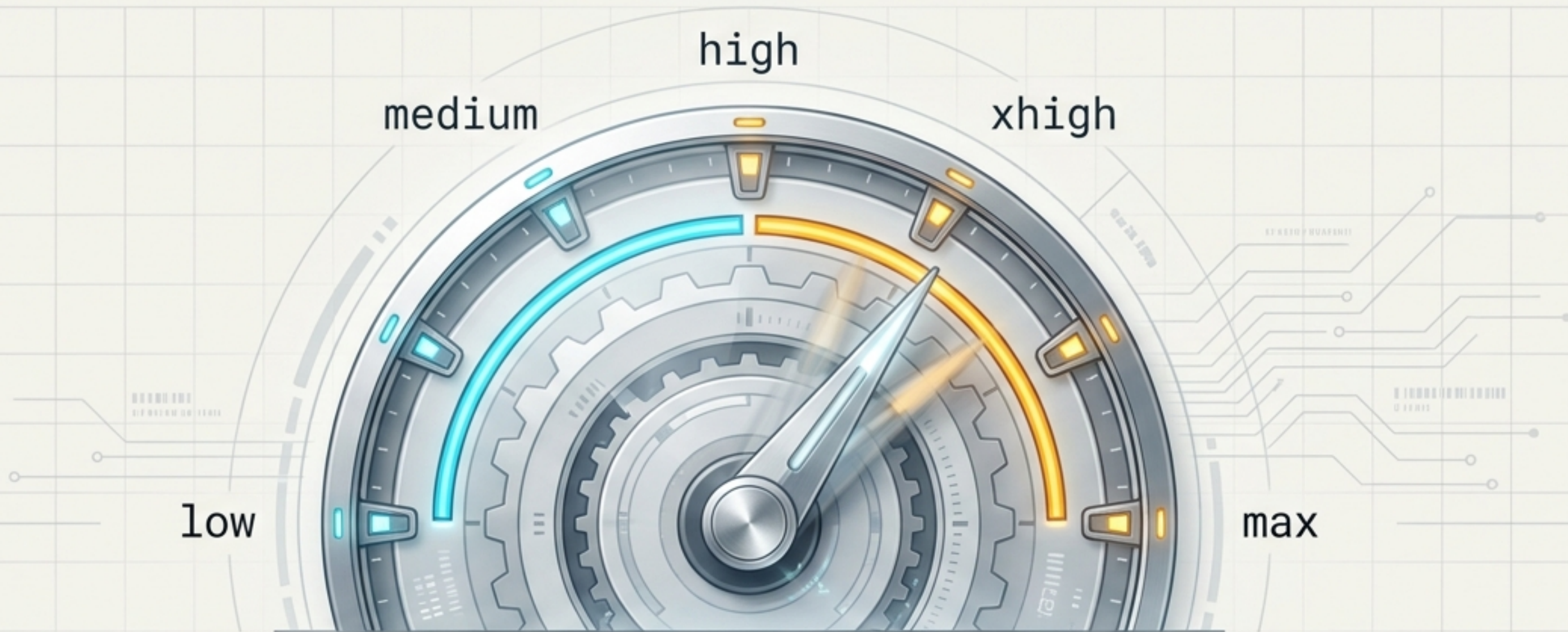




Opus 5.5: Medium vs. High Effort

An empirical evaluation of reasoning effort on real-world commits, autonomous completions, and invisible bugs.

// Target: Evaluate Time, Tokens, and Output Quality on 88 paired executions.

Executive Briefing & Bottom-Line Up Front


100% Tie

- **The Illusion:** Completion rates on automated test suites are identical. Medium and High both passed 100% of hidden tests.

~1.3x 
~1.3x Tax

- **The Cost:** High effort demands roughly 33% more time and 35% more tokens.

Invisible Bugs



-  **The Reality:** Blind judging reveals High effort acts as an insurance policy against edge cases that automated tests miss but break production.

Recommendation: Map effort to review capability. Default to High for zero-review autonomous agents; default to Medium for pair-programming.

Real Repositories. Real Commits.

Stepping away from SWE-bench to replay history on active, personal codebases.

11

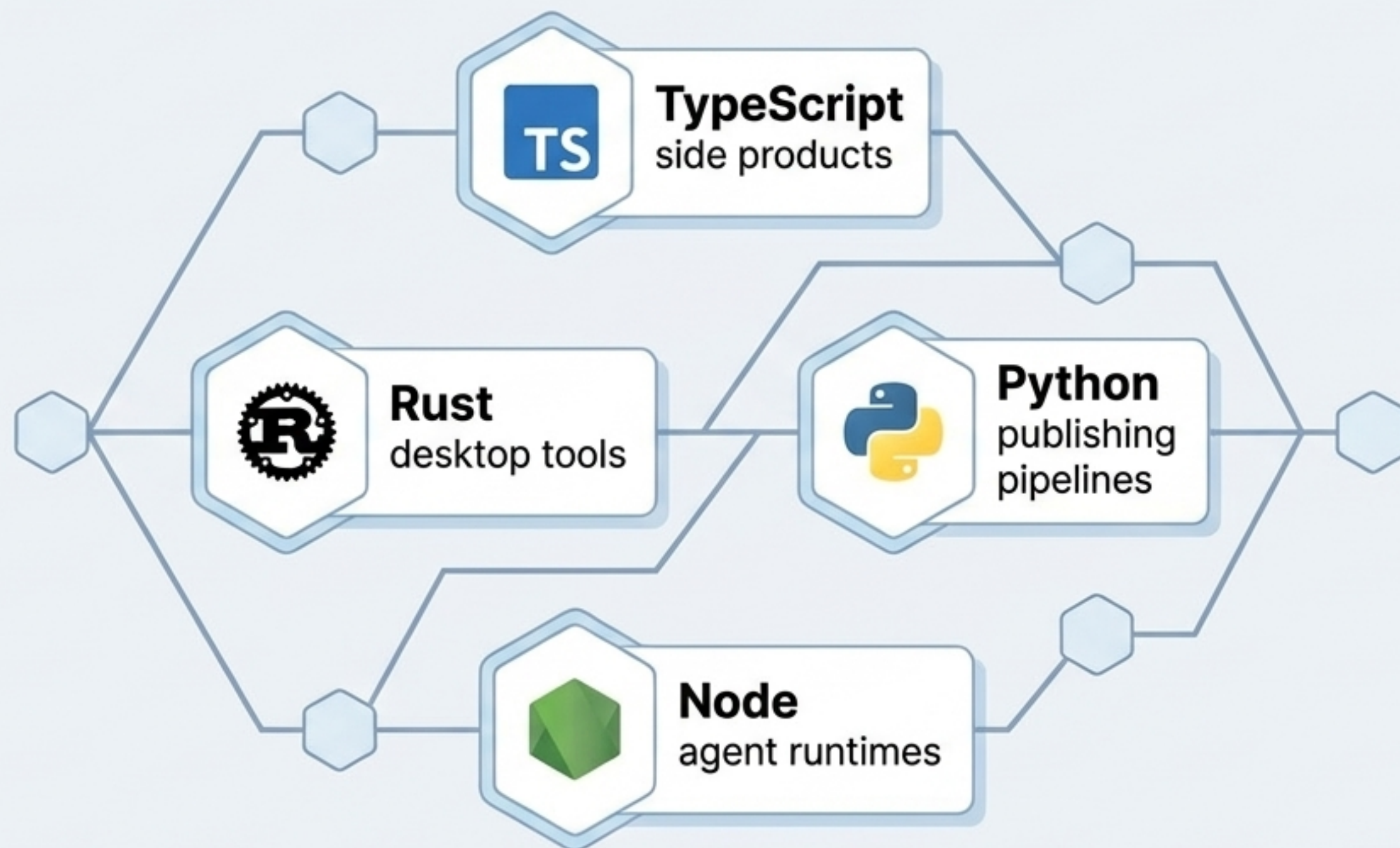
Repositories

22

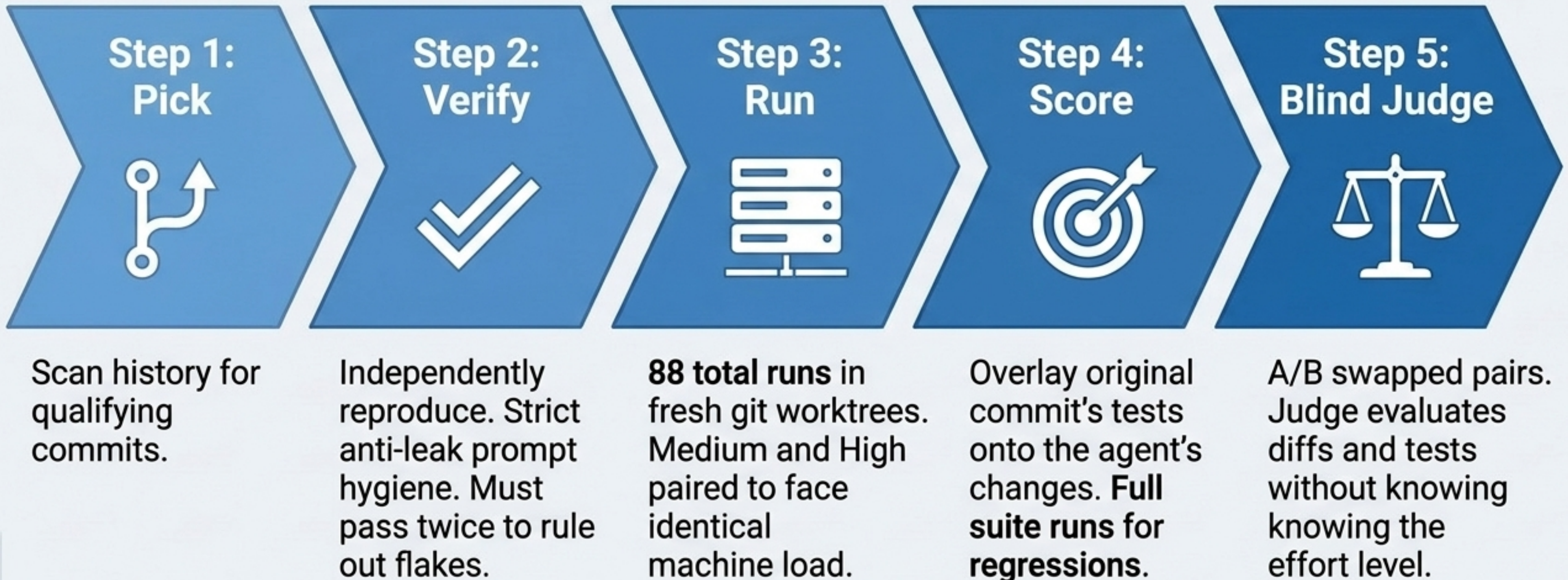
Eligible
Commits
(Touched source +
tests, feature/bug fix)

88

Total Agent
Runs



The Evaluation Pipeline



Note: 39 agent runs dropped due to a revoked local login token mid-test. Remaining dataset: 44 perfectly complete, equal-condition pairs.

The Quantitative Illusion: A Statistical Tie

If you only look at test suites, you are wasting tokens.

100% Pass Rate:

All 44 Medium runs and all 44 High runs passed the hidden tests.

44 / 44

0 Regressions Found

Zero Regressions:

Neither effort level introduced new failures into the full test suites.

Takeaway: Tests alone cannot justify the upgrade to High effort.
The difference lies in what the tests cannot see.

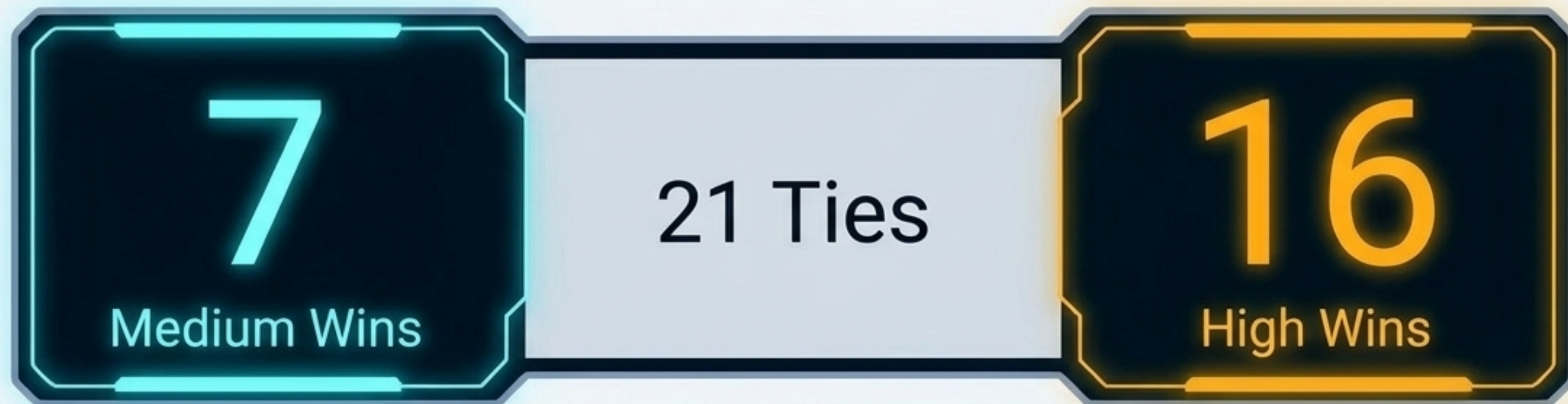
The Cost of High Effort

Metric	Medium (Baseline)	High (Multiplier)
Hidden tests passed	44/44	44/44
Median wall time	140s	234s
Wall time (paired geo-mean)	1.0x 	1.33x  Slower in 39 of 44 pairs
Output tokens (paired geo-mean)	1.0x 	1.35x  More in 43 of 44 pairs
Token-weighted cost	1.0x 	~1.29x 

High effort operates with a consistent ~30% tax across time, tokens, and money. What does that tax buy?

The Qualitative Reveal: If Tests Pass, Why Did High Win?

88 double-blind judgments swapping A/B positions reveal a clear preference.



Medium's Wins (7)

Mostly instances where High changed things nobody asked for, or forgot docs. Only 1 real bug (High stripped production client-IP headers).

High's Wins (16)

Dominated by the resolution of real, functional bugs that the automated tests simply couldn't catch.

Anatomy of an Invisible Bug

Examples of all-green tests resulting in broken production code under Medium effort.

The Illusion of Tests

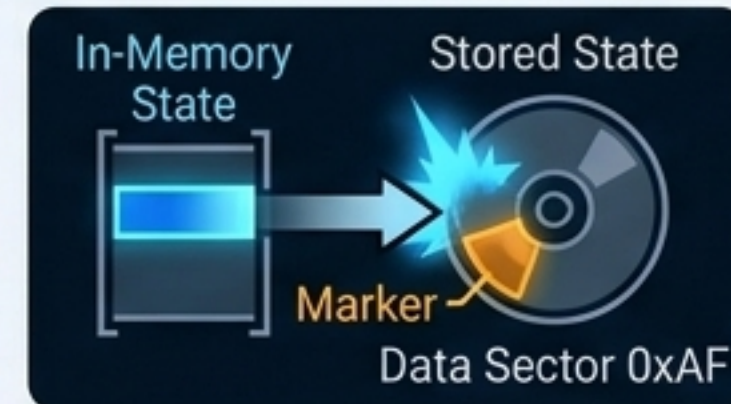
```
✓ PASS Test Suite A: 120/120
✓ PASS Test Suite B: 250/250
✓ PASS Test Suite C: 85/85
✓ PASS Test Suite D: 150/150
✓ PASS Test Suite E: 200/200
✓ PASS Test Suite F: 100/100
```

100% Coverage Achieved.
All Systems Green.

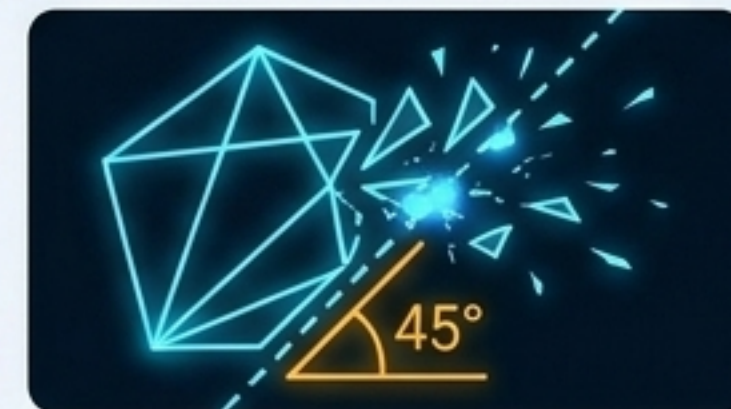
The Reality of Bugs



The Chart Overflow: Medium scaled bar heights against uncompressed series but drew compressed bars. 90-day charts overflowed past 100%. Rendering wasn't tested.



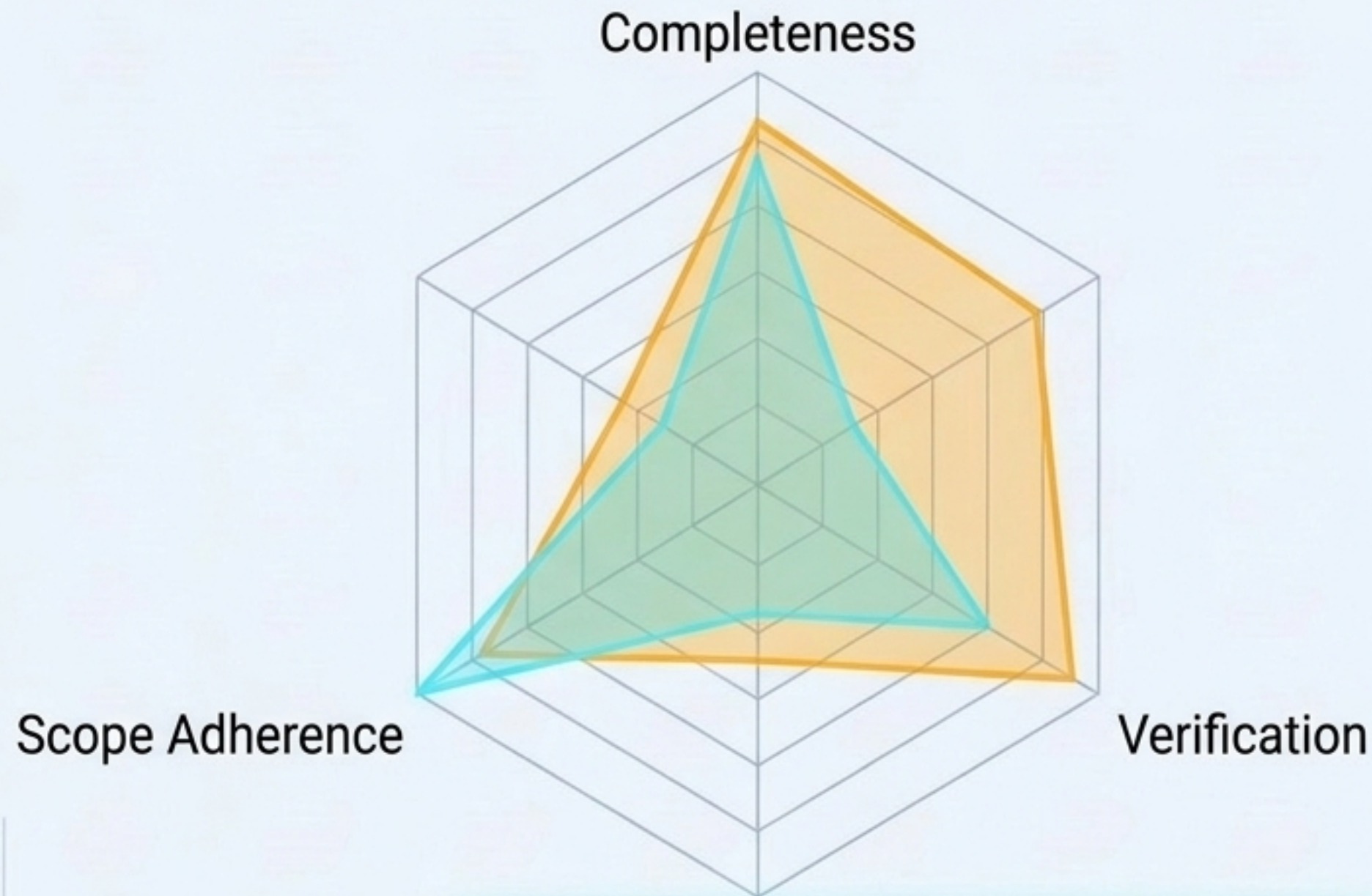
The Erased Marker: Medium updated stored state but not the in-memory copy. A later write-back erased its own marker.



The Geometric Rasterizer: High held tolerances at all angles. Medium's float/integer paths broke at small sizes and 45-degree angles.

Scope vs. Completeness

High thinks further ahead; Medium does exactly what it's told (and nothing more).



Completeness:

High (8.90) > Medium (8.72)

Verification:

High (8.89) > Medium (8.62)

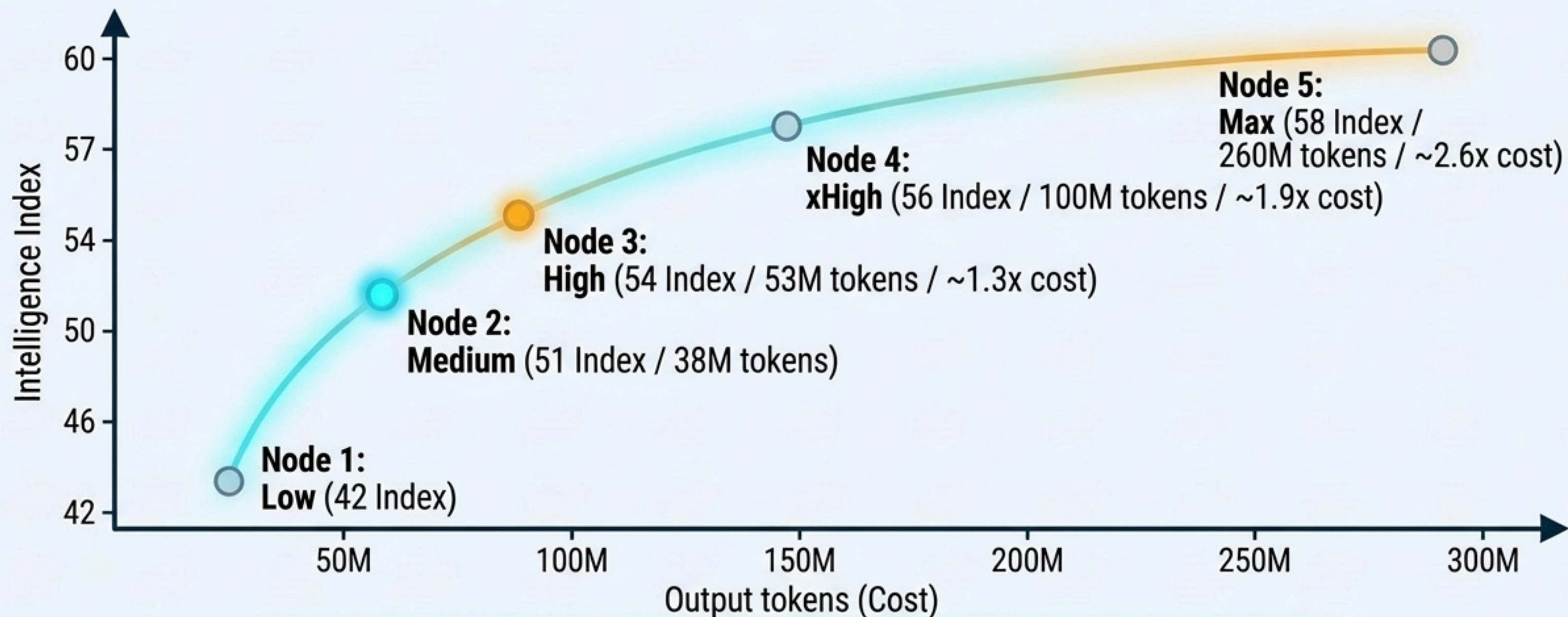
Staying in Scope:

Medium (8.76) > High (8.28)

Takeaway: High likes to do a little extra—adding tests and doc updates. Medium strictly stays in scope, avoiding unrequested changes.

External Validation: The Pareto Frontier

Cross-checking local results against the Artificial Analysis Intelligence Index.



Validation: Both datasets confirm: Medium is already strong, but High is on the exact edge of the Pareto frontier before severe diminishing returns set in.

Finding the Sweet Spot

Node 2
Medium: 51 Index
38M tokens

Step 1: Medium -> High
Buys 3 index points for only
~1.3x the token cost. This is
the most efficient step on
the curve.

Node 3
High: 54 Index
53M tokens / ~1.3x cost

Step 2: High -> xHigh
Buys only 2 index points for
nearly double the token
cost.

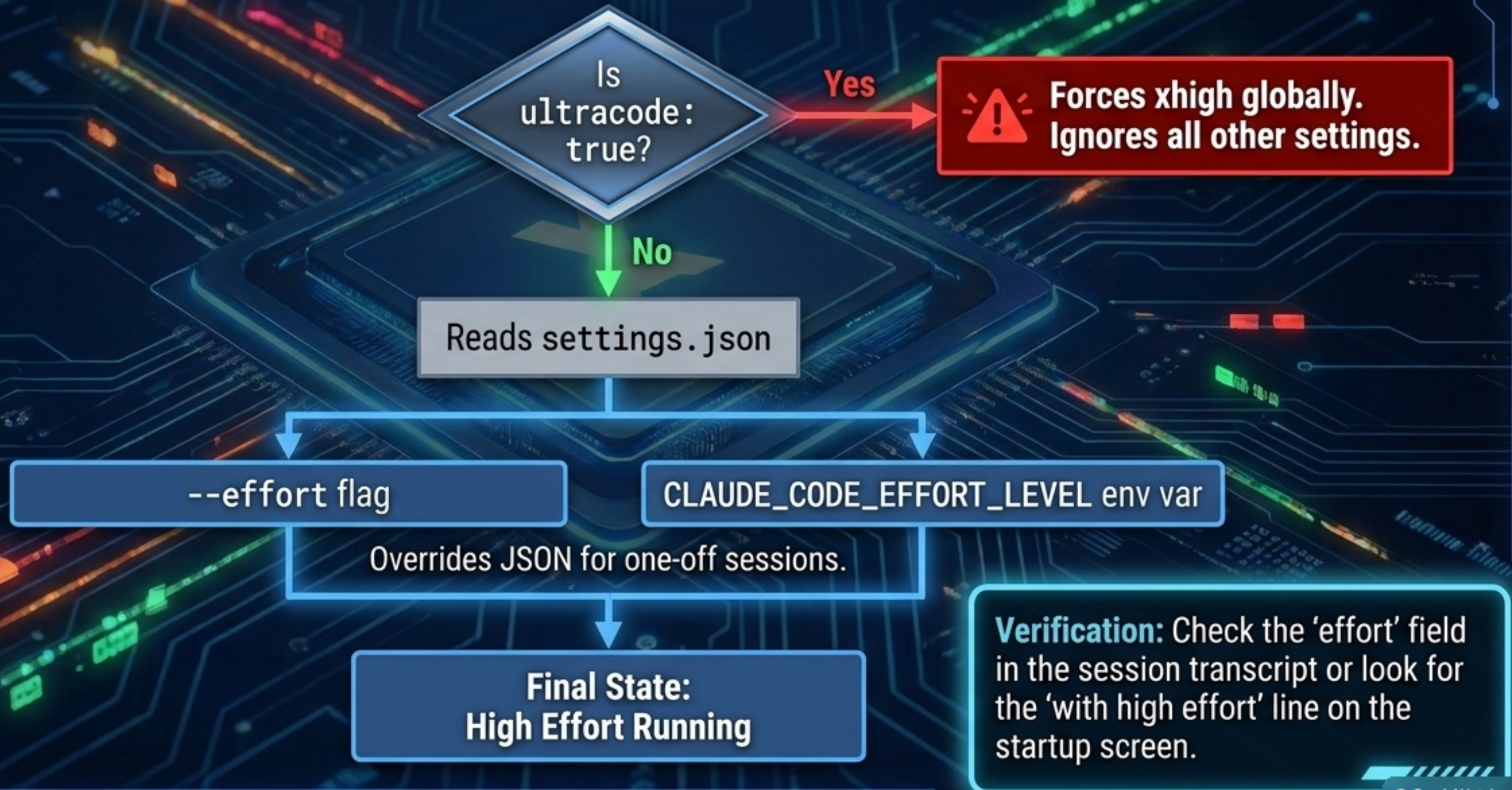
Node 4
xHigh: 56 Index
100M tokens / ~1.9x cost

Step 3: xHigh -> Max
Buys another 2 points for
2.6x the tokens.

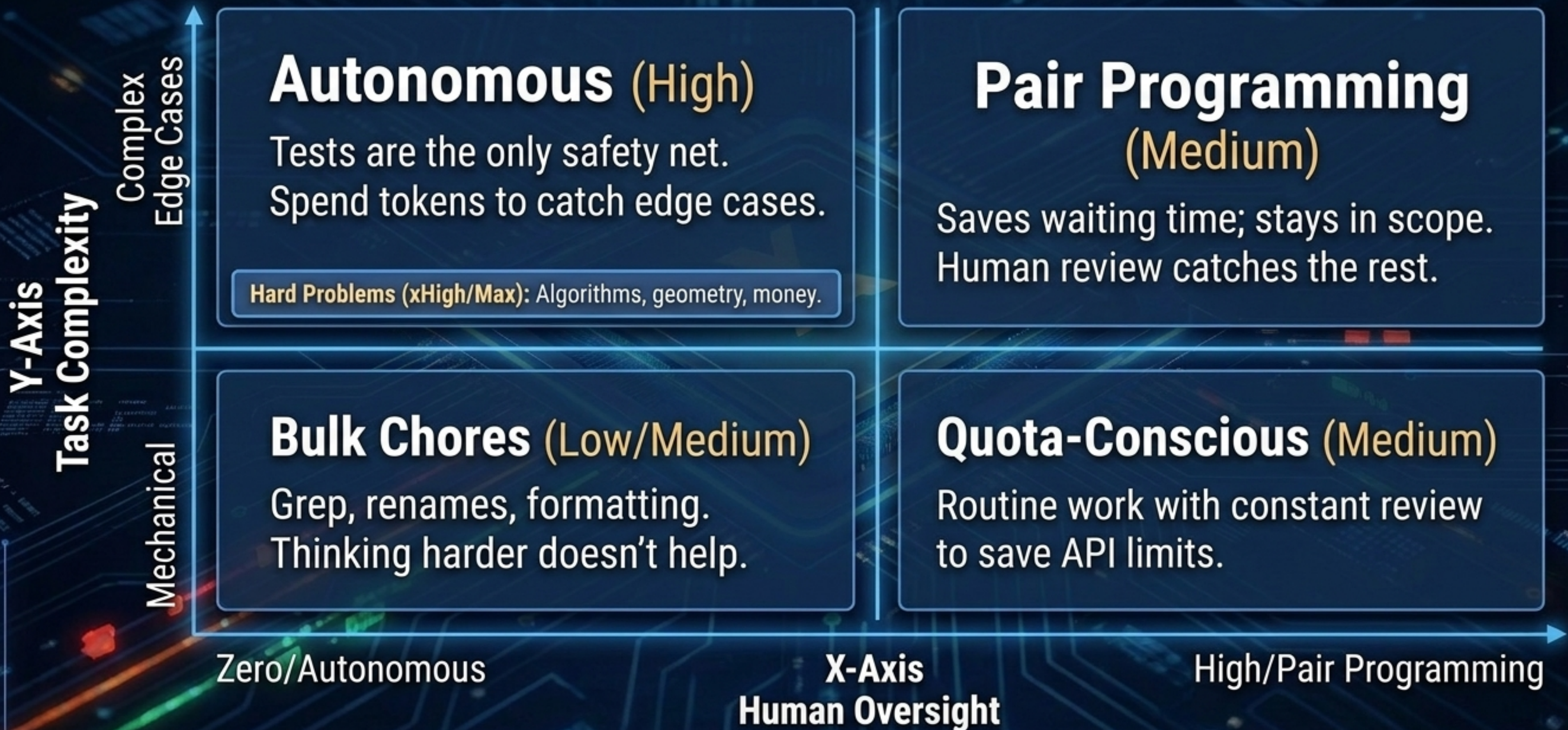
**Takeaway: The jump to High is the best value per token for ensuring output safety.
Beyond High, returns shrink exponentially.**

The Configuration Gotcha

How to ensure you are actually running High effort in Claude Code.



The User Persona Matrix: Which Effort For Which Task?



The Final Verdict

“Tests are just the safety net.
Model reasoning is the acrobat.
Size the effort to the stakes.”

- 
1. Don't pay for passing tests; pay for the edge cases the tests missed.
 2. Set Opus 5.5 to **High** as your default for autonomous runs.
 3. Verify your ultracode settings immediately.