AI Deliberation & Cursor

// disagree with me

Cursor's agent has your repository. Quorum has a panel — frontier models from different labs, arguing, returning one answer and where they split. A median of 28.5 seconds, for the decisions where being fast is worth less than being right.

What is in here

Sections 03 to 05 are the argument, and 04 decides whether this belongs in your editor at all. 06 and 07 are what you install and say. 08 to 10 are the work itself. 11 to 14 are the receipt and the wiring.

EXHIBITS

03

What Quorum is, in one page.

Quorum is a deliberation platform. One question goes to several frontier models from different labs at once. They answer independently, critique each other, and are judged — and what comes back includes **where they disagreed**.

If you have only ever worked with one assistant, the distinction that matters is this: asking the same model again gives you the same reasoning, more politely. That is why re-prompting a stuck agent so rarely helps. Independent models fail in different places. That is the whole mechanism.

A SEAT

One frontier model on the panel. Three seats is the standard panel — three independent lines of reasoning. Quorum represents them as shapes rather than naming vendors.

A MODE

The configuration of a deliberation — which engines sit, how many rounds, how deep. Not a knob on a single model: a repeatable decision setup you select.

A RECEIPT

The record of a completed deliberation: which model sat in each seat, judge scores, difficulty, cost, latency. §11.

WHERE IT RUNS

Quorum's own surfaces, an OpenAI-compatible REST API, and a hosted MCP server — which is how it reaches Cursor. Nothing installs locally.

THE CLAIM, STATED NARROWLY

Quorum does not claim a single model is bad. It claims that a single model, however capable, **cannot be its own second opinion** — it has one way of being wrong and applies it consistently across every retry in the thread. Panels exist for the questions where that matters.

WHO IS ACTUALLY IN THE ROOM

Cursor's agent is **not one of the panel**. It stays where it is, holding your repository and your rules, and calls in independent models — none of them the one you were just talking to. The agent has your codebase; Quorum has whatever you hand it. That is what shapes the pairing, and why §06 insists on real code.

AND WHERE IT DOES NOT APPLY

Most of what you do in an editor is execution, not judgement. Quorum is deliberately wrong for completions, renames, imports, syntax, test scaffolding and a regex you could write yourself — and its own `estimate` call says so before you spend anything (§14).

04

// disagree with me

The editor is fast. A few decisions should not be.

Tab finishes the line before you have finished thinking it. An inline edit rewrites the block in place. The agent changes six files and hands you a button. The whole instrument is tuned so the machine is never the thing you are waiting for, and that tuning is most of why you use it.

A panel is not fast and will not become fast. Convening one takes a median of 28.5 seconds, 88.2 at the ninetieth percentile. No version of this disappears into the flow.

The honest framing is not that deliberation makes you faster. A few decisions a week are not rhythm problems at all — you will be living inside them in six months — and there, **being fast is worth less than being right**.

MEASURED ON PRODUCTION, 2026-08-20

estimate	~0.9 s	10 prompts
deliberate p50	28.5 s	n=3,845
deliberate p90	88.2 s	n=3,845

A 28-second call costs more rhythm than a wrong rename costs time. Most of the discipline in this document is telling those two apart in the moment.

AND NEVER IN A LOOP

Anything a single undo puts back does not go to a panel — nor does anything inside a tight iteration loop, where the waiting compounds and the stakes do not.

CHAT — FEAT/ACTOR-ID AGENT

YOU

Rename `Session.user_id` to `actor_id` across the service.

CURSOR AGENT

41 references in 6 files. Before I apply this — the rule says a change whose reach I cannot see goes to the panel, and `billing/webhook.ts` parses a payload I do not control. `quorum · deliberate — running`

CURSOR AGENT · 28.5 S LATER

All three seats agree the in-repo rename is correct. They named **three different boundaries** and split on which one bites first. I have not reconciled the split.

Exhibit 01 — The turn where the agent stops and convenes. Every other turn in the session is answered with no panel and no wait. Illustrative, not a transcript.

05

The five moments it escalates on.

These are not one language or one stack. They recur in any codebase that has been alive long enough to have history in it — a service, a monorepo, a library other teams import.

01	**A change whose blast radius is unclear**	The agent lists the call sites it found and offers to apply the change to all of them. The list is accurate and the question is what is *not* on it — the queue consumer, the external payload, the row written by the previous deploy. **§08 works this one end to end.**
02	**A fix that keeps not fixing it**	The same test fails for the third time this week. Each proposed patch is confident, plausible and green by lunchtime. The thing worth arguing about is the diagnosis, not the patch. **§09.**
03	**A commitment with a migration behind it**	A dependency, an API shape, a schema, a queue's delivery semantics — anything the next three years of code will be written against. Differently-trained models genuinely disagree here, and a confidently-wrong single answer is expensive. **§10.**
04	**An answer you already have**	You have the design. Hand the panel the approach you are leaning toward and ask what breaks. The useful output here is usually the disagreement rather than the synthesis — a panel given a position to attack argues much harder than one asked an open question.
05	**Somebody else's code**	Reviewing a large pull request, reading a vendor SDK, taking over a service whose author has left. You have no intuition for what is load-bearing, and one model's summary reads as authoritative whether or not it is right.

THE COMMON SHAPE

In all five the code is not the hard part. The work is done, or nearly done, and what remains is a **judgement** about it. That is the line your standing instruction draws in the next section.

AND WHAT IT SHOULD NEVER ESCALATE

Completions, renames, imports, formatting, syntax, boilerplate, a regex, a test stub. Quorum's own classifier agrees: it returns `likely_hurts` when the answer is a verbatim artifact a synthesis pass would only rewrite — which describes a great deal of what an editor produces.

06

Teach it once.

Give the agent a standing rule for when to convene the panel, and the decision stops being yours to remember. This is the whole setup.

Put it where Cursor keeps rules for the repository, checked in beside the code — the right place for it, because when to escalate is a property of the codebase and not of your laptop. If your team keeps no rules file, paste it at the top of the chat you are about to work in.

STANDING INSTRUCTION — PASTE AS WRITTEN, THEN EDIT THE LAST LINE

```
You have a Quorum connector. It convenes a panel of frontier models
from different labs and reports where they disagree.

Use it on judgement, not on work you can do yourself.

Convene the panel when:
  - you are about to apply a change and cannot see its full reach
    — name the files you think it touches and the boundary you
    cannot see past;
  - you have proposed a fix for the same symptom twice and it has
    come back;
  - I am choosing a dependency, an API shape, a schema or a
    delivery guarantee later code is written against, or there is
    a migration behind the decision;
  - I ask you what breaks in an approach I already have.

When you do:
  - put the real code in the question — the diff, the failing
    assertion, the actual paths — not a description of them;
  - say what you are proposing and ask the panel to attack it;
  - report where the seats disagreed and do not reconcile it for me.

Never convene it inside a loop: completions, renames, imports,
syntax, formatting, boilerplate. Do those yourself, immediately.

Escalate without asking me first.   ← or "ask me first"
```

BEST PRACTICE

Install the rule and let it escalate on its own. The value of this setup is that it catches the moment you did not notice was a judgement call — and if you have to remember to invoke it, you will remember exactly when you least need it. Approving each escalation is right for your first week and wrong after that.

THE ONE LINE THAT MATTERS MOST

"Put the real code in the question." An agent left to summarise will write *a refactor that may have wide impact*. What you want sent is *renaming Session.user_id to actor_id; 41 references in 6 files; one of them parses an external webhook payload.* Same escalation, an entirely different argument comes back.

07

Steering it in the moment.

The standing rule handles the moments you did not spot. These are for the moments you did — when the diff is on screen, your hand is over Accept, and you already know the question is worth an argument.

"Before I accept this — put the diff to the panel and tell me what it touches that you have not listed."

Attacks the blast radius, not the code

"You have proposed a fix for this twice. Get a second opinion on the diagnosis, not the patch."

Moves the argument up one level

"We will be writing against this interface for years. Which risk am I taking with each option?"

Reframes a tie as a risk choice

"There is a migration behind this. Convene the panel before I write the down script."

Catches the one-way door while it is still two

"Here is the approach I am going with. Ask the panel what breaks."

A position to attack, on purpose

"Where did the seats disagree? Do not reconcile it — show me the split."

The follow-up worth making a habit

"Send the failing assertion and both call paths verbatim, not a summary of them."

When precision is the whole question

"Do not summarise the panel for me. Put each seat's position in the pull request."

Keeps the argument attached to the change

BEST PRACTICE

Use both. The standing rule is the safety net; these are the times you reach for it deliberately. If you only ever do one, do the standing rule — the escalations you would have typed by hand are the ones you already knew were hard.

WHAT MAKES AN ESCALATION LAND

Paste the code. Say what decision hangs on it — ship today, migrate next quarter, publish as a public interface. Say what you are leaning toward. And ask for the split explicitly; a tidy synthesis is the least useful thing a panel can hand you, and the easiest one for it to produce.

08

BLAST RADIUS → ROOT CAUSE → LOAD-BEARING

A change whose reach you cannot see.

A rename that looked local. The agent found every call site, updated them all, and the suite is green. The button says Accept.

WHAT THE AGENT IS ABOUT TO APPLY

FILE	+	−	
src/auth/session.ts	12	9	
src/api/middleware.ts	4	4	
src/billing/webhook.ts	6	6	EXTERNAL PAYLOAD
src/jobs/reconcile.ts	2	2	
tests/session.spec.ts	18	11	
docs/architecture.md	3	3	

ILLUSTRATIVE · INVENTED PATHS AND COUNTS

WHAT THE AGENT SAYS

"All 41 references to `user_id` updated across 6 files. The test suite passes."

Both true. Neither is the question. The suite covers what the codebase already does. It does not cover the two places where something outside the repository is holding the old name.

WHAT THE PANEL RETURNS

▲ **Complete inside the repository, wrong at the edge.** `billing/webhook.ts` parses a body sent by a provider that still sends `user_id`. The type was renamed; the wire format was not.

■ **Agrees the edge is the risk, disagrees about which edge.** The webhook is versioned and will fail loudly on the first bad payload. The quiet failure is `jobs/reconcile.ts`, which reads rows written before the deploy and will simply find nothing.

● **Both are treating this as a rename.** It is a schema change with a rename on top. During rollout two versions write concurrently, and nothing in this diff addresses the overlap.

The seats did not vote. They named three different boundaries — and the third re-classified the change, from a rename you can revert into a migration you cannot.

WHAT IT COST YOU

One pause, on the branch you told it to pause on. p50 28.5 s, p90 88.2 s, measured 2026-08-20 across 3,845 deliberations.

Exhibit 02 — A multi-file change, and the three boundaries the seats named. Positions are illustrative of the pattern, not a transcript of a specific call.

09

BLAST RADIUS → **ROOT CAUSE** → LOAD-BEARING

The fix that keeps coming back.

Third time this week. Each patch is confident, plausible and green by lunchtime; by Thursday the same assertion is red again.

THE FAILURE

```
$ npm test -- checkout
FAIL  tests/checkout.spec.ts
  applies discount before tax   failed 2 of 3 runs
  expected 92.40 to be close to 92.35
    at checkout.spec.ts:48
```

ILLUSTRATIVE · INVENTED OUTPUT

WHY IT RECURS

The agent is not being careless. It is being asked *make this test pass*, it has one reading of the symptom, and it applies that reading consistently — including on the retry where you told it to think again.

THE TWO PATCHES ALREADY APPLIED

src/checkout/total.ts · Tue

```
@@ -31,7 +31,7 @@
-  const t = round(sub * (1 + rate), 2);
+  const t = round(sub * (1 + rate) + 1e-9, 2);
```

tests/checkout.spec.ts · Thu

```
@@ -46,4 +46,4 @@
-  expect(total).toBeCloseTo(92.35);
+  expect(total).toBeCloseTo(92.35, 1);
```

ILLUSTRATIVE · INVENTED DIFFS. THE SECOND ONE EDITS THE TEST, WHICH IS THE TELL

WHAT THE PANEL RETURNS WHEN YOU SEND IT THE DIAGNOSIS INSTEAD

▲ **Not floating point.** The discount is applied to the pre-tax subtotal on one path and to the taxed total on another. The two differ by exactly the tax on the discount, which at these inputs is 0.05.

■ **Agrees the paths differ, disagrees that it is this bug.** Two of three runs failing is order dependence, not arithmetic — a shared rounding mode is being mutated by whichever spec ran first.

● **Cannot be separated from what you sent.** Both explanations fit this output. Get the seed and the run order from a failing run before changing any code.

Two diagnoses and one refusal to guess. A single model picks one of these and sounds certain — which is precisely what the last three commits were.

ESCALATE THE DIAGNOSIS

Not the patch, and not the file. The panel is arguing about a cause; give it the failure, both paths, and what you already tried.

Exhibit 03 — A failing test, and the two fixes that did not fix it. Seat positions are illustrative of the pattern, not a transcript of a specific call.

10

BLAST RADIUS → ROOT CAUSE → **LOAD-BEARING**

The choice you will live inside.

Two lines in a manifest and one new file. Everything written in this service for the next three years will be written against them.

THE COMMITMENT, AS IT APPEARS IN THE DIFF

```
package.json
@@ -18,6 +18,7 @@
   "dependencies": {
+    "@vendor/queue": "^3.1.0",
     "fastify": "^4.26.0",

migrations/
+   0042_jobs_outbox.sql
```

ILLUSTRATIVE · INVENTED PACKAGE AND MIGRATION

WHAT YOU ASKED

"We are going with this queue. Put the choice to the panel and tell me what breaks."

A position to attack, not an open question. The useful output here is the disagreement, not the synthesis.

WHERE THE PANEL SPLIT

▲ **The library is not the commitment; the delivery guarantee is.** At-least-once means every consumer written from here on has to be idempotent, and nothing in this diff says so anywhere a future author would read it.

■ **Agrees on the semantics, disagrees on the conclusion.** Taking that constraint deliberately is better than the alternative, which pushes the same duplicate handling into application code where nobody can see it.

● **Split the decision.** The dependency is reversible in an afternoon; `0042_jobs_outbox.sql` is not. Those are two decisions and only one of them is a one-way door.

The panel did not pick a library. It moved the decision — from *which package* to *which of these two lines is the one you cannot take back*. That is the answer you could not have got from agreement.

WHY THIS ONE, MOST OF ALL

Decisions you cannot cheaply reverse are where differently-trained models genuinely disagree — and where a confidently-wrong single answer is expensive for years rather than for an afternoon.

Exhibit 04 — Two lines you can revert, and one you cannot. Positions are illustrative of the pattern, not a transcript of a specific call.

11

The receipt, and what you can do with it.

Every deliberation leaves a record. It is the difference between “the AI said” in a pull request comment and a decision the next person can actually audit.

WHAT GET_RECEIPT RETURNS

request_id	rq_8f2c41a9
mode	quorum-standard
difficulty	0.78
seat 1	judge score, model
seat 2	judge score, model
seat 3	judge score, model
cost	billed, per call
latency	31.2 s

SHAPE OF THE RESPONSE · ILLUSTRATIVE VALUES

The judge scores are the part people underuse. They tell you whether the seats **actually disagreed** or converged immediately — which is how you learn that a question did not need a panel at all.

Quorum names seats by shape rather than by vendor on anything shareable, so a receipt can go into a public repository without carrying somebody else’s trademark.

IT EXPORTS

From Quorum, a completed deliberation renders as a full transcript PDF, a carousel, a card or a short video — the seats, their positions and the split, laid out.

IT TRAVELS

That artefact is what goes in the pull request description, the decision record, the ticket. The argument travels with the commit instead of evaporating in a chat pane you will close.

IT TUNES

Receipts that keep showing immediate convergence mean your rule is firing too widely — you are waiting 28.5 seconds for agreement. §14.

WHERE THIS IS GOING, STATED AS DIRECTION AND NOT AS A FEATURE

A receipt records one deliberation. What it does not do is attach itself to the commit that came out of it. The obvious next thing — a decision record that carries the disagreement, linked to the change it produced, retrievable a year later when somebody asks why the schema looks like this — does not exist today. There is no store, no linkage to your version control and no retrieval. It is named here as the direction the receipt makes possible, not as something you can switch on.

Exhibit 05 — A receipt.

12 Connect it.

Cursor speaks remote MCP. Quorum is a hosted MCP server, so there is no package to install and nothing runs on your machine — you add a URL and a key to one JSON file and the tools appear to the agent.

Project or global. The file beside the code commits the connector with the repository, which is usually what you want on a team — the same escalation behaviour for everyone who clones it. The one in your home directory follows you across every project instead. Get a key from your organisation dashboard.

ADD THE SERVER

.cursor/mcp.json (project) · ~/.cursor/mcp.json (global)

```
{
  "mcpServers": {
    "quorum": {
      "url": "https://www.quorum.dog/mcp",
      "headers": { "Authorization": "Bearer ${env:QUORUM_API_KEY}" }
    }
  }
}
```

WHY THE KEY IS NOT IN THE FILE

Cursor interpolates `${env:...}` in both `url` and `headers`, so the key can live in your environment while the config lives in the repository. That is what makes checking this file in safe.

OTHER EDITORS, SAME URL

Cursor and Claude Code nest servers under `mcpServers`. VS Code uses `servers`. Same protocol, same URL, different wrapper key — and the wrong one fails silently rather than telling you why.

> ONE CURSOR BEHAVIOUR WORTH KNOWING — IF YOUR CONNECTION BREAKS AND THE OTHERS DO NOT
>
> Cursor has a reported issue where a server advertising OAuth discovery endpoints causes it to ignore the `headers` block entirely and attempt OAuth instead. Quorum deliberately does not advertise those endpoints, so bearer auth works today. If that ever changes — if Cursor stops connecting while Claude Code and VS Code keep working against the same URL and the same key — this is the first thing to check, and it is a Cursor-side behaviour rather than a key problem.

13

What appears in the pane.

Six tools appear. Four are the working set, and the one that convenes the panel is the one that costs — though for the decisions in §08 to §10 its price is not what you are weighing. Ceilings belong in §14, where nobody is at the keyboard.

TOOL	COST	WHAT IT DOES
`estimate`	Free	Classifies and prices a question without running it. Returns `deliberation_value` — the escalation gate. §14.
`list_modes`	Free	Which panels your key may convene, and what they cost.
`deliberate`	Paid	Convenes the panel. The one that costs, and the one you wait for.
`get_receipt`	Free	For a finished call: which model sat in each seat, judge scores, cost and latency.
`run_test`	—	Mode testing. Added to the hosted server on 2026-08-22 — several published guides still describe it as local-only.
`certify_mode`	—	Mode certification, also hosted since 2026-08-22. These last two drive mode tuning rather than day-to-day work; call `list_modes` for what your key is priced at.

CHECK IT WORKS, IN ONE CALL

Ask the agent to call `list_modes`. It is free, it requires a valid key, and it returns the panels you are actually entitled to — so one real answer proves the URL, the key and the wiring together. A tool list — like the curl below — proves only that something answered.

TWO THINGS THAT LOOK LIKE FAULTS AND ARE NOT

A **GET** to that URL returns 405 — the server is Streamable HTTP, POST only, and that is the correct response. And the connection is stateless, so there is no session to keep warm.

BEFORE INVOLVING THE EDITOR AT ALL

```
curl -sS -X POST https://www.quorum.dog/mcp \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer qk_..." \
  -d '{"jsonrpc":"2.0","id":1,"method":"tools/list"}'

the same key works against the OpenAI-compatible REST API at https://www.quorum.dog/v1
```

14

Before you put it in a loop.

Everything so far assumes you are at the keyboard, watching one change. For that case the price of a call is not what you are weighing.

Automation changes the arithmetic. A CI job, an agent working a queue of issues, a codemod crossing a monorepo — each makes the escalation decision hundreds of times, with nobody present to notice that most of them were renames.

That is what **estimate** is for. It classifies a question without running it, free on every call, in about nine tenths of a second — so the loop can decide for itself what deserves a panel.

The gate to branch on is **deliberation_value**, not price. Price tells you what a call costs; this tells you whether the call is worth making.

WHAT ESTIMATE HANDS BACK

DELIBERATION_VALUE	WHAT IT MEANS, AND WHAT TO DO WITH IT
`likely_helps`	The shape of question a panel tends to improve. This is the branch that escalates.
`likely_hurts`	Fires when the answer is a verbatim artifact a synthesis pass would only rewrite — a migration file, a config block, a regex, generated types. In a codebase that is most of the traffic.
`unknown`	No pattern either way. A pass-through, not a reason to spend.

> READ THE BASIS FIELD BEFORE YOU BUILD A POLICY ON THIS
>
> The response carries `basis:"hypothesis"` and an evidence line, and it means what it says: the classification is a per-task pattern observed across 36 tasks, not a measured effect on your work. A reasonable default gate; a poor thing to defend as proven. Log what it said and what the receipt showed, and let your own traffic correct it.

FAIL OPEN, ALWAYS

If the classification call fails, answer with your single model rather than erroring or defaulting to the expensive path. A gate that breaks the build when one endpoint has a bad minute is worse than no gate.

TUNE IT WITH RECEIPTS

If receipts on escalated items keep showing the seats converging with no dissent, escalate less. If reviewers keep overturning single-model answers in the same place, escalate there.

Start with the standing rule and one repository you already care about. You will know inside a week which of the five moments is yours — and how rarely it fires.

WHERE TO GO NEXT

The escalation router guide at `quorum.dog/docs` covers the code version of this decision in full.