THANAAYA TECHNOLOGIES

RESEARCH NOTE

The Matcher, Not the Verifier

Three rounds of work went into loosening a filter that was working correctly. One change in instrumentation showed where the defect actually was.

ABOUT THANAAYA TECHNOLOGIES

Thanaaya Technologies is a software company licensed by the Dubai Department of Economy and Tourism, building since 2021. We design, build and operate production systems across financial software, distributed ledger, gaming, commerce, media and quantitative research — and we run several of them as our own products rather than only delivering them for clients.

We take a small number of engagements at a time, and the engineer who designed a system is the one who gets called when it misbehaves. This paper comes out of our internal research programme, which exists to find out cheaply whether an idea is real. We publish the negative results as well as the positive ones: an undocumented failure gets rediscovered, and a threshold nobody wrote down cannot be checked.

2021	24	80+	9
BUILDING SINCE	PRODUCTION DOMAINS	SERVICES IN PRODUCTION	INDUSTRIES SERVED

Subject: defect attribution in a multi-stage model pipeline
Programme: internal quantitative research — paper trading only, no capital at any stage
Status: diagnostic finding carried forward to a successor design
Published: September 2026 · **Licence:** Dubai Department of Economy and Tourism 964356 · **Unified licence:** BL1484

PART ONE

In plain English

No mathematics in this section. We spent three rounds of work fixing the wrong component, and one change in instrumentation showed us that.

1. The setup

We were building a system to trade news into **Polymarket** event contracts. It had two stages.

The first, the **matcher**, took a news headline and found the contract it might affect — a story about a central bank, say, pointed at a contract on interest rates. The second, the **verifier**, checked that connection with an independent model and threw it out if it did not hold up.

The matcher proposes; the verifier disposes. And the verifier was throwing out almost everything.

0.7%	1	151
of proposed trades the verifier accepted, in the final run	trade produced in five days of continuous running	rejections in the same period

2. The reasonable, wrong diagnosis

When a filter rejects 99% of what reaches it, the natural conclusion is that the filter is too strict. We reached that conclusion three separate times, and each time we adjusted the verifier — reworded its instructions, softened its criteria, changed what it was asked to confirm.

The acceptance rate moved every time: 2%, then 61%, then 23%, then 8.5%. That looked like tuning. Turn the dial, watch the number respond.

> WHY THAT READING WAS SEDUCTIVE
>
> The number *did* respond to what we changed. It is very hard to believe you are adjusting the wrong component when the thing you adjust visibly moves the outcome. What we were watching was a high-variance component swinging around — not a control surface responding to input.

3. The change that settled it

Instead of adjusting anything, we made the verifier record *why* it rejected each item, choosing from a fixed set of categories decided in advance. Not a free-text note — a code on every single rejection, with a requirement that every rejection carry one.

In the final run there were 151 rejections. All 151 were categorised, none were malformed, and none landed in an "other" bucket. So the picture was complete rather than a sample of whatever happened to be legible.

4. What the reasons said

Two thirds of the rejections were not the verifier being fussy. They were the verifier correctly noticing that **the connection the matcher proposed did not exist.**

WHY IT WAS REJECTED	COUNT	WHOSE FAULT
Headline is only *indirectly* related — about the topic, but not about the thing the contract actually settles on	**60**	**matcher**
Routine commentary — an article about the subject reporting no new fact	**39**	**matcher**
Not relevant to how the contract resolves	33	matcher
Topic overlap only	17	matcher

> THE FINDING
>
> The verifier was doing its job correctly. **The matcher was manufacturing plausible-looking connections that did not survive inspection** — and when the pool of available contracts was narrow, it manufactured harder rather than proposing nothing.

Three rounds of work had gone into making the verifier more permissive. Had that succeeded, it would have made things *worse* — letting through exactly the invented matches the verifier was correctly catching.

5. Why the matcher behaved that way

It was asked to find the *best* contract for each headline. When lots of contracts are available, "best" is a real choice. When few are — because filters upstream have removed most of the inventory — "best" quietly degrades into "least bad", and it returns something regardless.

This is visible in the data: when the system was starved, rejections clustered onto a handful of the same contracts, proposed again and again for headlines that had nothing to do with them.

Each round of tightening narrowed the inventory further, which worsened the forced matching, which lowered the acceptance rate, which was then read as evidence that the verifier needed loosening. It was a loop, and the reason codes broke it.

6. The general lesson

A stage that rejects things tells you a rate. A stage that rejects things *with a reason* tells you which stage is broken.

The costs are wildly asymmetric. Recording the reasons is nearly free and can be added afterwards. Not recording them cost three rounds of work aimed at the wrong component, plus the collapse in throughput that followed.

We now require any filtering stage to attribute its rejections, with a fixed vocabulary and a rule that everything must be attributable. If a rejection cannot be categorised, that is itself a finding about the filter.

PART TWO

The technical record

Throughput by run, verifier pass-rate variance, and the rejection attribution that relocated the defect.

1. Throughput across seven runs

Trades produced per hour and verifier pass rate, by recording run.

RUN	WINDOW	RUNTIME	TRADES	RATE/H	VERIFIER PASS	WHAT IT ESTABLISHED
0	08-04	—	—	—	—	Plumbing; deduplication defect
1	08-07→08-10	—	100	—	2%	24% of markets expired; 2 markets took 49% of the book
2	08-10→08-13	72.4 h	24	0.33	61%	Clean routing baseline; 79% of book in 2 correlated markets
3	08-13→08-15	43.7 h	10	0.23	18%	Breadth fix worked; headline return proved an artefact
4	08-15→08-16	36.1 h	3	0.083	9.7%	Reconciliation exact; starvation traced to a stale universe
5	08-16→08-21	106.1 h	17	0.160	8.5%	Fixed throughput, not quality; produced the go-live refusal
6	**08-21→08-26**	121.7 h	1	0.008	0.7%	**Maturity + moneyness gates; throughput collapsed 20×**

Run 6 is the terminal state of the loop described in Part One: each tightening narrowed inventory, force-routing intensified, and the pass rate fell. At 0.008 trades per hour the system could not generate the sample required to falsify its own hypothesis, which is a distinct failure from generating a sample that disconfirms it.

2. Verifier pass rate is a high-variance component

Across successive revisions of the verification prompt the pass rate ran **2% → 61% → 23% → 8.5%**. That range, on a component that was never the binding constraint, is the reason it absorbed three rounds of attention.

Two operational rules follow. First, a verification prompt must never be revised in the same change as anything else, because its variance will be attributed to whatever else moved. Second, a pass rate is not a quality metric in isolation — it is a ratio whose numerator and denominator are both under active development.

3. Rejection attribution

Run 6: 151 rejections, 151 attributed, 0 malformed, 0 unspecified.

REASON CODE	N	SHARE	LOCATES THE DEFECT IN
`indirect-effect`	60	40%	**matcher**
`routine-commentary`	39	26%	**matcher**
`not-resolution-relevant`	33	22%	matcher / market selection
`topic-only`	17	11%	matcher

66% of rejections are the matcher manufacturing plausible matches, not the verifier applying an excessive standard. The two leading categories describe headlines that are topically adjacent to a market but carry no bearing on its resolution criteria, and headlines that report no new fact at all.

The completeness of the attribution matters as much as the distribution. With 151 of 151 rejections categorised and no unspecified bucket, the conclusion does not depend on assuming the uncategorised remainder resembles the categorised part.

4. Force-routing under a narrow universe

When the available market inventory is small, the matcher does not decline — it returns the closest available candidate. Empirically this shows as rejection clustering: under starvation, rejections concentrate on a handful of the same tokens, repeatedly proposed for unrelated headlines.

The corollary is a design constraint rather than a tuning parameter: **narrowing selection gates without simultaneously widening inventory recreates this every time.** Runs 4, 5 and 6 each reproduced it.

5. What this hands to a successor

FINDING	CONSEQUENCE FOR A REBUILD
Two thirds of rejections originate in the matcher	**Matcher design is the primary work; verifier tuning is not**
The matcher force-routes under narrow inventory	It must be permitted to return nothing, and inventory width must be a monitored quantity
Verifier pass rate swings 2%–61% on prompt revision	Isolate prompt changes; never co-deploy them with pipeline changes
Attribution was close to free and relocated the defect immediately	Every filtering stage attributes its rejections from the outset

The return evidence that closed the programme is a separate matter and is recorded in the companion paper *Markets That Do Not Move*. This paper is about the diagnostic failure rather than the economic one, and it is the more transferable of the two: the economics are specific to one market structure, whereas misattributing a defect to the stage that reports it is a general hazard of any multi-stage pipeline with a model in it.

6. Disposition

Version 1 was stopped, frozen and archived. Live trading was requested and refused, and the refusal stands. No capital was committed at any stage. Three defects were deliberately left unfixed and recorded as not to be inherited: there is no exit path anywhere in the codebase, no scheduled-release blackout, and a verifier that will confirm both directions of the same headline.
