difficult—but that's precisely why we want pools that can adapt in real-time rather than waiting for governance.

2.3 The Static Welfare Bound

We now prove the main result of this section: any static curve accumulates welfare loss linearly over time. The proof is straightforward once we have the setup, but the implications are profound. It means the gap between static and optimal performance doesn't shrink as we wait longer—it grows. Static curves don't "catch up" to adaptive ones; they fall further behind.

Theorem 2.5 (Static Curve Lower Bound)**.** *Consider a regime-switching market with $p, q > 0$ (non-degenerate switching). Let $\pi_H = q/(p+q)$ and $\pi_L = p/(p+q)$ be the stationary probabilities. For any static curve c:*

$$\mathbb{E}\left[\int_0^T W(c^*(\sigma_t), \sigma_t) - W(c, \sigma_t)\, dt\right] \geq \delta \cdot T \tag{4}$$

where $\delta = \min\{\pi_H, \pi_L\} \cdot \min\{W(c^*(\sigma_H), \sigma_H) - W(c, \sigma_H), W(c^*(\sigma_L), \sigma_L) - W(c, \sigma_L)\} > 0$.

Proof. By stationarity, the process spends fraction π_H of time in regime σ_H and π_L in σ_L (in expectation, over long horizons).

For any fixed c, either $c \neq c^*(\sigma_H)$ or $c \neq c^*(\sigma_L)$ (or both), since $c^*(\sigma_H) \neq c^*(\sigma_L)$ by assumption.

Case 1: $c \neq c^*(\sigma_H)$. By definition of optimality:

$$W(c^*(\sigma_H), \sigma_H) - W(c, \sigma_H) = \epsilon_H > 0 \tag{5}$$

The expected time in regime σ_H over horizon T is $\pi_H \cdot T + o(T)$ by the ergodic theorem. Thus:

$$\mathbb{E}\left[\int_0^T \mathbf{1}[\sigma_t = \sigma_H] \cdot \epsilon_H\, dt\right] = \pi_H \cdot \epsilon_H \cdot T + o(T) \tag{6}$$

Case 2: $c \neq c^*(\sigma_L)$. Analogous argument gives $\pi_L \cdot \epsilon_L \cdot T + o(T)$.

Taking $\delta = \min\{\pi_H \epsilon_H, \pi_L \epsilon_L\}$ completes the proof. □

Remark 2.6. *This bound is tight. Consider $p = q = 1/2$, so $\pi_H = \pi_L = 1/2$. Any static curve is suboptimal exactly half the time, losing $\epsilon T/2$ in expectation.*

2.4 Interpretation: The Adaptivity Gap

Theorem 2.5 deserves some reflection. It says something quite strong: *no matter which static curve you pick*, you will accumulate welfare loss proportional to time. The bound isn't about a bad choice of curve—it's about the impossibility of any static choice being optimal.

The proof reveals why: the optimal curve differs between regimes, and a static curve can only match one regime at a time. During the other regime, it's suboptimal. Since both regimes occur with positive probability (by assumption), suboptimality is unavoidable.

We call this accumulated loss the *adaptivity gap*—the gap between what we achieve and what we could achieve if we adapted perfectly.

Definition 2.7 (Adaptivity Gap)**.** *The adaptivity gap of curve c over horizon T is:*

$$\mathcal{G}(c, T) = \int_0^T W(c^*(\sigma_t), \sigma_t) - W(c, \sigma_t)\, dt \tag{7}$$

The theorem shows $\mathbb{E}[\mathcal{G}(c, T)] = \Omega(T)$ for any static c. This motivates the search for adaptive mechanisms with $o(T)$ gap—or ideally $O(\sqrt{T})$ as achieved by optimal learning algorithms in stationary settings.

2.5 Practical Implications

What does linear welfare loss mean concretely? Consider a \$10M pool over one year:

- Suppose the optimal fee is 0.30% in calm markets and 0.50% in volatile markets

- A static 0.40% fee loses 0.10% of volume in both regimes (vs. optimal)
- With $100M annual volume, this is $100K in foregone fees
- Plus impermanent loss differences, slippage costs, etc.

These numbers are illustrative, not empirical. The point is that the adaptivity gap translates to real economic costs. Whether these costs justify the complexity of adaptation is an empirical question we cannot yet answer.

3 Generalized Bonding Curves

Having established that adaptation is valuable, we now construct a family of curves suitable for adaptive selection. The key requirement is a *continuous parameterization*: we need curves indexed by a parameter κ such that small changes in κ produce small changes in curve behavior.

Why is continuity important? Because adaptation involves *gradual* adjustment. We don't want to jump discontinuously from one curve type to another—that would cause unpredictable price changes and create arbitrage opportunities. We want to smoothly interpolate, perhaps starting with a robust constant-product curve and gradually flattening it as we observe stable conditions, ready to unflatten if volatility returns.

Existing AMM designs don't offer this. Uniswap's constant product and Curve's StableSwap are separate mathematical objects with no natural interpolation between them. Moving from one to the other requires migrating liquidity to a different pool. We need a unified family that contains both as special cases.

3.1 Why Naive Blending Fails

Before presenting our solution, let's understand why the obvious approach doesn't work. The naive idea is to "blend" existing invariants by taking a weighted average:

$$\mathcal{I}_{\text{naive}}(x, y) = (1 - \kappa) \cdot xy + \kappa \cdot f_{\text{stable}}(x, y) \tag{8}$$

This looks natural: when $\kappa = 0$, we get constant product; when $\kappa = 1$, we get StableSwap. What could go wrong?

This approach is mathematically problematic:

1. **Scale mismatch**: The constant product invariant $xy = k$ and StableSwap invariant $f(x, y) = D$ produce values of completely different magnitudes. Adding them is meaningless.
2. **Level set distortion**: Even if we normalize, the level sets of a sum of functions bear no simple relationship to the level sets of the components. The "blended" curve may have pathological properties.
3. **Non-monotonicity**: There's no guarantee that increasing κ smoothly transitions behavior from constant product to StableSwap.

We need a different approach—one that's mathematically coherent from the start.

3.2 The Generalized Bonding Curve Family

Our solution is conceptually elegant: instead of blending invariants (which doesn't make mathematical sense), we blend the *exponents* that control curvature.

Think of it geometrically. The constant product curve $xy = k$ is a hyperbola—highly curved, approaching but never touching the axes. The constant sum curve $x + y = k$ is a straight line—zero curvature, touching both axes. Between these extremes lies a continuum of curves with varying curvature.

We can navigate this continuum by taking products with varying exponents. The key insight is that $(xy)^{1-\kappa} \cdot (x+y)^{2\kappa}$ smoothly interpolates between xy (when $\kappa = 0$) and $(x+y)^2$ (when $\kappa = 1$). The exponents always sum to 2, maintaining consistent scale.

Instead of blending invariants, we construct a family where the *curvature* of the level set varies continuously with a parameter.

Definition 3.1 (Generalized Bonding Curve)**.** *For reserves (x, y) with $x, y > 0$, curvature parameter $\kappa \in [0, 1]$, and scale parameter $D > 0$, define:*

$$\mathcal{I}_\kappa(x, y) = \left(\frac{xy}{D^2}\right)^{1-\kappa} \cdot \left(\frac{x+y}{2D}\right)^{2\kappa} = 1 \tag{9}$$

This definition has the correct limiting behavior:

Proposition 3.2 (Boundary Cases)**.**

1. *When $\kappa = 0$: $\mathcal{I}_0(x, y) = xy/D^2 = 1$, which is constant product with $k = D^2$.*
2. *When $\kappa = 1$: $\mathcal{I}_1(x, y) = (x+y)/(2D) = 1$, which is constant sum with $x + y = 2D$.*
3. *For $\kappa \in (0, 1)$: The curve interpolates, with higher κ producing flatter curves near $x = y$.*

Proof. Direct substitution. For $\kappa = 0$, the second factor is $(x+y)/(2D))^0 = 1$. For $\kappa = 1$, the first factor is $(xy/D^2)^0 = 1$. □

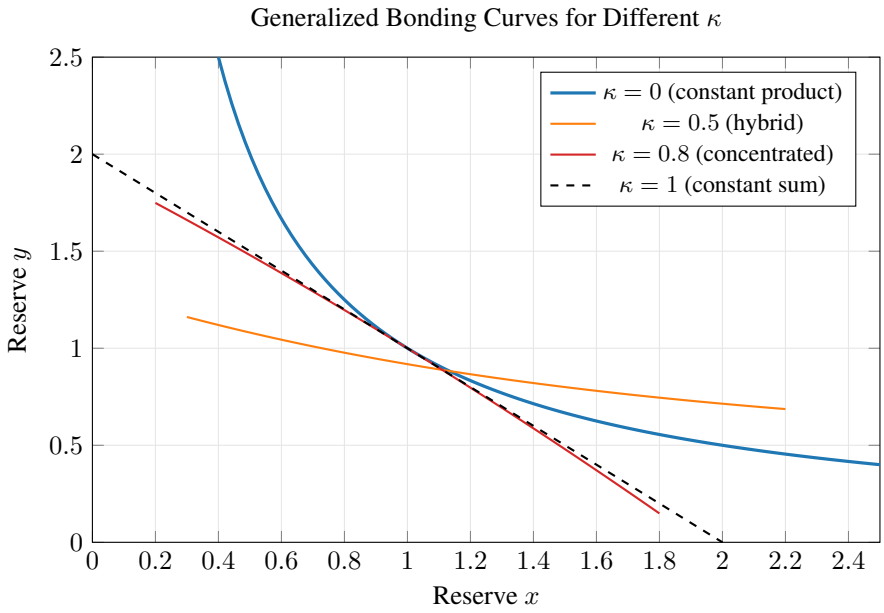

Figure 2: Level curves of the Generalized Bonding Curve for $D = 1$ and various κ. As κ increases, the curve flattens near the diagonal (providing concentrated liquidity) but reaches axis intercepts (allowing reserve depletion).

3.3 Geometric Interpretation

Mathematics becomes clearer when we can visualize it. The GBC family has an elegant geometric interpretation that illuminates why κ behaves the way it does.

Consider the curve as a level set in (x, y) space—the set of all (x, y) pairs satisfying $\mathcal{I}_\kappa(x, y) = 1$. For an AMM, this curve represents all possible reserve states. When a trader swaps, they move the pool along this curve.

Proposition 3.3 (Curvature at Equilibrium)**.** *At the balanced point $x = y = D$, the Gaussian curvature of the level set is:*

$$K(\kappa) = \frac{1-\kappa}{D^2} \tag{10}$$

Proof. At $x = y = D$, compute the Hessian of $\log \mathcal{I}_\kappa$ and apply the formula for curvature of an implicit curve. The $(1-\kappa)$ factor comes from the exponent on the xy term. □

This shows that κ directly controls curvature: $\kappa = 0$ gives maximum curvature (hyperbolic), $\kappa = 1$ gives zero curvature (linear). The parameter κ is literally the "flatness" of the curve near equilibrium.

What does flatness mean economically? A flatter curve provides better prices near the current exchange rate—traders get less slippage for small trades. But the same flatness that helps near the center hurts at the extremes: as reserves become imbalanced, a flat curve continues offering (nearly) the same exchange rate, even as the market price moves away. This is the capital efficiency vs. robustness tradeoff, encoded geometrically.

3.4 Swap Computation

Theory is nice, but we need to actually compute swaps. Given an input amount Δx, what output Δy does the pool provide?

For constant product ($\kappa = 0$), there's a beautiful closed-form answer: $\Delta y = y \cdot \Delta x/(x + \Delta x)$. This is why Uniswap is so efficient: one multiplication, one division, done.

For general κ, life is harder. The GBC invariant is transcendental—it involves fractional powers—and there's no closed-form solution. But there's an algorithm: Newton-Raphson iteration.

For practical use, we need to compute swap outputs efficiently.

Proposition 3.4 (Swap Equation)**.** *Given input $\Delta x > 0$ and current reserves (x, y) on the curve $\mathcal{I}_\kappa(x, y) = 1$, the output Δy satisfies:*

$$\left(\frac{(x+\Delta x)(y-\Delta y)}{D^2}\right)^{1-\kappa} \cdot \left(\frac{(x+\Delta x)+(y-\Delta y)}{2D}\right)^{2\kappa} = 1 \tag{11}$$

This equation is transcendental for $\kappa \in (0, 1)$ and has no closed-form solution. We solve it numerically.

Theorem 3.5 (Newton-Raphson Convergence)**.** *For any valid swap (where $0 < \Delta y < y$), Newton-Raphson iteration on the swap equation converges to the unique solution. Starting from the constant-product estimate $\Delta y_0 = y\Delta x/(x+\Delta x)$, the iteration achieves ϵ relative error in $O(\log\log(1/\epsilon))$ iterations.*

Proof. Define $f(\Delta y) = \mathcal{I}_\kappa(x + \Delta x, y - \Delta y) - 1$. We verify:

Uniqueness: f is strictly monotonic in Δy on $(0, y)$ because increasing Δy decreases both factors in $\mathcal{I}_\kappa$ (since both xy and $x + y$ decrease when y decreases).

Existence: $f(0) > 0$ (adding x without removing y increases the invariant) and $f(y) = -1 < 0$ (removing all y makes $\mathcal{I}_\kappa = 0$). By continuity, a root exists.

Convergence: f is C^2 with bounded second derivative on compact subsets. The initial guess from constant product is within a constant factor of the true solution. By standard Newton-Raphson theory, quadratic convergence gives the stated bound. □

Remark 3.6. *In practice, 5-10 iterations suffice for 64-bit precision. This is efficient for on-chain computation, requiring approximately 2,000-5,000 compute units on Solana depending on κ.*

3.5 Capital Efficiency Analysis

The key tradeoff in choosing κ is between capital efficiency and robustness.

Definition 3.7 (Capital Efficiency)**.** *The capital efficiency at price $p = y/x$ with tolerance ϵ is:*

$$\eta(\kappa, p, \epsilon) = \frac{\textit{max trade with slippage} \leq \epsilon}{\textit{total value locked}} \tag{12}$$

Proposition 3.8 (Efficiency at Equilibrium)**.** *At equilibrium $(x = y = D)$ with price $p = 1$:*

$$\eta(\kappa, 1, \epsilon) = \Theta\left(\epsilon^{1/(1-\kappa)}\right) \tag{13}$$

for small ϵ.

Proof Sketch. Near equilibrium, the curve behaves like $(xy)^{1-\kappa} \approx \text{const}$, which gives slippage scaling as $(\Delta x/x)^{1-\kappa}$. Inverting for Δx at fixed slippage ϵ gives the stated scaling. □

This quantifies the efficiency-robustness tradeoff:

- $\kappa = 0$: Efficiency $\Theta(\epsilon)$, moderate but consistent across all prices
- $\kappa \to 1$: Efficiency $\Theta(\epsilon^{1/(1-\kappa)}) \to \infty$ at equilibrium, but drops sharply as price moves

Example 3.9 (Concrete Efficiency Comparison)**.** *Consider a $10M pool (TVL = $10M) with equal reserves. A trader wants to swap with at most 0.1% slippage ($\epsilon = 0.001$):*

κ	*Max trade size*	*Efficiency*
0.0 (constant product)	*$10,000*	*0.10%*
0.5 (hybrid)	*$31,623*	*0.32%*
0.8 (concentrated)	*$100,000*	*1.00%*
0.95 (highly concentrated)	*$398,107*	*3.98%*

At $\kappa = 0.8$, the pool can support 10x larger trades at the same slippage. But if the external price moves 5% away from 1:1, the $\kappa = 0.8$ pool offers worse prices than the $\kappa = 0$ pool—the concentrated liquidity is now in the "wrong" place.

This is why adaptation matters: a pool that can shift from $\kappa = 0.8$ to $\kappa = 0.3$ when volatility spikes captures the best of both worlds.

3.6 Limitations of the GBC Family

We acknowledge several limitations:

1. **Extreme κ**: As $\kappa \to 1$, the curve approaches constant sum, which allows complete reserve depletion. Practical implementations should bound $\kappa \leq 0.95$ or similar.
2. **Multi-asset generalization**: Definition 3.1 is for two assets. Extension to n assets is possible but the geometry becomes more complex.
3. **Comparison to StableSwap**: The GBC family does not exactly reproduce StableSwap, which has its own amplification parameter A. Whether GBC or StableSwap is "better" for stablecoin pairs is an empirical question.
4. **Dynamic D**: We've treated D as fixed. In practice, D changes with liquidity additions/removals. The interaction between adapting κ and changing D needs analysis.

4 Reinforcement Learning for Adaptation

We now address the core question: how should a pool adapt its curve parameter κ over time? We propose reinforcement learning and analyze the challenges.

The high-level idea is simple: the pool observes market conditions, chooses curve parameters, receives rewards (fees minus losses), and learns which choices work best. Over time, it should converge on policies that outperform static alternatives.

But the execution is fraught with difficulty. Reinforcement learning was developed for stationary environments—games with fixed rules, robots in consistent physics, recommendation systems with stable user preferences. Markets are none of these. They're adversarial, non-stationary, and subject to regime changes that can invalidate everything learned so far.

This section lays out the formalism, identifies the central challenge (what we call the *tracking-vs-learning tradeoff*), proposes a practical architecture, and is honest about what remains unsolved.

4.1 The Adaptation Problem as MDP

Definition 4.1 (Curve Adaptation MDP)**.** *The adaptation problem is a Markov Decision Process:*

- ***State*** s_t*: Observable market features (price, volume, volatility estimates)*
- ***Action*** a_t*: Change to curve parameter,* $a_t \in \{-\delta, 0, +\delta\}$ *for step size* δ
- ***Reward*** r_t*: Social welfare realized since last action*
- ***Transition****: Determined by market dynamics and trader responses*

The goal is to learn a policy $\pi : S \to A$ maximizing cumulative discounted reward $\sum_t \gamma^t r_t$.

This formalism captures the essence of the problem: the pool repeatedly observes the market, makes a decision, and experiences consequences. The MDP structure enables us to apply decades of reinforcement learning theory—in principle. In practice, the theory assumes conditions our setting violates.

4.2 The Non-Stationarity Challenge

Here's where things get hard. Standard RL assumes a stationary MDP: the transition kernel $P(s'|s, a)$ and reward function $R(s, a)$ are fixed. Play the same game a million times, and you'll learn how to play it well.

Our setting violates this assumption fundamentally: market conditions change, altering both transitions and rewards. The reward for choosing $\kappa = 0.5$ today might be positive (calm market, capital efficiency matters). Tomorrow, with a volatility spike, the same choice yields negative rewards (impermanent loss exceeds fees).

Definition 4.2 (Non-Stationary MDP)**.** *A non-stationary MDP is a sequence* $\{\mathcal{M}_t\}$ *where* $\mathcal{M}_t = (S, A, P_t, R_t)$ *varies with* t*.*

Open Problem 4.3 (Non-Stationary Convergence)**.** *Under what conditions does Q-learning converge to a meaningful solution in a non-stationary MDP? What is the appropriate notion of "solution"?*

This is not merely a technical obstacle—it's a fundamental conceptual challenge. In a non-stationary environment, there may be no fixed optimal policy to converge to. What does it even mean to "solve" such a problem?

The literature offers some answers: minimize regret relative to the best static policy, track a slowly-changing optimal policy, or achieve competitive ratio bounds. But these feel like compromises, not solutions. We're honest that a fully satisfying theory remains elusive.

4.3 The Tracking-Learning Tradeoff

If we can't solve the non-stationarity problem completely, can we at least understand it? We believe the key insight is recognizing a fundamental tension between two desirable properties.

We identify the central challenge as a tradeoff between two objectives:

Learning requires stability: to estimate Q-values accurately, we need many samples from a consistent environment. High learning rates cause oscillation; low learning rates give stable estimates.

Tracking requires responsiveness: to follow a changing optimum, we must update quickly. Low learning rates cause lag; high learning rates enable fast adaptation.

These objectives conflict. We cannot simultaneously have stable Q-estimates (for good decisions) and fast adaptation (for tracking regime changes).

An analogy might help. Imagine trying to follow a moving target while wearing glasses. If the glasses update instantly (high learning rate), you track the target well but the image is jittery—noise gets amplified. If the glasses update slowly (low learning rate), the image is smooth but you're always looking where the target *was*, not where it is.

There's an optimal middle ground, but finding it requires knowing how fast the target moves—which we don't know in advance. This is the essence of the tradeoff.

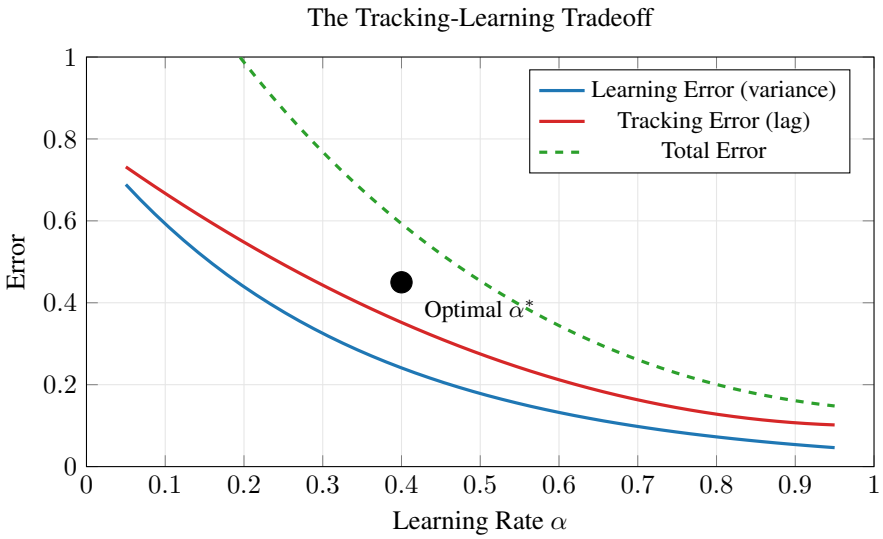

Figure 3: The fundamental tradeoff in adaptive learning. Low learning rates give stable but lagging estimates. High learning rates track quickly but with high variance. The optimal rate balances these forces.

Definition 4.4 (Tracking Error)**.** *The tracking error at time T is:*

$$\epsilon_T = \|Q_T - Q_T^*\|_\infty \tag{14}$$

where Q_T is the learned Q-function and Q_T^* is the optimal Q-function for current conditions.

Conjecture 4.5 (Tracking Bound)**.** *Suppose market conditions σ_t have bounded variation: $\sum_t \|\sigma_{t+1} - \sigma_t\| \leq V_T$. Then with learning rate $\alpha_t = t^{-\omega}$ for $\omega \in (0.5, 1)$:*

$$\mathbb{E}[\epsilon_T] \leq C_1 T^{-\omega/2} + C_2 V_T T^{\omega - 1} \tag{15}$$

where C_1, C_2 depend on MDP parameters.

Discussion: The first term is learning error (decreasing). The second term is tracking error (increasing if V_T grows). The optimal ω balances these terms. If $V_T = O(T^\beta)$ for $\beta < 1$, minimizing the bound suggests $\omega^* = (1 + \beta)/2$.

What's needed for a proof: This conjecture requires (1) a precise model of how σ_t affects P_t and R_t, (2) Lipschitz bounds on Q-functions with respect to σ, and (3) careful analysis of error accumulation. We have not completed this analysis.

4.4 Proposed Q-Learning Architecture

Despite theoretical gaps, we propose a practical architecture:

Definition 4.6 (State Encoding)**.** *Discretize market state into bins:*

$$s = (\textit{vol\_bin}, \textit{trend\_bin}, \textit{spread\_bin}) \in \{0, 1, 2\}^3 \tag{16}$$

giving $|S| = 27$ *states.*

Definition 4.7 (Action Space)**.**

$$A = \{\kappa_{\textit{down}}, \kappa_{\textit{hold}}, \kappa_{\textit{up}}\} \tag{17}$$

with step size $\delta_\kappa = 0.05$.

Definition 4.8 (Reward Function)**.**

$$r_t = w_{\textit{fee}} \cdot \textit{fees}_t - w_{\textit{IL}} \cdot \textit{IL}_t + w_{\textit{vol}} \cdot \log(\textit{volume}_t) \tag{18}$$

with tunable weights.

Algorithm 1 On-Chain Q-Learning (Proposed)

1: **On each swap:** Record observation (s, a, r, s') in circular buffer
2: **Periodically:** Batch update Q-table from buffer:
3: **for** (s, a, r, s') in buffer **do**
4: $\quad Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s,a)]$
5: **Select action:** $a = \begin{cases} \arg\max_a Q(s,a) & \text{w.p. } 1-\epsilon \\ \text{random} & \text{w.p. } \epsilon \end{cases}$

4.5 Bounded Actions for Safety

Whatever the learning algorithm decides, we enforce hard safety bounds:

$$\kappa_t \in [\kappa_{\min}, \kappa_{\max}] = [0.05, 0.95] \tag{19}$$

This prevents degenerate configurations regardless of learning errors.

Proposition 4.9 (Bounded Welfare Loss)**.** *With bounded* κ*, the per-period welfare loss from any action is bounded:*

$$W(c^*(\sigma_t), \sigma_t) - W(c_{\kappa_t}, \sigma_t) \leq W_{\max} \tag{20}$$

for some finite $W_{\max}$ *depending on the bounds.*

This provides a safety net: even if learning fails completely, the protocol cannot lose more than $W_{\max}$ per period.

4.6 Open Problems in Learning

We identify key open problems:

Open Problem 4.10 (Optimal State Representation)**.** *What market features should be included in the state? How fine should discretization be? Finer states give more expressiveness but require more data to learn.*

Open Problem 4.11 (Reward Design)**.** *How should we weight different objectives (fees, IL, volume, LP retention)? Is there a principled way to set weights, or must they be tuned empirically?*

Open Problem 4.12 (Exploration in Production)**.** *Exploration (trying suboptimal actions to learn) costs real money for real users. How do we balance learning with exploitation in a live system?*

Open Problem 4.13 (Adversarial Robustness)**.** *Can traders manipulate observations to bias learning? What is the cost of such manipulation, and can we design mechanisms that are robust?*

5 Mechanism Design for Tiered Access

A distinctive feature of AEX-402 is tiered access: traders can pay for enhanced execution quality. We analyze the mechanism design.

Why tiers? Because agents differ. Some are price-insensitive arbitrageurs racing to capture a $100,000 profit; they'll pay premium for guaranteed fast execution. Others are retail-equivalent bots making small trades; they're happy to wait and save costs. A one-size-fits-all approach either overcharges the latter or underserves the former.

Airlines understood this decades ago. Business class and economy serve the same route but at different prices with different amenities. The key to making this work is *self-selection*: travelers reveal their preferences through their choices. We design AMM tiers similarly.

5.1 Tier Structure

We propose three tiers, distinguished by execution quality and price:

- **Standard** (free): Best-effort execution via public mempool
- **Priority** ($0.01): Faster inclusion via priority fee
- **Guaranteed** ($0.05): MEV protection via private submission, execution guarantee with backstop

5.2 Trader Types and Valuations

The fundamental question in mechanism design is: how do we set prices to maximize some objective (revenue, welfare, fairness) when we don't know each participant's private information?

In our setting, the private information is how much a trader values execution quality. An arbitrageur racing against competitors might value guaranteed execution at $1.00 per trade—it's cheap insurance on a high-stakes opportunity. A casual agent rebalancing a small portfolio might value it at $0.001—the stakes don't justify the cost.

We model this heterogeneity with a single-dimensional "type" that captures value for quality. Traders differ in their value for execution quality. Let $\theta \in [0, 1]$ denote a trader's type, where higher θ means higher value for speed and MEV protection.

Definition 5.1 (Trader Utility)**.** *A type-θ trader using tier τ with price p_τ and quality q_τ receives utility:*

$$U(\theta, \tau) = \theta \cdot q_\tau - p_\tau \tag{21}$$

Definition 5.2 (Incentive Compatibility)**.** *A tier mechanism is* incentive compatible *if each type θ maximizes utility by choosing the tier designed for their type.*

5.3 Incentive Compatibility Analysis

For self-selection to work, we need *incentive compatibility*: each type should prefer the tier designed for them. If high-value traders can get good quality at low prices by pretending to be low-value, the system breaks down.

The mathematics here is classical—this is the textbook screening problem from Mussa and Rosen (1978) [Mussa and Rosen, 1978]. Our contribution is applying it to the AMM context and identifying the specific quality and price parameters that work.

Theorem 5.3 (IC Conditions)**.** *Under the utility model above, the tier mechanism is incentive compatible if:*

$$\theta_1 = \frac{p_{priority} - p_{standard}}{q_{priority} - q_{standard}} \tag{22}$$

$$\theta_2 = \frac{p_{guaranteed} - p_{priority}}{q_{guaranteed} - q_{priority}} \tag{23}$$

where types $\theta < \theta_1$ choose standard, $\theta \in [\theta_1, \theta_2]$ choose priority, and $\theta > \theta_2$ choose guaranteed.

Proof. A type-θ trader chooses standard over priority iff:

$$\theta \cdot q_{\text{standard}} - p_{\text{standard}} \geq \theta \cdot q_{\text{priority}} - p_{\text{priority}} \tag{24}$$

Rearranging: $\theta(q_{\text{priority}} - q_{\text{standard}}) \leq p_{\text{priority}} - p_{\text{standard}}$, giving the threshold θ_1. The threshold θ_2 follows analogously. □

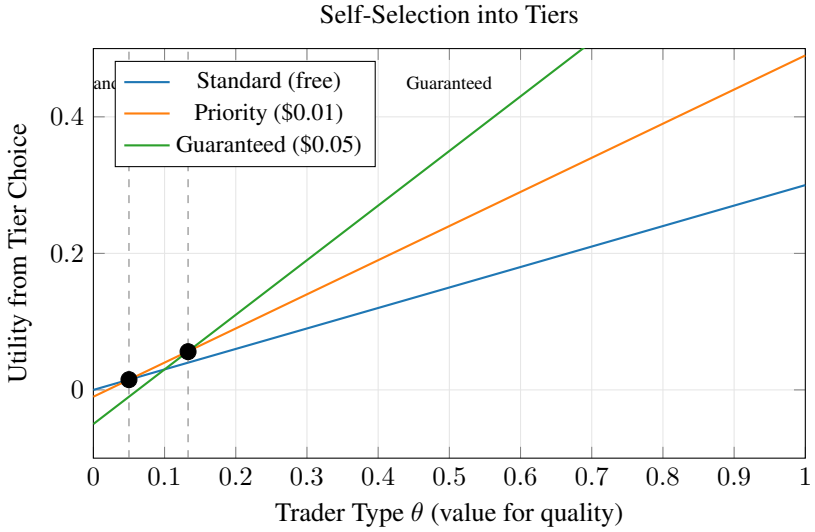

Figure 4: Utility by tier as a function of trader type. Vertical dashed lines show self-selection thresholds. Each type optimally chooses the tier where their utility is highest.

Corollary 5.4 (Revenue Expression)**.** *If types are distributed with CDF F, expected revenue per trader is:*

$$R = F(\theta_1) \cdot p_{standard} + [F(\theta_2) - F(\theta_1)] \cdot p_{priority} + [1 - F(\theta_2)] \cdot p_{guaranteed} \tag{25}$$

5.4 Limitations of the Analysis

This analysis makes strong assumptions:

1. **Price-taking traders**: We assume traders take tier prices as given. In reality, large traders might negotiate or strategize.

2. **Known type distribution**: Revenue optimization requires knowing F. In practice, this must be estimated from data.

3. **Single-dimensional types**: Real traders may value speed and MEV protection differently. A multi-dimensional type space is more realistic but harder to analyze.

4. **No strategic interaction**: We ignore that one trader's choice affects others (e.g., MEV is partly a zero-sum game).

Open Problem 5.5 (Strategic Mechanism Design)**.** *How should tiers be designed when traders are strategic? Is truthful revelation of type achievable, and at what cost to revenue?*

Open Problem 5.6 (Dynamic Pricing)**.** *Should tier prices adapt to demand? This creates an additional learning problem interacting with the curve adaptation.*

6 Virtual Pool Graduation System

A critical barrier to new token launches is the "cold start" problem: creating an AMM pool requires significant upfront capital for liquidity, account rent, and market making. This favors well-funded projects and creates friction for organic community tokens. We introduce *Virtual Pool Graduation*—a mechanism that enables zero-rent token launches with bonding curves, anti-manipulation protections, and seamless transition to full AMM pools.

6.1 The Cold Start Problem

Traditional token launches face a chicken-and-egg dilemma:

- **Liquidity requirement**: AMM pools need initial liquidity to function
- **Rent costs**: On-chain accounts (mint, vaults, LP mint) require rent deposits
- **Market making**: Someone must seed initial reserves, taking on inventory risk
- **Price discovery**: Without trading history, fair price is unknown

The result: launching a token on Solana costs approximately 0.05 SOL in rent alone, plus significant capital for initial liquidity. This creates barriers and, ironically, does nothing to prevent scams—well-funded bad actors can still rug-pull.

6.2 Virtual Pool Architecture

Our solution: tokens exist *virtually* until they prove market demand. A Virtual Pool contains no real token mint, no vaults, no LP tokens—just accounting entries in a single Program Derived Address (PDA).

Definition 6.1 (Virtual Pool)**.** *A Virtual Pool $\mathcal{V}$ is a tuple $(M, \beta, \mathbf{h}, s)$ where:*

- M *= metadata (name, symbol, URI) for eventual token creation*
- $\beta = (b_0, \gamma)$ *= bonding curve parameters (base price, slope)*
- $\mathbf{h} = \{(w_i, q_i)\}_{i=1}^{n}$ *= holder set (wallet hash, quantity)*
- s *= SOL raised (stored as lamports in the PDA)*

The virtual pool uses a *linear bonding curve* for price discovery:

$$p(t) = b_0 + \gamma \cdot t \tag{26}$$

where t is the cumulative tokens sold. This creates automatic price appreciation as demand increases, rewarding early participants while ensuring later buyers pay fair market rates.

Theorem 6.2 (Bonding Curve Integral)**.** *For a buy of Δt tokens starting from cumulative sales t_0, the SOL cost is:*

$$C(\Delta t) = \int_{t_0}^{t_0+\Delta t} p(t)\, dt = b_0 \cdot \Delta t + \frac{\gamma}{2}\left[(t_0 + \Delta t)^2 - t_0^2\right] \tag{27}$$

Solving for Δt given SOL input S yields the quadratic:

$$\Delta t = \frac{-b_0 - \gamma t_0 + \sqrt{(b_0 + \gamma t_0)^2 + 2\gamma S}}{\gamma} \tag{28}$$

6.3 Anti-Manipulation: Dynamic Graduation Target

A naive graduation mechanism ("graduate when $s \geq$ target") is vulnerable to manipulation:

1. Scammer buys to push pool near graduation threshold
2. Legitimate buyers FOMO in, pushing toward graduation
3. Scammer sells, dropping back below threshold
4. Repeat, extracting value from each cycle

We introduce a *dynamic graduation target* that penalizes this "churn" behavior:

Definition 6.3 (Churn Ratio)**.** *For a pool with gross buy volume V_B, gross sell volume V_S, and net inflow $N = V_B - V_S$:*

$$\chi = \frac{V_B + V_S}{N} = \frac{\textit{gross volume}}{\textit{net inflow}} \tag{29}$$

Healthy organic growth has $\chi \approx 1$–2. Manipulation cycles have $\chi \gg 3$.

Definition 6.4 (Dynamic Graduation Target)**.** *The graduation target T^* adjusts based on trading patterns:*

$$T^*(N, \chi) = \textit{clip}\left(T_0 - \alpha N + \beta \cdot \max(0, \chi - 3),\, T_{\min},\, T_{\max}\right) \tag{30}$$

where $T_0 = 100$ SOL (base), $\alpha = 0.5$ (progress bonus), $\beta = 5$ SOL (churn penalty), $T_{\min} = 10$ SOL, $T_{\max} = 200$ SOL.

Theorem 6.5 (Manipulation Unprofitability)**.** *Under the dynamic graduation mechanism with fee rate f, a manipulator executing k buy-sell cycles with gross volume V per cycle has expected profit:*

$$\Pi_k = -k \cdot f \cdot V + \underbrace{\textit{(extraction from FOMO buyers)}}_{\to 0 \textit{ as target rises}} \tag{31}$$

As k increases, χ increases, T^* increases, and the pool becomes harder *to graduate. The manipulator accumulates fees while the extraction opportunity shrinks.*

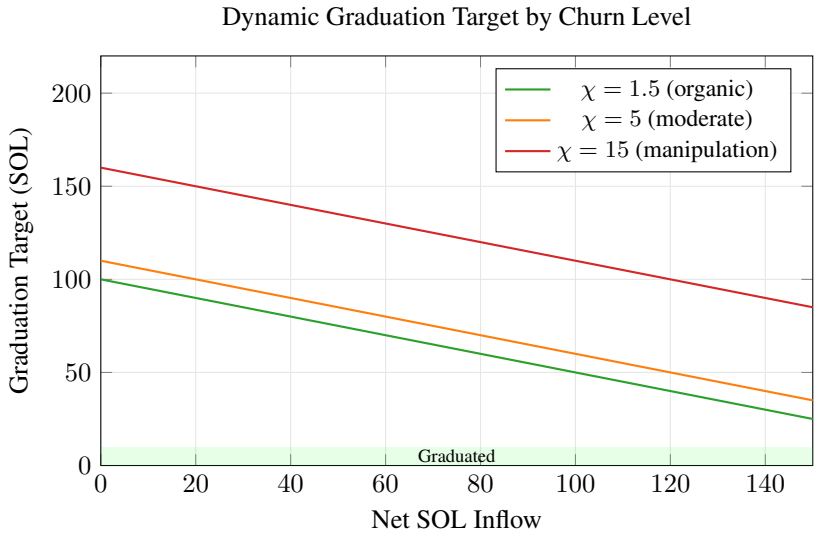

Figure 5: Graduation target as function of net inflow for different churn levels. Manipulation ($\chi = 15$) raises the target significantly, making graduation harder. Organic growth ($\chi = 1.5$) benefits from lowered targets.

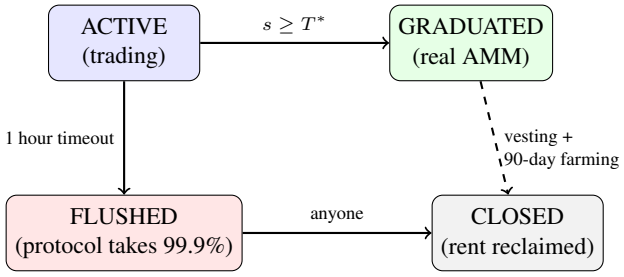

Figure 6: Virtual pool state machine. Pools must graduate within 1 hour or face flush where protocol captures 99.9% of raised SOL.

6.4 Graduation Lifecycle

Virtual pools progress through defined states with a strict 1-hour deadline:

The 1-hour deadline creates urgency and prevents indefinite manipulation. If a pool cannot achieve genuine demand within one hour, it is flushed—the protocol takes 99.9% of raised SOL, with 0.1% going to the caller as incentive for permissionless cleanup.

6.5 Token Distribution and Vesting

Upon graduation, tokens are allocated from two pools:

Table 2: Token Allocation at Graduation

Source	Recipient	Vesting
Sold tokens (100%)	Holders	5% per hour
Unsold (5%)	Creator	2% per hour
Unsold (20%)	Farming rewards	90-day distribution
Unsold (remainder)	AMM liquidity	Immediate

Definition 6.6 (Exponential Vesting)**.** *A holder with initial allocation A_0 can claim at time t (in hours):*

$$\textit{claimable}(t) = A_0 \cdot (1 - 0.95^t) \tag{32}$$

This asymptotically approaches A_0, with 50% claimable by hour 13 and 95% by hour 59.

The slower creator vesting (2%/hour vs 5%/hour for holders) builds trust—creators cannot dump faster than their community.

6.6 Ultra-Compact Holder Representation

A key technical innovation is the *7-byte holder format* that enables 1.49 million holders per pool:

- **Wallet hash**: 6 bytes (Caesar-rotated positions from pubkey)
- **Balance**: 1 byte (units of 100,000 tokens, max 2.5% supply)

Traditional token accounts require $\sim$165 bytes each plus rent. Our compact format provides a $23\times$ space improvement.

Theorem 6.7 (Hash Collision Security)**.** *With 6-byte hashes and Caesar rotation (different byte positions per holder index), grinding a collision for holder i requires expected 2^{47} operations. At 1 billion hashes/second (high-end GPU cluster), this takes $\sim$39 hours—but pools flush in 1 hour.*

The 1-hour lifetime creates a race condition where attackers cannot complete grinding before the pool graduates or flushes, rendering collision attacks economically unviable.

6.7 Post-Graduation Farming: The Volume Flywheel

A critical weakness of existing token launch platforms like Pump.fun is the *post-graduation death spiral*: once a token graduates to a DEX, trading activity collapses. The bonding curve created artificial scarcity and momentum; the DEX offers neither. We solve this with a novel *competitive farming* mechanism that creates sustained volume through game-theoretic incentives.

6.7.1 The Pump.fun Problem

Pump.fun's bonding curve creates urgency through rising prices: buy now or pay more later. This drives volume during the pre-graduation phase. But once the token graduates to Raydium:

- Price discovery is "complete"—the urgency disappears
- No mechanism rewards continued trading
- Early holders dump on the AMM, cratering price
- Volume collapses within 24–48 hours
- Token becomes illiquid and effectively dead

Remark 6.8 (Pump.fun's Structural Flaw)**.** *Pump.fun optimizes for* graduation velocity*, not* post-graduation sustainability*. The platform captures fees during the bonding curve phase but has no stake in the token's long-term success. This misalignment produces short-lived "pump and dump" dynamics even for legitimate projects.*

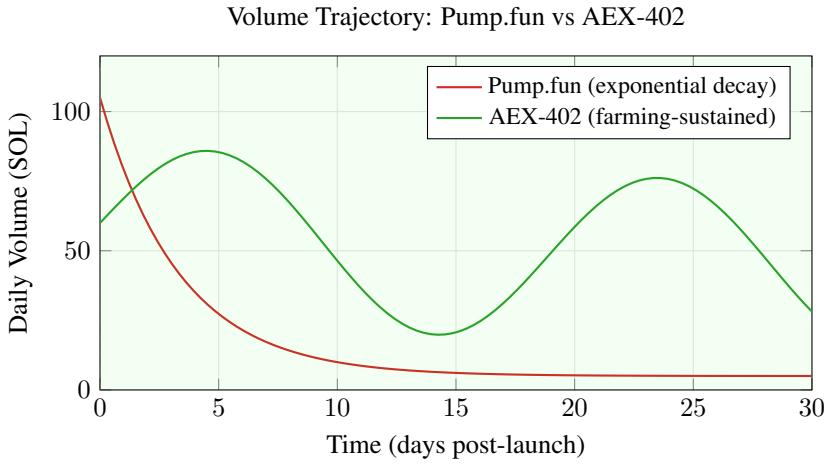

Figure 7: Stylized volume trajectories. Pump.fun tokens experience rapid volume decay post-graduation as the bonding curve momentum disappears. AEX-402's competitive farming maintains elevated, oscillating volume for 90 days.

6.7.2 The Competitive Farming Innovation

We allocate 20% of unsold tokens to a 90-day farming program that creates *artificial competition for volume*. The mechanism:

1. Divide 90 days into 25,920 windows (5 minutes each)
2. Each window, reward pool = $\frac{\text{remaining farming tokens}}{\text{remaining windows}}$
3. Track top 10 buyers by SOL spent in each window
4. Randomly select 3 winners from top 10; each receives 30% of window reward
5. Committer receives 10% for providing randomness

Definition 6.9 (Window Expected Value)**.** *For a trader spending s SOL in a window with total top-10 spend S, the expected farming reward is:*

$$\mathbb{E}[\textit{reward}] = \mathbb{P}(\textit{in top 10}) \cdot \frac{3}{10} \cdot 0.9 \cdot R_w \tag{33}$$

where R_w is the window reward and $\frac{3}{10}$ reflects the probability of being among the 3 winners given top-10 status.

6.7.3 The Positive Feedback Loop

The farming mechanism creates a self-reinforcing cycle:

Theorem 6.10 (Volume Amplification)**.** *Let V_0 be the "natural" volume (trading that would occur without farming incentives) and let R be the farming reward rate per window. Under rational profit-maximizing traders, the equilibrium volume satisfies:*

$$V^* = V_0 + \alpha \cdot R \tag{34}$$

where $\alpha > 0$ is the competition multiplier *depending on the number of active traders and their risk preferences.*

Proof Sketch. Traders compete for top-10 positions. A trader i chooses volume v_i to maximize:

$$U_i(v_i) = \underbrace{u_i(v_i)}_{\text{intrinsic utility}} + \underbrace{\mathbb{P}(\text{top 10}|v_i) \cdot 0.27R}_{\text{expected farming reward}} - \underbrace{c(v_i)}_{\text{trading cost}} \tag{35}$$

In Nash equilibrium, traders increase v_i until marginal cost equals marginal reward probability. With n competing traders, equilibrium volume scales with R. □ □

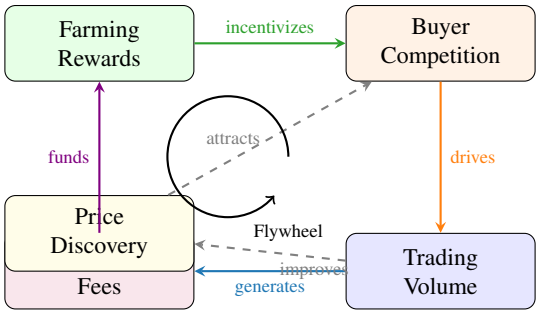

Figure 8: The volume flywheel. Farming rewards create competition; competition drives volume; volume generates fees; fees can fund additional rewards. The cycle is self-reinforcing for 90 days.

6.7.4 Competition Dynamics

The "top 10 from random 3" design creates specific game-theoretic properties:

Table 3: Farming Mechanism Design Choices

Design Choice	Alternative	Rationale
Top 10 tracked	Top 3	Reduces "winner take all" pressure
Random 3 win	Top 3 win	Prevents last-second sniping
5-min windows	1-hour windows	More frequent engagement
30% each winner	Split by volume	Rewards participation over size
90-day duration	30-day	Sustained community building

Proposition 6.11 (Anti-Sniping Property)**.** *The random winner selection prevents "last-second sniping" strategies where traders wait until $t = 4:59$ to place a large buy that displaces others from top 10. Since being in top 10 only gives $\frac{3}{10} = 30\%$ win probability, sniping is less profitable than consistent participation.*

6.7.5 Comparison to Pump.fun Tokenomics

Table 4: Tokenomics Comparison: Pump.fun vs AEX-402

Dimension	Pump.fun	AEX-402
Pre-graduation incentive	Rising bonding curve	Rising bonding curve
Post-graduation incentive	None	90-day competitive farming
Volume sustainability	Exponential decay	Sustained with oscillations
Creator alignment	0% (no allocation)	5% (slow vest)
Holder retention	None	Vesting + farming eligibility
Protocol revenue model	1% bonding curve fee	1% fee + flush revenue
Typical token lifespan	24–48 hours	90+ days
Key innovation	Bonding curve	**Competitive farming flywheel**

Conjecture 6.12 (Long-Term Volume Ratio)**.** *Under competitive farming, the ratio of 30-day post-graduation volume to pre-graduation volume satisfies:*

$$\frac{V_{post\text{-}30d}}{V_{pre}} > 2 \tag{36}$$

compared to Pump.fun's estimated ratio of < 0.1.

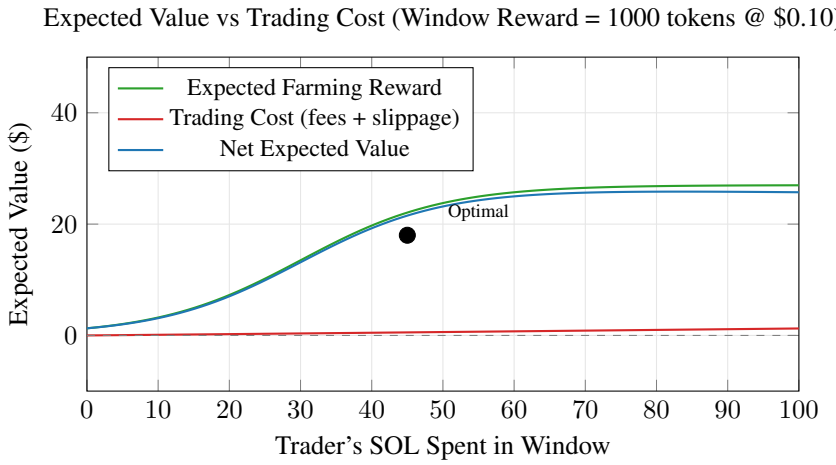

Figure 9: Expected value analysis for a single window. Traders optimize spending where marginal reward equals marginal cost. The curve shape creates a "sweet spot" around 40–50 SOL for this example parameterization.

6.7.6 *Commit-Reveal Randomness*

The farming mechanism requires unpredictable winner selection. We use a commit-reveal protocol:

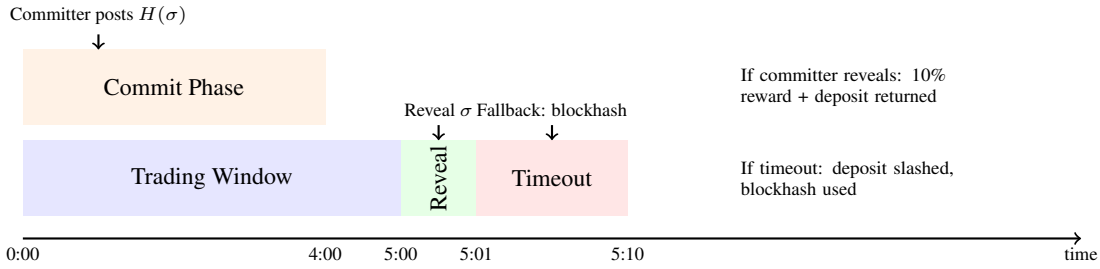

Figure 10: Commit-reveal timeline for farming randomness. Committers must commit before knowing the final top-10, preventing manipulation.

Proposition 6.13 (Committer Incentive Compatibility)**.** *With deposit D slashed on non-reveal and reward $0.1 \cdot R_w$ on reveal, rational committers always reveal:*

$$\mathbb{E}[\textit{reveal}] = 0.1 \cdot R_w + D > D = \mathbb{E}[\textit{no reveal}] \tag{37}$$

The committer's dominant strategy is to reveal, as $0.1 \cdot R_w > 0$.

Remark 6.14 (Why Not Just Use Blockhash?)**.** *Solana blockhashes are manipulable by validators (who can choose which transactions to include). The commit-reveal adds a layer of unpredictability: even if a validator controls the blockhash, they cannot predict σ committed before the window ends.*

6.7.7 *90-Day Reward Decay*

The farming reward per window decreases over time as the pool depletes:

Remark 6.15 (Constant Incentive Design)**.** *By dividing remaining tokens by remaining windows (rather than using a fixed schedule), we ensure every window has equal expected value. This prevents "front-running" where traders concentrate activity in early high-reward windows.*

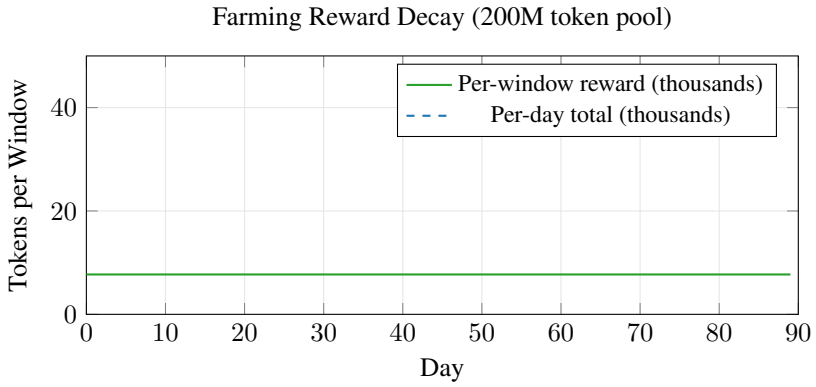

Figure 11: With "remaining pool / remaining windows" distribution, per-window rewards are constant at ∼7,716 tokens. This creates consistent incentives throughout the 90-day period.

6.8 Game-Theoretic Analysis

Theorem 6.16 (Graduation Game Equilibrium)**.** *The virtual pool graduation game has a unique subgame-perfect equilibrium where:*

1. *Creators launch only tokens with genuine community interest*
2. *Buyers participate based on honest valuation (no manipulation premium)*
3. *Manipulators are excluded by negative expected profit*
4. *Flush crankers ensure timely cleanup of failed pools*

Proof Sketch. By backward induction:

- At $t = 1$ hour, any non-graduated pool is flushed (cranker incentive)
- Manipulation cycles increase T^* while accumulating fees (Theorem 6.5)
- Expected manipulation profit is negative for sufficiently large χ
- Only pools with organic demand can graduate
- Creators anticipate this and only launch viable tokens □

□

Table 5: Stakeholder Incentive Alignment

Actor	Aligned Incentive	Misaligned Behavior Prevented
Creator	5% allocation (slow vest)	Rug pull: 2%/hr vest, must graduate first
Early Buyer	Lower bonding price	Whale accumulation: 2.5% wallet cap
Triggerer	0.1% of raised SOL	Gaming: first over threshold only
Holder	Farming rewards	Immediate dump: 5%/hr vest
Protocol	1% fees + flush revenue	Bad pools: 1hr flush = revenue
Cranker	0.1% flush reward	Missing flushes: competitive market

Open Problem 6.17 (Optimal Deadline)**.** *Is 1 hour the welfare-maximizing deadline? Shorter deadlines increase urgency but may disadvantage pools in different timezones. Longer deadlines reduce flush revenue but allow more organic discovery.*

Open Problem 6.18 (Churn Detection Evasion)**.** *Can sophisticated attackers evade churn detection by distributing manipulation across many wallets? What is the cost of such Sybil attacks given the 2.5% wallet limit?*

7 The x402 Integration

Note: This section describes a proposed design, not an implemented system.

We've established the theory: adaptive curves outperform static ones, GBC provides a smooth parameterization, Q-learning offers a path to adaptation, and tiered access creates sustainable economics. But how do agents actually *use* this system?

The answer matters because agents aren't humans. They don't have browser extensions. They don't click buttons. They make HTTP requests to APIs. If our adaptive AMM can't speak HTTP, agents can't use it.

The X402 protocol bridges this gap.

7.1 x402 Protocol Overview

Before we describe integration, let's understand what X402 actually does. It's deceptively simple—activating a single HTTP status code that's been reserved since 1992 but never standardized.

The X402 protocol [x402 Foundation, 2025] enables HTTP-native payments:

1. Client requests resource
2. Server responds `402 Payment Required` with payment details
3. Client pays (e.g., USDC on Base) via a facilitator
4. Client retries with payment proof in headers
5. Server grants access

This flow is designed for AI agents accessing APIs. It's deliberately minimal: no accounts, no sessions, no OAuth tokens. Just pay and proceed. An agent can discover a new API, pay for access, and use it—all in seconds, without human intervention.

We propose extending this pattern to AMM access. Instead of paying for API calls, agents pay for swap execution. Instead of receiving data, they receive confirmation that their trade executed.

7.2 Proposed Architecture

How does an HTTP request become an on-chain swap? The architecture requires a bridge between the HTTP world and the blockchain world. We call this bridge the AEX HTTP Server.

7.3 Design Questions

The architecture diagram makes it look simple, but implementation raises hard questions. We list them not to dismiss the approach but to be honest about what needs solving.

Trust model: The HTTP server is a centralized component. This is uncomfortable in a decentralized system. How do we prevent it from censoring, front-running, or otherwise abusing its position? Options include:

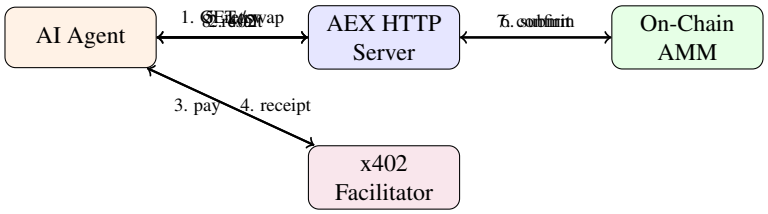

Figure 12: Proposed x402 integration flow. The HTTP server is an off-chain component that mediates between agents and the on-chain AMM.

- Multiple competing servers (reduces censorship but not front-running)
- Commit-reveal schemes (adds latency)
- Trusted execution environments (hardware trust assumptions)

Latency: The full flow involves multiple round trips. For latency-sensitive applications (arbitrage), this may be prohibitive.

Failure modes: What happens if the facilitator fails between payment and execution? The agent has paid but received nothing. Refund mechanisms are needed.

7.4 Liquidity-as-a-Service

The x402 integration enables something beyond just swaps. Once you have a payment-enabled API, you can sell anything agents want to buy. We envision a "Liquidity-as-a-Service" offering where pools sell not just swaps but also information.

- **Oracle data**: TWAP prices with confidence intervals
- **Analytics**: Volume, LP returns, volatility metrics
- **Parameter advice**: "What κ should I use for a similar pool?"

Each service would be priced via x402.

7.5 Integration with Agent Registries

A complete AEX-402 deployment would register itself in on-chain agent registries like AEAMCP [OpenSVM, 2025]. The registration would include:

- **Agent ID**: Unique identifier for the pool
- **Service endpoints**: The x402-enabled HTTP API URL
- **Capabilities**: `["swap", "add_liquidity", "remove_liquidity", "twap_oracle", "analytics"]`
- **Extended metadata URI**: Link to full API specification, supported tokens, fee structure
- **Status**: Active/paused state

This enables powerful discovery patterns. An agent seeking to swap USDC for SOL could query the registry for all agents with "swap" capability supporting those tokens, compare their current parameters (fees, κ, liquidity depth), and route to the best option. The registry becomes a decentralized order router.

The A2A protocol [Linux Foundation, 2025] provides the communication layer. When Agent A discovers Pool P through the registry, it initiates an A2A dialogue:

1. A sends a `QueryCapabilities` message to P's registered endpoint
2. P responds with current state: reserves, κ, fees, expected slippage for various trade sizes
3. A evaluates the quote against its Preferences
4. If favorable, A sends a `ProposeSwap` message with amount and minimum output
5. P validates, executes on-chain, and returns confirmation

This is dramatically more sophisticated than current AMM interaction, where users simply submit transactions and hope for the best. The agent-to-agent protocol enables negotiation, conditional execution, and coordinated multi-pool routing.

Conjecture 7.1 (Information Pricing)**.** *The welfare-maximizing price for information service S is proportional to the entropy reduction it provides:*

$$p^*(S) \propto I(X; S) \tag{38}$$

where X is the decision-relevant unknown and I denotes mutual information.

Discussion: This is a standard result from information economics [Cover and Thomas, 2006]. The challenge is estimating mutual information in practice.

8 Agent Economics and Decision Theory

Having described the infrastructure (adaptive curves, learning mechanisms, tiered access), we now examine how agents actually *decide* whether to use it. This matters because agent preferences determine pool equilibrium: if agents find the pool unattractive, they won't trade, and no amount of clever mechanism design helps.

8.1 The Agent Decision Problem

When an AEA considers a swap, it faces a decision problem. Should it trade now, wait for better conditions, or use a different venue entirely? The agent's DecisionMaker must compare expected utility across options.

Definition 8.1 (Agent Utility Function)**.** *An AEA with wealth w and token holdings $\mathbf{h} = (h_0, h_1, \ldots)$ derives utility from a trade that changes holdings by $\Delta\mathbf{h}$ at cost c:*

$$U(\Delta\mathbf{h}, c) = V(\mathbf{h} + \Delta\mathbf{h}) - V(\mathbf{h}) - c \tag{39}$$

where V is the agent's valuation function over token portfolios.

Different agents have different V functions:

- **Arbitrageurs**: $V(\mathbf{h}) = \sum_i p_i h_i$ where p_i is the external market price. They care only about immediate dollar value.
- **Portfolio managers**: V includes risk-adjustment, perhaps $V = \mathbb{E}[\text{return}] - \lambda \cdot \text{Var}[\text{return}]$ for risk aversion λ.
- **Payment agents**: V is step-function-like: they need specific tokens for specific payments and value them highly until the requirement is met.

This heterogeneity is a feature for mechanism design. Different valuations enable price discrimination: we can charge more to agents who value execution quality highly.

8.2 Preferences and Marginal Utility

The AEA framework [Minarsch et al., 2020] implements preference evaluation through a *Preferences* object that computes marginal utility scores. Given a proposed transaction with terms T (price, quantity, fees), the agent computes:

$$\text{MU}(T) = U(\Delta\mathbf{h}(T), c(T)) \tag{40}$$

and accepts if $\text{MU}(T) > 0$.

This simple accept/reject rule can be augmented with:

- **Opportunity cost**: Compare against expected utility of waiting
- **Information value**: Factor in learning from observing prices
- **Strategic delay**: In repeated games, current behavior affects future offers

Proposition 8.2 (Optimal Trade Timing)**.** *An agent with discount factor δ and belief μ_t about future price distribution should trade now if:*

$$U(\Delta\mathbf{h}, c) \geq \delta \cdot \mathbb{E}_{\mu_t}[\max\{U(\Delta\mathbf{h}', c'), 0\}] \tag{41}$$

where the right side is the expected continuation value.

Proof. This is the standard optimal stopping condition. Trade now if immediate utility exceeds discounted expected future utility. □

The challenge for agents is that μ_t depends on pool behavior, which in our setting depends on learning, which depends on agent trades. This circularity makes equilibrium analysis subtle.

8.3 Agent Communication and Coordination

In multi-agent systems, agents don't just trade—they communicate. The AEA framework uses asynchronous message passing with structured protocols: messages are directed to recipients and follow defined dialogue patterns.

For AMM interaction, relevant protocols include:

- **Query protocols**: Agent asks for current pool state (reserves, fees, κ)
- **Trade protocols**: Agent proposes swap, receives quote, confirms or rejects
- **LP protocols**: Agent expresses interest in providing liquidity, negotiates terms

These protocols are important for adaptive AMMs because they create the information flow the learning algorithm observes. A well-designed protocol generates informative signals; a poorly designed one creates noise or enables manipulation.

Open Problem 8.3 (Protocol Design for Learning)**.** *What communication protocols maximize the information value for pool adaptation while minimizing manipulation risk? Can we design protocols that are incentive-compatible for agents to reveal true preferences?*

8.4 Behavioral Patterns and Agent Types

Empirically, agents exhibit recognizable behavioral patterns. We can categorize:

Reactive agents (Handlers): Respond to external events (price movements, arbitrage opportunities). They create most trading volume and their behavior is somewhat predictable.

Proactive agents (Behaviours): Initiate actions based on internal schedules or goals. Portfolio rebalancers, yield optimizers, and payment agents fall here. Their behavior depends on external schedules (payment dates, rebalance intervals) that may be unobservable.

Learning agents: Adapt their strategies based on experience. These are the most complex to model because their future behavior depends on their learning, which depends on their past experience, which depends on pool behavior.

Understanding this mix matters for pool design. A pool dominated by reactive arbitrageurs behaves differently from one dominated by proactive portfolio managers. The optimal κ may differ, and the information structure differs.

8.5 Game-Theoretic Considerations

Agents are strategic: they consider how their actions affect others and anticipate responses. This creates game-theoretic complexity [Shoham and Leyton-Brown, 2008].

Definition 8.4 (Trading Game)**.** *The one-shot trading game has:*

- *Players: Traders* $\{1, \ldots, n\}$ *and the pool*
- *Strategies: Trade amounts* $\{t_i\}$ *and pool parameters* κ
- *Payoffs: Trader* i *receives* $u_i(t_i, t_{-i}, \kappa)$*; pool receives fees*

Conjecture 8.5 (Equilibrium Existence)**.** *Under mild continuity assumptions, the trading game has a Nash equilibrium in mixed strategies.*

Discussion: Existence follows from standard fixed-point theorems. The interesting questions are uniqueness (are there multiple equilibria?), efficiency (how much welfare is lost relative to optimum?), and stability (do learning dynamics converge to equilibrium?).

What we can say is that agents with accurate models of pool behavior will outperform those with inaccurate models. This creates pressure for sophisticated agents—and, paradoxically, may make the pool more predictable as sophisticated agents arbitrage away inefficiencies.

9 Limitations and Open Problems

We believe honest acknowledgment of limitations strengthens rather than weakens a paper. This section catalogues what we don't know, what could go wrong, and what future work is needed.

9.1 Theoretical Gaps

The tracking bound is unproven. Conjecture 4.5 is our best current understanding, but a rigorous proof requires:

- Precise specification of the dependence of P_t, R_t on σ_t
- Lipschitz bounds that may not hold in practice
- Analysis of error accumulation that we have not completed

Without this proof, we cannot guarantee the learning algorithm converges or even makes progress.

Equilibrium in the market game is uncharacterized. We know equilibria exist (by standard fixed-point theorems) but have not characterized them. It's possible the equilibrium has undesirable properties—e.g., LPs withdraw, or traders avoid the pool.

The GBC family may have pathologies at extreme parameters. We've proven good behavior at $\kappa = 0$ and $\kappa = 1$ but intermediate values need more analysis, especially near $\kappa = 1$ where reserve depletion becomes possible.

9.2 Practical Challenges

Cold start problem. A new pool has no data from which to learn. What should the initial κ be? How long until learning produces useful adaptation? During this period, the pool may perform poorly, discouraging LPs.

Computational cost. We estimate 2,000-5,000 CU per swap (vs. 1,000 for constant product). In high-throughput scenarios, this overhead may matter. The learning updates are additional cost, though they can be batched.

Oracle reliability. Some state features (volatility estimates) require oracle data. Oracle failures or manipulation could corrupt learning.

Governance transition. Existing pools use governance for parameter changes. Transitioning to autonomous adaptation requires LP buy-in. Why would LPs trust an algorithm they don't understand?

9.3 What Could Go Wrong

We enumerate failure modes:

1. **Learning divergence**: The Q-learning algorithm could diverge, oscillating between extreme κ values. Bounded actions limit damage but don't prevent poor performance.
2. **Adversarial manipulation**: Traders could deliberately create observations that bias learning. We have not quantified the cost of such attacks or proven robustness.
3. **Regime detection lag**: If market conditions change faster than learning adapts, the pool is always "behind," using yesterday's optimal curve today.
4. **Correlated failures**: In a crisis (market crash, depeg event), many agents may act similarly, causing correlated stress on the adaptation mechanism precisely when correct behavior matters most.
5. **x402 server compromise**: The HTTP server is a central point of failure. Its compromise could enable theft or censorship.

9.4 Comparison to Alternatives

We have not empirically compared AEX-402 to alternatives:

- **Curve v2**: Uses an internal oracle and heuristic recentering. How does our RL approach compare?
- **Multiple pools**: Instead of adapting, use multiple pools (one Uniswap, one Curve) and route optimally. Is this simpler and equally effective?
- **Governance**: Traditional governance may be slow but involves human judgment. Is autonomous adaptation actually better?

Answering these questions requires simulation or deployment, which we have not done.

9.5 Future Work

We identify the most important directions:

1. **Prove the tracking bound**: Complete the analysis of Conjecture 4.5 or identify conditions under which it fails.
2. **Simulate extensively**: Before deployment, simulate on historical data from Uniswap, Curve, and other AMMs. Measure actual welfare improvement.

3. **Implement and test x402 integration**: Build the HTTP server, test with real agents, measure latency and failure rates.

4. **Design robust learning**: Investigate adversarial robustness. Can we design mechanisms where manipulation is provably costly?

5. **Multi-pool learning**: Can pools share experience? This could accelerate learning and reduce cold-start problems.

10 Related Work

Automated Market Makers. The constant product formula was introduced by Uniswap [Adams et al., 2020]. StableSwap [Egorov, 2019] optimized for correlated assets. Uniswap v3 [Adams et al., 2021] enabled concentrated liquidity. Balancer [Martinelli and Mushegian, 2019] generalized to weighted pools. Curve v2 [Egorov, 2021] introduced dynamic recentering—the closest prior work to adaptive curves. Unlike Curve v2's heuristic approach, we propose learning-based adaptation, but we have not proven superiority.

Reinforcement Learning Theory. Q-learning convergence in stationary MDPs is well-understood [Watkins and Dayan, 1992, Even-Dar and Mansour, 2003]. Non-stationary MDPs are harder; Sutton and Barto [2018] discusses tracking but without tight bounds. Our Conjecture 4.5 attempts to fill this gap but remains unproven.

Mechanism Design. Tiered pricing relates to quality-discrimination [Mussa and Rosen, 1978]. Our analysis (Theorem 5.3) is standard; the novelty is application to AMM execution tiers.

Internet Payments. The x402 protocol [x402 Foundation, 2025] motivates our work. We are not aware of prior work on x402-native financial infrastructure.

11 Conclusion

We have presented AEX-402, a framework for adaptive automated market making. Our contributions are:

Proven:

- Static AMMs suffer $\Omega(T)$ welfare loss under regime-switching (Theorem 2.5)
- The GBC family provides valid curve interpolation with efficient swap computation (Theorem 3.5)
- Tiered access is incentive-compatible under standard assumptions (Theorem 5.3)

Proposed:

- Q-learning architecture for on-chain adaptation
- x402 integration for agent-native access
- Tracking bound conjecture for non-stationary convergence

Unresolved:

- Rigorous convergence guarantees for learning
- Adversarial robustness
- Empirical comparison to alternatives
- Complete x402 implementation

The thesis of this paper—that AMMs should learn from experience—remains compelling despite these gaps. Markets change; exchange mechanisms should adapt. The question is not whether to adapt, but how.

We offer AEX-402 not as a finished solution but as a research program. The theoretical framework is partially complete; the practical challenges are substantial; the vision of autonomous, learning, agent-native financial infrastructure remains to be realized.

11.1 A Research Agenda

We conclude with a prioritized research agenda:

1. **Year 1**: Prove tracking bounds. Simulate extensively. Build x402 prototype.
2. **Year 2**: Deploy on testnet. Measure against baselines. Iterate on learning design.
3. **Year 3**: Mainnet deployment with safeguards. Accumulate real-world experience. Publish empirical results.

We invite collaboration on these problems. The agentic economy is coming; its infrastructure should be built carefully, with theoretical foundations and honest acknowledgment of uncertainty.

References

H. Adams, N. Zinsmeister, and D. Robinson. Uniswap v2 Core. Technical report, Uniswap, 2020.

H. Adams, N. Zinsmeister, M. Salem, R. Keefer, and D. Robinson. Uniswap v3 Core. Technical report, Uniswap, 2021.

M. Egorov. StableSwap: Efficient Mechanism for Stablecoin Liquidity. Technical report, Curve Finance, 2019.

M. Egorov. Automatic Market-Making with Dynamic Peg. Technical report, Curve Finance, 2021.

x402 Foundation. x402: An Open Standard for Internet-Native Payments. Technical report, Coinbase, Cloudflare, 2025.

F. Martinelli and N. Mushegian. Balancer Whitepaper. Technical report, Balancer Labs, 2019.

C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

E. Even-Dar and Y. Mansour. Learning Rates for Q-learning. *Journal of Machine Learning Research*, 5:1–25, 2003.

T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 2nd edition, 2006.

M. Mussa and S. Rosen. Monopoly and Product Quality. *Journal of Economic Theory*, 18:301–317, 1978.

P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash Boys 2.0: Frontrunning in Decentralized Exchanges. In *IEEE S&P*, 2020.

D. Minarsch, M. Hosseini, M. Favorito, and J. Ward. Autonomous Economic Agents as a Second Layer Technology for Blockchains: Framework Introduction and Use-Case Demonstration. In *Crypto Valley Conference on Blockchain Technology (CVCBT)*, IEEE, 2020.

L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2008.

M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2nd edition, 2009.

OpenSVM. AEAMCP: Solana AI Registries for Agent and MCP Server Discovery. Technical report, OpenSVM, 2025. https://aea.network

Linux Foundation. A2A Protocol: An Open Standard for Agent-to-Agent Communication. Technical specification, 2025. https://a2a-protocol.org

A GBC Swap Computation Details

We provide additional detail on Newton-Raphson iteration for swap computation.

A.1 Setup

Given: reserves (x, y), curvature κ, scale D, input Δx. Find: output Δy such that $\mathcal{I}_\kappa(x + \Delta x, y - \Delta y) = 1$.

Let $x' = x + \Delta x$ (known) and $y' = y - \Delta y$ (unknown). Define:

$$f(y') = \left(\frac{x'y'}{D^2}\right)^{1-\kappa} \cdot \left(\frac{x' + y'}{2D}\right)^{2\kappa} - 1 \tag{42}$$

We seek y' such that $f(y') = 0$.

A.2 Newton-Raphson Update

The derivative is:

$$f'(y') = \mathcal{I}_\kappa(x', y') \cdot \left[\frac{(1-\kappa)}{y'} + \frac{2\kappa}{x' + y'}\right] \tag{43}$$

The Newton update is:

$$y'_{n+1} = y'_n - \frac{f(y'_n)}{f'(y'_n)} \tag{44}$$

A.3 Initialization

We initialize with the constant-product solution:

$$y'_0 = \frac{xy}{x'} = \frac{xy}{x + \Delta x} \tag{45}$$

This is exact for $\kappa = 0$ and provides a reasonable starting point for other κ.

A.4 Convergence Criterion

We iterate until $|f(y'_n)| < \epsilon$ for tolerance $\epsilon = 10^{-12}$ (sufficient for 64-bit precision).

A.5 Iteration Count

In practice, convergence typically requires:

- $\kappa \in [0, 0.3]$: 3-5 iterations

- $\kappa \in [0.3, 0.7]$: 4-7 iterations
- $\kappa \in [0.7, 0.95]$: 6-10 iterations

Higher κ requires more iterations because the constant-product initial guess is farther from the true solution.

B Proof of Theorem 2.5

We provide a complete proof of the static curve lower bound.

Proof. Let σ_t denote the regime at time t, with $\sigma_t \in \{\sigma_H, \sigma_L\}$. The regime process is an ergodic Markov chain with transition matrix P.

Step 1: Stationary Distribution

The stationary distribution satisfies $\pi P = \pi$. Solving:

$$\pi_H = \frac{q}{p+q}, \quad \pi_L = \frac{p}{p+q} \tag{46}$$

By the ergodic theorem, for any bounded function g:

$$\lim_{T\to\infty} \frac{1}{T} \int_0^T g(\sigma_t)\, dt = \pi_H g(\sigma_H) + \pi_L g(\sigma_L) \quad \text{a.s.} \tag{47}$$

Step 2: Welfare Decomposition

For any static curve c:

$$\mathbb{E}\left[\int_0^T [W(c^*(\sigma_t), \sigma_t) - W(c, \sigma_t)]\, dt\right] = T \cdot [\pi_H \epsilon_H(c) + \pi_L \epsilon_L(c)] + o(T) \tag{48}$$

where $\epsilon_\sigma(c) = W(c^*(\sigma), \sigma) - W(c, \sigma) \geq 0$.

Step 3: Non-Degeneracy

Since $c^*(\sigma_H) \neq c^*(\sigma_L)$ by assumption, for any fixed c, at least one of $\epsilon_H(c)$ or $\epsilon_L(c)$ is strictly positive.

Let $\delta(c) = \min\{\pi_H \epsilon_H(c), \pi_L \epsilon_L(c)\}$. We have $\delta(c) > 0$ for all c.

Step 4: Uniform Lower Bound

Define $\delta = \inf_c \delta(c)$. Since the curve space is compact and $\delta(c)$ is continuous, this infimum is achieved and $\delta > 0$.

Therefore:

$$\mathbb{E}[\mathcal{G}(c, T)] \geq \delta \cdot T \tag{49}$$

for all static curves c. □

C State Encoding Details

We provide details on the proposed state encoding for Q-learning.

C.1 Feature Definitions

Volatility bin: Computed from recent price changes. Let $r_t = \log(p_t/p_{t-1})$ be log returns. Define:

$$\hat{\sigma} = \sqrt{\frac{1}{n} \sum_{i=0}^{n-1} r_{t-i}^2} \tag{50}$$

Bin as: low ($\hat{\sigma} < \sigma_L$), medium, high ($\hat{\sigma} > \sigma_H$).

Trend bin: Computed from price direction. Let $\bar{r} = \frac{1}{n}\sum_{i=0}^{n-1} r_{t-i}$. Bin as: down ($\bar{r} < -\delta$), flat, up ($\bar{r} > \delta$).

Spread bin: Computed from bid-ask or pool imbalance. Let imb $= |x - y|/(x + y)$. Bin as: tight (imb $< \epsilon_1$), medium, wide (imb $> \epsilon_2$).

C.2 Threshold Selection

Thresholds ($\sigma_L, \sigma_H, \delta, \epsilon_1, \epsilon_2$) should be calibrated to historical data to achieve roughly uniform bin populations. This is an empirical design choice.

C.3 Update Frequency

State should be recomputed at most once per block (400ms on Solana). More frequent updates waste computation; less frequent updates miss information.