Anatomy of a Verdict: How SimplyCodes Adjudicates Truth in a Broken Economy

A Technical Whitepaper

Michael Quoc and Bri Stanback

SimplyCodes

February 2026

Abstract

SimplyCodes operates a coupon verification system spanning over 400,000 merchants in an environment where 40–60% of codes on the public internet are dead, restricted, or misleading. This paper details the architecture that adjudicates promotional truth through three independent verification layers (Automated Code Testing, a reputation-weighted Human Consensus Network, and real-time Fleet Signal telemetry from browser extension users), unified by a design principle borrowed from distributed systems: Byzantine Fault Tolerance, the assumption that any single source may be lying, malfunctioning, or mistaken. We describe how these layers converge into a single Health Score, a confidence measure derived from an event stream rather than stored as a static value, that decays at variable rates depending on code type and is traceable through six forensic dimensions from verdict back to raw evidence. We introduce the Confident No (a verified absence of codes that delivers cognitive closure at checkout) and the Proof Packet (a structured evidence bundle designed for auditability and machine consumption in the agentic economy). We conclude with the structural moats that make this system irreplicable by design: fifteen years of accumulated merchant signal, a reputation economy of tens of thousands of human verifiers, and an edge case corpus that cannot be replicated without experiencing the failures firsthand.

1 The Hidden Job

The most valuable thing we can tell you is that there is nothing to find.

That sentence contains our entire product philosophy. Most people assume a coupon code is a simple database lookup: enter string, receive discount. The reality is far messier. A coupon code is a conditional, ephemeral, adversarial contract living in a hostile environment. It can be true at 9am and false by noon. It can work for one cart and fail for another. It can exist in a merchant's system but apply to nothing you want to buy.

The job we do is not finding codes. The job is ending the search—delivering the certainty that allows you to act. When we tell you a code works, that is a verdict backed by evidence. When we tell you no codes exist, that verdict is equally valuable: you have permission to stop searching and complete your purchase with confidence.

We call this the Confident No. It is not a failure state. It is the product. A verified absence of codes saves you ten minutes of clicking through SEO garbage. That time saved, that anxiety eliminated, that cognitive closure delivered—that is what you hire us for.

This document exposes the machine that produces these verdicts. We will follow a single code through our verification gauntlet—from chaotic ingestion to confident output. By the end, you will understand why our position in this market is not a marketing claim. It is a structural consequence of architecture.

2 Part I: The Entropy

Why This Is Impossible

2.1 The Scale of Chaos

SimplyCodes covers over 400,000 merchants—an order of magnitude beyond typical competitors. Each merchant has bespoke checkout logic, custom cart implementations, and idiosyncratic promo code handling. There is no standard. Thousands of codes are created and killed every day, most without announcement.

At any given moment, 40–60% of codes on the public internet are dead, restricted, or misleading. Search for any major retailer's promo codes and you will find pages of recycled garbage from 2019, affiliate sites republishing expired deals, and AI-generated content farms inventing codes that never existed.

This is the environment we operate in. This is what we must adjudicate.

2.2 The Five Vectors of Entropy

Merchant Entropy. Retailers expire codes silently. They add hidden constraints. They break their own cart logic during site updates. A code that worked Monday can die Tuesday with no announcement, no changelog, no notification.
Temporal Decay. Codes rot. A flash sale code lives six hours. A seasonal promotion lasts weeks. An evergreen welcome code might survive months. Each type decays at a different rate, and the decay is invisible until the moment of failure.
Contextual Validity. The same code string can have different truth values depending on what is in your cart. "SAVE20" works—unless you have sale items, or you are not a new customer, or your subtotal is under fifty dollars, or you are shipping to Hawaii. The code exists. It just does not apply to you.
Human Error. Merchants fat-finger expiration dates. Affiliates publish stale codes. Users misremember or mistype. A code entered as "SUMMER52" instead of "SUMMER25" now exists as two entries in the ecosystem—one real, one phantom.
Adversarial Noise. SEO farms publish fake codes to capture search traffic. Affiliates recycle dead codes because removing them costs more than leaving them. The incentive structure of the internet actively rewards pollution.

2.3 The Checkout Crucible

The moment of code application is high-stakes, high-emotion. The user has invested time finding the code, building the cart, entering payment details, reaching the final step. A failed code at this moment does not just cost money—it costs trust. It is a small betrayal at the worst possible time, and it is the experience that the entire coupon industry has normalized.

The question is not "Is this code valid?" The question is: "Will this code work for my cart, right now?"

These are different questions with different answers:

Question	What It Means	Difficulty
Does this code **exist**?	Is it in the merchant's system?	Easy
Does this code **function**?	Does it produce any discount?	Medium
Will this code **work for me**?	Does it apply to my specific cart?	Hard

Most verification systems answer the first question. We reliably answer the second. We are actively building toward the third.

That is the chaos. Here is the machine we built to adjudicate it.

3 Part II: The Machine

The Verification Stack

We do not "scrape" codes. We adjudicate them. Our system is designed around a principle borrowed from distributed systems: Byzantine Fault Tolerance (BFT)—the assumption that any single source may be lying, malfunctioning, or mistaken.

Truth emerges only when independent verification nodes converge.

Our verification stack has three layers, each operating independently, each producing evidence that feeds into the final verdict.

3.1 Layer 1: Automated Code Testing

We operate automated verification systems that simulate real checkout sessions across our merchant network. These are not scrapers. They are headless browsers that navigate to real merchant storefronts, add products to carts, apply codes, and capture the results—exactly as a human shopper would.

We run tens of thousands of automated tests every day across our merchant coverage.

3.1.1 Tiered Intelligence

Different merchants require different approaches. We deploy a tiered system that selects the most efficient verification method for each merchant:

Platform Adapters. Major e-commerce platforms share semi-standardized checkout structures. For these, we build deterministic adapters that know exactly where the promo field lives, how to read the cart response, and what success or failure looks like. This covers a large portion of online retail because thousands of merchants run on a handful of platforms.

Heuristic Pattern Matching. For merchants on custom or less common platforms, the system uses heuristics: scan the page for common checkout patterns, locate input fields near words like "promo" or "coupon," diff the page after application to detect price changes or error messages. This approach exploits the fact that while every checkout is different, most follow common UX conventions.

AI-Powered Navigation. For truly custom checkouts where heuristics fail, the system uses an LLM to read the page as a human would. The bot captures the checkout DOM and asks the model: "Where is the promo code input? What changed after application? What error appeared?" This agentic layer can navigate arbitrary checkout flows without pre-built adapters.

The tiering is what makes the system tractable. We are not building one universal bot. We pick the cheapest, most reliable method for each merchant and fall back to more expensive intelligence when needed.

3.1.2 The Test Pipeline

Each test follows a structured pipeline:

1. **Session Initialization.** A headless browser launches targeting the merchant's storefront with realistic browser fingerprints.
2. **Product Discovery.** The bot navigates to a product page and adds an item to cart.
3. **Cart Simulation.** The bot navigates to checkout and locates the promo code input field.
4. **Code Application.** The bot applies the code using multiple strategies—typing and pressing Enter, submitting forms, clicking adjacent buttons. Different merchants require different approaches.
5. **Result Validation.** The bot parses the page for success or failure indicators—discount applied, error messages, or ambiguous states.
6. **Evidence Capture.** A screenshot is taken as proof of outcome. The result is logged with full diagnostic metadata.

3.1.3 The Failure Taxonomy

When a test fails, we do not simply record "failed." We diagnose why:

Failure Mode	Meaning	System Response
Bot detection	Cloudflare, CAPTCHA, or similar blocked access	Route to human verification
Authentication required	Account creation or login needed	Flag as restricted access
Checkout UI changed	Promo field not found in expected location	Alert for adapter update
Code expired	Merchant confirmed code is dead	Confidence drops to floor
Promotion condition	Code exists but has scope constraints	See honest disclosure below

3.1.4 What We Verify and What We Don't

Here is something competitors will not tell you. We will.

A code can return a product condition message and we still recognize it as functional. Why? Because the code exists and the merchant accepted it—it just does not apply to the test cart. Our automated tests reliably answer "Does this code exist and function in the merchant's system?" They do not yet answer "Will it work for your specific cart?"

This is a known limitation. A code can display a healthy confidence score but fail for users with sale items. We are actively building contextual applicability modeling to address this gap—more on this in Part V. We mention it here because transparency about limitations is part of our architecture, not a weakness to hide.

There is a related limitation worth stating plainly: we verify existence, not arithmetic. Confirming that "20% off" actually produces a 20% discount on your specific cart total is harder than it sounds. Merchants calculate discounts differently, display results differently, and sometimes the numbers do not add up the way a customer expects. We catch existence and function. Arithmetic verification is an active area of development.

One more: merchants actively resist verification. Some merchants make automated testing deliberately difficult through bot detection, CAPTCHAs, dynamic checkout flows, and anti-automation defenses. We measure this resistance—both system friction (network latency, blocking) and UX friction (confusing flows, soft blocks). When verification is hard, that itself is a signal about a merchant's posture toward transparency. When automated testing is blocked entirely, the multi-layer architecture catches what bots cannot.

3.2 Layer 2: Human Verification Network

Bots can simulate checkouts, but they cannot reason about edge cases. For that, we maintain a global network of human verifiers—a community of tens of thousands of trained contributors producing millions of monthly verification actions.

This is not a contractor pool. It is a reputation economy. Contributors accumulate trust scores based on their verification accuracy over time. Their earnings and status depend on precision. Long-tenured, high-accuracy verifiers carry more weight in consensus than new or inconsistent ones.

3.2.1 The Consensus Protocol

Verifiers operate under blind voting—they cannot see each other's assessments until consensus is reached. This prevents herding behavior and ensures independent verification.

Negative Consensus (Invalidation). Multiple independent "No" votes within a short window are required to kill a code. This prevents a single bad actor or mistaken user from poisoning the database.

Positive Consensus (Validation). A single high-trust verifier with strong evidence (a valid screenshot showing the discount applied) can boost a code's confidence significantly. The threshold is lower for positive signals because false positives are self-correcting—users encountering dead codes will quickly submit negative votes.

3.2.2 Trust Weighting

Not all votes carry equal weight. Our system assigns trust tiers based on each verifier's historical accuracy, consistency, and tenure:

- **Administrative verifiers** have authoritative override capability
- **High-trust contributors** carry amplified weight based on proven track records
- **Standard contributors** carry baseline weight
- **New or inconsistent contributors** carry reduced weight
- **Anonymous signals** carry minimal weight

All verification submissions require screenshot proof. The screenshot must show the cart, the discount applied (or error message), and identifiable merchant branding. This evidence becomes part of the code's audit trail.

3.2.3 Structured Data Capture

Our internal verification team uses specialized tooling that transforms verification from subjective judgment into structured data capture. Rather than simply asking "did it work?", the tooling forensically parses the checkout DOM to capture full cart context—items, prices, subtotal, merchant response text. It packages this structured data into what we call a sensing event that feeds directly into the verification pipeline.

This transforms a subjective observation into auditable, machine-readable evidence. The system architecturally separates the witness (what was observed during code application) from the judge (the downstream system that computes truth from accumulated observations). Witnesses capture evidence. Judges issue verdicts. These roles never blur.

3.3 Layer 3: The Signal Fleet

When a real user with our browser extension successfully checks out, that is the gold standard signal. It is not a simulation—it is ground truth.

What We Capture:

Cart composition at moment of code application. Exact discount applied. Merchant response. Success or failure state. Timestamp and session metadata.

The Flywheel:

More users generate more real-time signal. Better signal produces more accurate confidence scores. More accurate scores build more trust. More trust attracts more users. Each rotation makes the moat deeper.

This is the moat that cannot be purchased. A competitor can build bots. They cannot conjure years of accumulated real-world transaction signal.

Three layers. Independent signals. Convergent verdicts. That is the machine. Now—how does it reach a conclusion?

4 Part III: The Verdict

From Chaos to Confidence

4.1 The Health Score

Every code in our system carries a Health Score: a confidence measure that this code will work right now. Think of it as a freshness-adjusted trust rating.

The score is not static. It decays over time because codes rot. And it is not stored as a fixed value—it is derived from an event stream. Every test, every human verification, every real checkout signal is an event. We call each one a sensing event—a particle of truth. Not a fact, but evidence. The current health score is a computation over the full history of these events, not a number in a database column.

This event-sourced architecture means we can recompute any code's health at any point in time, trace exactly why a score changed, and audit the full chain of evidence behind any verdict.

4.1.1 The Decay Model

We model code freshness using time-based decay with variable rates depending on code type:

Evergreen codes (perpetual welcome offers, student discounts) decay slowly. A verified evergreen code stays healthy for months without revalidation.

Regular codes (seasonal promotions, holiday sales) decay at a moderate rate. Without fresh verification, a regular code's confidence erodes within weeks.

Flash codes (weekend sales, six-hour promotions) decay rapidly. Confidence drops within days or hours.

The decay model ensures that stale data self-identifies. If we have not recently verified a code, the system automatically reduces confidence rather than presenting old data as current.

4.1.2 The Confidence Tiers

Health scores map to confidence tiers that determine how codes are presented to users:

- **High confidence** codes have been recently validated by multiple independent signals. These are displayed prominently.
- **Moderate confidence** codes are aging or have mixed signals. These are presented with appropriate caveats.
- **Low confidence** codes have no recent positive signal. These are de-prioritized or suppressed.
- **Invalidated** codes have been confirmed dead by consensus. These are hidden from users.

4.1.3 The Inputs to Verdict

The Health Score is not a single measurement. It is a convergence of independent signals:

Freshness. When was this code last verified? A test from yesterday carries more weight than one from last month.

Outcome. Did the last test succeed or fail? A recent failure drops health immediately.

Verifier Trust. Weighted consensus from human verifiers, accounting for each contributor's historical accuracy and trust tier.

Automated Test Confidence. Result reliability from our bot infrastructure, factoring in the test conditions and merchant complexity.

Merchant Pattern. Historical reliability of the store. Some merchants expire codes predictably; others are chaotic. Over years of accumulated data, we have built behavioral profiles for hundreds of thousands of merchants.

Decay Clock. Time-based confidence erosion appropriate to the code type.

Fleet Signal. Real-world transaction confirmations from extension users.

Every observation in this system is traceable through six forensic dimensions: who observed it, what code was tested, which merchant, what software captured it, which build version, and a session identifier linking the observation to the final verdict. This chain of custody means any verdict can be forensically audited from conclusion back to raw evidence.

4.2 The Confident No

Most coupon sites optimize for "Yes"—showing you codes. We optimize for closure—ending your search.

When our system returns "No verified codes available," that is not a failure. That is a verdict:

We tested this merchant recently with automated systems. We checked human verifier submissions. We analyzed real-time extension signal. Conclusion: There is nothing to find. Stop searching. Buy with confidence.

This saves you ten minutes of frustration clicking through SEO garbage. That time saved is the product.

4.3 The Proof Packet

Every verdict can generate what we call a Proof Packet—a structured evidence bundle that documents exactly how we reached our conclusion:

```json
{
  "code": "SAVE20",
  "merchant": "example.com",
  "verdict": "VERIFIED",
  "confidence": 0.91,
  "evidence": {
    "automated_test": {
      "timestamp": "2026-02-03T08:23:00Z",
      "result": "FUNCTIONAL",
      "tier": "PLATFORM_ADAPTER",
      "session_id": "a1b2c3d4-e5f6-..."
    },
    "human_consensus": {
      "independent_verifications": 4,
      "positive": 3,
      "weighted_confidence": 0.88
    },
    "fleet_signal": {
      "recent_successes": 47,
      "recent_failures": 3,
      "trend": "STABLE"
    },
    "health": {
      "score": 0.91,
      "decay_rate": "MODERATE",
      "last_verified": "2026-02-03T08:23:00Z",
      "staleness": "FRESH"
    }
  }
}
```

This is what we serve to AI agents. Not a list of codes—a verifiable proof of adjudication.

5 Part IV: The Moat

Why This Cannot Be Copied

5.1 The Numbers

A 2022 Testbirds study tested 500 stores across multiple coupon providers. SimplyCodes had working codes for 334 stores—a 96% coupon availability rate. The next closest competitor had 138 stores. Competitor availability ranged from 34–44%.

We continue to benchmark internally, and the gap has widened since that study. Our automated testing infrastructure, human verifier network, and fleet signal have all scaled significantly since 2022.

Metric	SimplyCodes
Merchants covered	400,000+
Coverage vs. next competitor	∼10x
Working codes per merchant	∼3x average
Daily automated verifications	Tens of thousands
Monthly human verifications	Tens of millions

5.2 The Three Moats

Moat 1: Time-Accumulated Signal. Every verification we run—automated or human—adds to a corpus of merchant behavior patterns. We know which merchants expire codes without warning. We know which merchants have buggy cart logic. We know which code formats each merchant uses. We have historical reliability profiles for over 400,000 stores.

A competitor starting today begins with zero. We have over fifteen years. That gap does not close.

Moat 2: The Human Network. Tens of thousands of verifiers producing tens of millions of monthly verification actions cannot be purchased. This is a reputation economy. Contributors have trust scores, streak bonuses, badge progressions. They are invested in accuracy because their earnings depend on it.

You can hire a thousand contractors on day one. You cannot manufacture years of accumulated trust relationships and behavioral calibration data.

Moat 3: The Edge Case Corpus. Our codebase is not elegant. It is a scar tissue map of every weird merchant behavior we have encountered over the years:

The merchant who uses JavaScript to hide the promo field until you hover over a specific element. The checkout that requires cookies from a specific referrer. The code that only works if you have exactly three items in cart. The flash sale that activates four minutes late because of CDN caching.

This corpus cannot be replicated without experiencing the failures firsthand.

5.3 The Flywheel

Better coverage attracts more users. More users generate more signal. Better signal produces more accurate health scores. More accurate scores build more trust. More trust attracts more users.

Each rotation makes the moat deeper. We are not maintaining a lead—we are compounding it.

6 Part V: The Future

From Verification to Adjudication

6.1 The Shift

We built this system for humans clicking on websites. The next generation is built for AI agents executing transactions.

When your AI assistant says "I found a 20% discount at Nike," it needs to know: Is this code real? Will it work for this specific cart? What is the confidence level? What is the evidence?

That is not a coupon lookup. That is a verification API call.

6.2 What We Are Building

The Applicability Engine. The biggest leap in our roadmap is moving from "Does this code exist and function?" to "Will this code work for this cart?" We are building a system that evaluates promotion scope, conditions, and restrictions against a specific cart context in real time. Does the cart meet the minimum spend? Are the items in qualifying categories? Is the user eligible?

This requires a fundamentally different data architecture. We are moving from flat code records to a constitutional model for promotions—where every code has structured, machine-readable conditions (minimum spend, category inclusions/exclusions, user eligibility requirements) rather than freetext descriptions. These conditions are derived from accumulated test evidence and merchant behavior patterns, then validated against real checkout outcomes.

The goal is a confidence score that answers the hard question: “Given this specific cart, what is the probability this code applies?”

Event-Sourced Truth. Our next-generation promotion architecture extends fifteen years of verification infrastructure into a clean, event-sourced system. Every observation—every bot test, every human verification, every real checkout—is an immutable event in a stream. Truth is not a value stored in a database. Truth is a computation derived from the full history of evidence, with complete provenance: who discovered this code, how, when, and with what evidence. This means we can trace exactly why any verdict was issued, replay the evidence chain, and correct errors without data loss.

The new architecture is being validated in shadow mode alongside our existing system—ingesting the same data, computing the same scores, comparing results—before it takes over production traffic.

Agent-Native Endpoints. The Proof Packet is not just internal—it is our product for the agentic economy. When AI agents need verified commerce data, we become infrastructure. We are building APIs designed for machine consumption: structured verdicts with confidence scores, evidence chains, and applicability predictions.

The Confident No at Scale. In a world of AI-generated commerce noise, the ability to definitively say “There is nothing to find” becomes the scarcest resource.

We do not sell codes. We sell verdicts. And in an economy built on trust, verdicts are the only currency that matters.

Glossary

Term	Definition
ACT	Automated Code Testing—headless browser systems simulating real checkout sessions
BFT	Byzantine Fault Tolerance—consensus model assuming any source may lie
Health Score	Confidence measure combining freshness, outcome, and multi-source consensus
Confident No	Verified absence of codes—closure, not failure
Proof Packet	Structured evidence bundle documenting the adjudication chain
Sensing Event	An immutable, raw observation from a code application—evidence, not conclusion
Applicability	Probability a code applies to a specific cart context
Fleet Signal	Real-time telemetry from browser extension users during actual checkouts
Constitutional Model	Structured, machine-readable conditions governing a promotion’s validity
Forensic Anchors	Six traceability dimensions (actor, asset, target, agent, build, bridge) linking any observation to its verdict