AI Deliberation & Vibe Coding

`model: "quorum-standard"`

The one line in the app you generated that decides how hard its hardest question gets argued. Point `baseURL` at Quorum; the rest of your SDK code is unchanged.

A WORKFLOW GUIDE

QUORUM.DOG

What is in here

Sections 03 to 05 are the argument. 06 to 09 are what you choose, where you wire it and what you say. 10 to 12 are the work itself, with exhibits. 13 and 14 are the record and the automation. If you are short of time, read 06 and 08.

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**.

One thing separates this guide from the rest of the series. In an editor, you call a panel while you work and you are the one reading the answer. In something you built, the panel usually goes **into the product** — and you are not its user. Your users are. That changes the door, the limits and the interface, and the next twelve pages are mostly about that.

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 model families sit in which seats, how hard they argue, what a call may cost. It is the string you pass as `model`. §06.

A RECEIPT

The record of a completed deliberation: which model sat in each seat, judge scores, difficulty, cost, latency. §13.

WHERE IT RUNS

An OpenAI-compatible REST API, and a hosted MCP server. For a thing you are shipping, the API is the door. §07.

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 it applies that way consistently. Panels exist for the questions where that matters, and a generated app usually has one or two of them and several hundred of everything else.

AND WHERE IT DOES NOT APPLY

Anything mechanical, anything in a tight loop, anything where one right answer exists. A fast single model is better at those and always will be. Quorum's own `estimate` call returns `likely_hurts` on exactly that class, free, before you spend anything (§14).

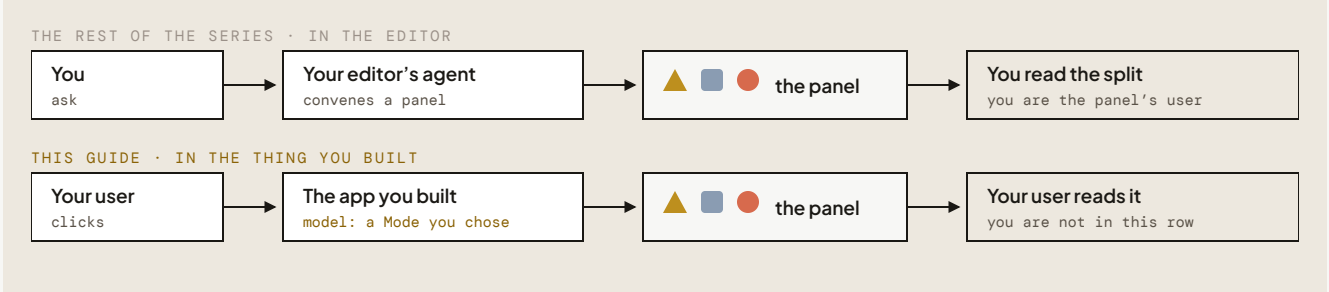

Exhibit 01 — The panel moves out of your hands and into the product. Your only appearance in the second row is the Mode you chose.

04

model: "quorum-standard"

Building the interface got cheap. Building the judgement did not.

You can now describe an app and have it exist by lunchtime. Auth, a database, a deploy, a passable interface. The parts that used to take a fortnight take an afternoon, and the constraint moved somewhere else.

What did not get cheap is anything the app has to be **right** about. Wiring a model call is three lines. Knowing whether the answer coming back can be trusted is the actual product, and no amount of scaffolding generates it for you.

That is the gap. Most vibecoded apps that stall stall there — not because the builder could not ship, but because they shipped something confident and could not tell when it was wrong.

The file opposite runs. There is no bug in it. It is also an entire product compressed into one field name — **verdict** — that nothing else in the repository has any way to check.

WHAT YOU TYPED

"Build me a tool that reads a contract clause and tells the user whether it is risky."

One sentence. Thirty seconds.

ILLUSTRATIVE PROMPT

WHAT IT DID NOT ASK YOU

Which model. What the app should do when the answer is wrong. Whether one opinion is enough for something a user will act on. None of those have a sensible default, and all three were settled anyway.

APP/API/REVIEW/ROUTE.TS · GENERATED, ABRIDGED

```
import OpenAI from "openai";
const client = new OpenAI();

export async function POST(req: Request) {
  const { clause } = await req.json();
  const r = await client.chat.completions.create({
    model: "<fast-model>",
    messages: [
      { role: "system", content: "You are a contract reviewer." },
      { role: "user", content: clause },
    ],
  });
  return Response.json({ verdict: r.choices[0].message.content });
}
```

Exhibit 02 — A prompt, and the file that came back. Illustrative generated code, abridged; not a transcript of a specific session.

Every line in that file is wiring except one. Nothing around `verdict` can tell you when it is wrong, and nothing you prompt for next will add that.

WHERE THE PANEL GOES

Not on every request. On the one screen a user will act on — behind a deliberate action, with an honest wait. §11.

05

The five moments it escalates on.

Three of these are yours, while you are building. Two belong to the app, after you have stopped. Both kinds recur in anything assembled by prompt, whatever it does.

01	**A stack decision made at prompt one**	A database, an auth model, a data shape — chosen in the first thirty seconds by something that was optimising for a working preview, not for the next six months. It never asked you. **§10 works this one end to end.**
02	**The one screen a user acts on**	Most of your app does not need a panel. The review step, the recommendation, the thing somebody will do something about — that one might. This is the only moment where the panel is in the product rather than in your hands. **§11.**
03	**A change you cannot cleanly undo**	A migration, a deletion, an auth swap, a first deploy with real users behind it. Your agent proposes it confidently and it runs in under three seconds. Argue about it while arguing is still free. **§12.**
04	**Your own output, checked**	Generate with a fast model, then pass the result to a panel and ask what is wrong with it. You are buying disagreement, not a rewrite — and the two are easy to confuse when you are moving quickly.
05	**The build that runs and nobody can explain**	You have working software and no intuition for which parts are load-bearing, because you did not write them. A panel argues about it from several directions at once, which is the closest thing available to the review it never got.

THE COMMON SHAPE

Every one is a moment where the code exists and the **judgement** is what remains. Speed did not remove that step; it removed the pause in which you used to notice it. That is the line your standing instruction draws in §09.

AND WHAT IT SHOULD NEVER ESCALATE

Syntax, imports, types, formatting, renames, anything mechanical, anything in a tight loop. Quorum's own classifier agrees: it returns `likely_hurts` when the answer is a verbatim artifact a synthesis pass would only rewrite — which describes most of what a code generator produces.

06

A Mode is the decision you keep.

When wiring a model call is trivial, choosing what sits behind it is where the judgement lives. Two apps can make the identical API call and be completely different products, because **one of them thought about who should be in the room**.

In a Quorum call, **model does not name an engine**. It names a Mode — a saved configuration of which model families sit in which seats, how hard they argue, and what a call may cost.

Engines get deprecated and replaced constantly. A Mode is a statement about *how a question should be argued*, and that outlives whichever specific model was best last quarter. Picking one, or building one, is a product decision that survives the churn underneath it.

This is the part that matters most if you built fast. You can regenerate every file in the repository. You cannot regenerate the judgement about how your app's hardest question should be argued — and in a codebase you did not write line by line, a Mode is very often the only decision that is written down anywhere at all.

You can browse the Modes that exist or build your own. A signed-in account is the whole requirement.

A MODE, WRITTEN DOWN

mode key	contract-review
seats	▲ ■ ● different labs
argument	independent, then critique
judging	scored, split reported
budget	what a call may cost
called as	model: "contract-review"

ILLUSTRATIVE · THE SHAPE OF A MODE

Exhibit 03 — A Mode, written down. Seats are shapes, not vendors; the values are illustrative of the shape of a Mode.

TESTING IT BEFORE YOUR USERS DO

Two of the six tools on the hosted MCP server, `run_test` and `certify_mode`, drive Quorum's Mode testing and certification. They were added to the hosted server on **2026-08-22**; guides that still describe them as local-only are out of date.

Which gives the two doors a clean division of labour: build and test the Mode from your editor over MCP, then ship it as one string over the API. §07.

A Mode is a repeatable decision setup. That is exactly the thing a codebase assembled by prompt does not otherwise contain.

ONE CONSEQUENCE

Changing how your app reasons stops being a code change. It is a string, and the argument behind it is somewhere you can point at.

07

Two doors, and the one line that changes.

The rest of this series is about calling a panel while you work — an agent convenes one mid-task, in your editor. That is not this. If you are vibe coding, Quorum goes into **the thing you built**, not the editor you built it in.

Which means the door you ship on is the API. It is OpenAI-compatible: point `baseURL` at Quorum and change the model string. Existing SDK code runs unchanged, and the second line of the call is the part that matters.

THE WHOLE CHANGE · EXISTING SDK CODE RUNS UNCHANGED

```
const client = new OpenAI({
  baseURL: "https://www.quorum.dog/v1",
  apiKey: process.env.QUORUM_API_KEY,
});

const r = await client.chat.completions.create({
  model: "quorum-standard",   // a Mode, not an engine
  messages: [{ role: "user", content: q }],
});
```

THE OTHER DOOR — SIX TOOLS ON THE HOSTED MCP SERVER AT QUORUM.DOG/MCP

TOOL	COST	WHAT IT IS FOR
`estimate`	Free	Classifies and prices a question without running it, in about 0.9 s. The gate. §14.
`list_modes`	Free	Which Modes your key is allowed to convene.
`deliberate`	Paid	Convenes the panel. The one that costs, and the one you wait for.
`get_receipt`	Free	Per-seat model and judge score, difficulty, cost, latency. §13.
`run_test`	—	Drives Quorum's Mode testing.
`certify_mode`	—	Drives Quorum's Mode certification.

ONE KEY, BOTH DOORS

The same key works against the MCP server and the REST API at `quorum.dog/v1`. Use MCP in the editor while you are choosing and testing a Mode; use the API in the app. Do not ship the editor's connection — your users never touch it.

TWO THINGS THAT LOOK LIKE FAULTS AND ARE NOT

A **GET** to the MCP 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.

08

Two facts decide the architecture.

01 · THERE IS NO STREAMING

`stream: true` returns a **400**. Not degraded, not slower — refused. Almost every generated chat interface streams by default, so the scaffolding you started from probably assumes it. You get a clean error rather than a mystery, and an interface to change.

02 · IT IS NOT FAST

A deliberation runs several models and has them argue. Median **28.5 s**, and **88.2 s** at the ninetieth percentile, measured over 3,845 real deliberations. There is no version of this that feels like autocomplete.

Together those two mean a panel is a **considered interaction, not a conversational one** — and that, not the platform you generated on, is what dictates where the call lives.

MEASURED

P50 28.5 S · P90 88.2 S
N=3,845 · 2026-08-20

WHERE THE CALL BELONGS, BY SURFACE

SURFACE	WHERE THE CALL BELONGS	WHAT BREAKS FIRST
`Bolt`	A server route inside the project, never a component. What previews in the browser also runs in your user's browser.	The key. If it is readable from the client bundle it is public the moment you deploy.
`v0`	A route handler or server action, with the key as an environment variable on the deploy target.	The stream. Generated chat components stream by default; replace the hook with one request and an honest wait.
`Replit`	Server-side in the workspace, with the key in the secrets store rather than in a file.	The timeout. An 88-second p90 is longer than a lot of default request limits — app server, proxy, and the browser's own fetch.
`Lovable`	A backend function, with the key wherever the rest of the app's server-side secrets live.	The same client-side call as Bolt, because the generated front end is where the interesting code tends to accumulate.
`Base44`	Backend functions on Deno; secrets via `secrets.get()`. Note the five-minute function ceiling — ample, but it is a real ceiling.	Calling the panel from the browser. Base44 proxies custom integrations precisely so keys never reach it.

NO CLICK-BY-CLICK, ON PURPOSE

These products ship weekly and a stale set of menu names is worse than none. The outcome is identical on all five: **the call runs on a server you control, the key never reaches the browser, and the interface expects to wait.**

AND WHERE IT DOES NOT FIT AT ALL

Anything mechanical, anything in a tight loop, anything where one right answer exists. A fast single model is better at those and always will be. Most of your app is that.

09

Teach it once.

Give the agent that writes your code a standing rule for when to convene a panel, and the decision stops being yours to remember at the exact moments you are least likely to.

Put it wherever your builder keeps persistent guidance — the project rules file, the agent instructions, the system prompt on your own agent. Then commit it. In a repository you did not write line by line, this is one of the few files that is genuinely yours.

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

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

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

Convene the panel when:
  - you are about to make a choice I will be stuck with: a database,
    an auth model, a data shape, a hosting decision;
  - you have proposed something I cannot cleanly undo -- a migration,
    a deletion, anything that touches live user data;
  - I am choosing between two designs and they are close;
  - you have generated something confident that I have not read.

When you do:
  - put the actual file, schema or diff in the question, not a
    description of it;
  - report where the seats disagreed and do not reconcile it for me;
  - tell me the strongest argument against the thing you are about
    to implement.

Do not convene it for syntax, imports, types, formatting, renaming,
or anything with one right answer. Do those yourself.

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

BEST PRACTICE

Install it, and let it escalate on its own. The whole value is that it catches the decision you did not notice was a decision — and vibe coding is fast precisely because it removes the pause in which you used to notice. If you have to remember to invoke it, you will remember on the afternoon you were already being careful.

THE ONE LINE THAT MATTERS MOST

“Put the actual file, schema or diff in the question.” A panel argues about what it is given. An agent left to summarise will write *a schema that may not scale*; you want *the twelve lines of the table, and the four places the front end parses the JSON column*. Same escalation, an entirely different argument comes back.

10

STACK → SCREEN → SHIP

The stack you are stuck with.

The generator made a dozen decisions in the first thirty seconds and mentioned none of them. Most do not matter. One or two you will be living inside in six months, and they are not the ones it asked you about.

WHAT IT GENERATED

```
package.json · dependencies
  next ^15   react ^19   tailwind ^4
  openai ^4        the product is in here
  <baas>-js ^2     auth + db + storage

db/schema.ts · the only table
  app_data
    id          uuid
    user_id     uuid
    kind        text
    payload     jsonb   ← everything
    created_at  timestamptz
```

ILLUSTRATIVE GENERATED PROJECT, ABRIDGED

WHAT YOUR AGENT SAYS

"The stack is coherent, everything is on a current major version, and the schema is flexible enough to handle new record types without a migration."

All true. None of it is the question. Flexibility is what a schema has instead of a shape, and the agent has no way to know you will still be here in a year.

WHAT THE PANEL RETURNS

▲ **Reasonable for now; the coupling is the risk.** Auth, database and storage are one vendor, so you cannot move any of the three without moving all three.

■ **The coupling is not the problem.** The access rules were generated and nobody has read them, so nobody in this conversation can say what the app currently permits. That is true today, not in a year.

● **Both are arguing about the vendor.** The decision you cannot undo is the shape: everything is one table with a JSON column. You will be migrating out of `payload`, not away from the backend.

The seats did not vote. Two argued about the vendor and one said the vendor is not the decision — and that third position is the one that changes what you do this week.

ASK IT EARLY

This is the cheapest of the three jobs to act on and the one people skip, because at the point you can still change it the app is working and nothing hurts yet.

Exhibit 04 — A generated stack, and where three seats split on it. Seat positions are illustrative of the pattern, not a transcript of a specific call.

11

STACK → **SCREEN** → SHIP

The one screen that has to be right.

This is the only one of the three where the panel is **in the product**. Most of your app does not need it. The review step, the recommendation, the thing a user will act on — that one might.

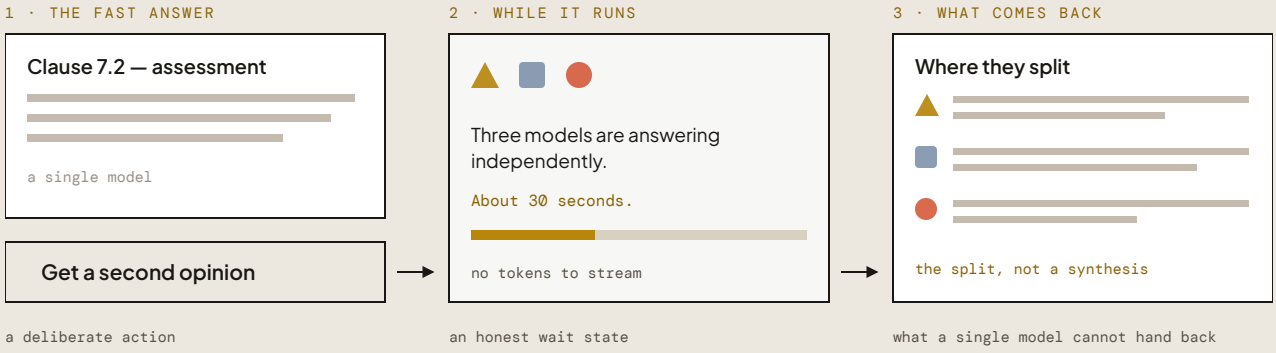

Exhibit 05 — What your user sees while a panel is running. Illustrative interface, not a screenshot of a shipped product.

01	**Behind a button, not in the stream**	"Get a second opinion" is a good affordance: it sets the expectation of a wait and makes the cost legible. The interfaces that fail drop a panel into a chat box, where people expect tokens to appear immediately.
02	**Show them something honest**	Not a fake token stream. Name what is happening — several models answering independently — and say roughly how long. Half a minute is fine if the user knows why.
03	**Return the split, not a synthesis**	A tidy consensus is the least useful thing a panel can hand back, and it is also the one your user cannot tell apart from a single model's answer. If they waited, show them what they waited for.

THE PATTERN UNDERNEATH ALL THREE

Generate with a fast model, then pass the result to a panel and ask what is wrong with it. **You are buying disagreement, not a rewrite** — and if what comes back reads like a better draft, you asked the wrong question.

ONE SCREEN, NOT EVERY SCREEN

If you cannot name the single screen this belongs on, that is the finding, and it is worth more than the integration.

12

STACK → SCREEN → SHIP

The change you cannot undo.

This is the moment to stop generating and start verifying. It never announces itself. It arrives as a command that takes under three seconds to run, on an afternoon that has been going well.

TERMINAL · THE MIGRATION YOUR AGENT JUST WROTE FOR YOU

```
$ npm run db:generate

  1 file written — migrations/0004_drop_payload.sql

  ALTER TABLE app_data DROP COLUMN payload;
  CREATE TABLE documents (id uuid primary key, ...);
  CREATE TABLE reviews   (id uuid primary key, ...);

$ npm run db:push        ← not yet
```

WHAT YOUR AGENT SAYS

"This normalises the schema we discussed. The SQL is valid and the new tables match the shapes the front end already uses."

WHAT IT CANNOT KNOW

Whether the sentence it just wrote about the front end is true. It generated both sides and has no independent read on either.

WHAT THE PANEL RETURNS, BEFORE YOU RUN IT

▲ **Correct SQL, correct target shape.** It is also not reversible, and there is no down migration in the generated file. Run it and the previous shape exists only in whatever backup you have not tested.

■ **Agrees it should happen; disagrees about the order.** Create the two tables, backfill, ship the read path, and drop the column in a later release — three reversible steps instead of one that is not.

● **Both assume the backfill has a target.** Nothing in the repository reads `payload` against a schema — the front end parses it in four places, in four different shapes. There is no single correct destination to migrate to yet.

Three seats, three different objections, and only the first one is about the SQL. That is the difference between a review and a second opinion.

ELAPSED

A panel takes a median of **28.5 seconds**, 88.2 at the ninetieth percentile. The migration took longer than that to write, and there is no figure for undoing it.

P50 28.5 S · P90 88.2 S · N=3,845 · 2026-08-20

Exhibit 06 — A migration, and what the panel said before it ran. Illustrative: terminal output abridged, seat positions typical of the pattern rather than a transcript of a specific call.

13

The receipt, and what you can do with it.

Every deliberation leaves a record. In an app nobody fully read, that record is the only place a decision has a reason attached to it.

WHAT GET_RECEIPT RETURNS

```
request_id          rq_8f2c41a9
mode                contract-review
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

Exhibit 07 — A receipt.

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, and where the threshold in §14 comes from.

Quorum names seats by shape rather than by vendor on anything shareable, so a receipt can be shown to a user, or leave your organisation, without carrying somebody else's trademark.

IT EXPORTS

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

IT TRAVELS

Which is the answer to "why does your app say that?" — a question a generated codebase is otherwise badly placed to answer.

IT TUNES

Receipts that keep showing immediate convergence mean your gate is too loose — you are paying users a wait for agreement. §14.

READING A RUN OF RECEIPTS

WHAT YOU KEEP SEEING	WHAT TO CHANGE
Seats converging immediately, no dissent	Your gate is too loose. You are making users wait for agreement — raise it.
Users pushing back on single-model answers	Your gate is too tight. Lower it, and check which screen the complaints cluster on.
Judge scores splitting hard on one task type	That is the screen the panel belongs on, found from evidence rather than guessed at. §11.

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

A receipt records one deliberation. What it does not do is accumulate: question, disagreement, what the user did next, and eventually whether it worked. Quorum does not do this today — no store, no outcome linkage, no retrieval — and it is named here as a direction the receipt makes possible, not something you can switch on.

14

When the app decides for itself.

Everything so far assumes you are the one choosing. Once the app is live, it is choosing — for every user, on every request, without you in the room. A button answers that for the screen in §11. It does not answer it for the request that arrives while you are asleep.

That is what **estimate** is for. It classifies a question without running it, in about 0.9 seconds, and it is free — not as a trial allowance, but because it costs almost nothing to provide.

LEAD ON THE GATE, NOT ON THE PRICE

FIELD	WHAT YOU BRANCH ON
`deliberation_value`	**The gate.** `likely_helps` / `likely_hurts` / `unknown`. It carries `basis:"hypothesis"` and an evidence line — a per-task pattern over n=36, not a measured effect.
`difficulty_score`	0 to 1 — the dial, once the gate is open. Set the threshold from your own measured distribution, then move it with evidence.
`task_type`	What kind of question it is, which is how you exempt whole categories in one line.
`estimated_price_usd`	What the real call would cost, so a budget ceiling can sit in front of it.
`billed_usd`	Returns 0 on every estimate call, by design.

WHY THE GATE BEATS THE PRICE

`likely_hurts` fires when the answer is a verbatim artifact a synthesis pass would only rewrite — which is most of what a generated app asks a model to do. On those a panel is not expensive so much as **wrong**.

FAIL OPEN, ALWAYS

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

CALL	TIME	WHAT YOU PAY
`POST /v1/estimate`	~0.9 s	Nothing, on every call.
`POST /v1/chat/completions`	28.5 s	Varies by Mode. That is the median; 88.2 s at p90.

Start with one screen and one Mode. Inside a week you will know whether your app's hardest question is one a panel improves — worth knowing either way.

WHERE TO GO NEXT

The escalation router guide at `quorum.dog/docs` covers this branch in full, including how to pick a starting threshold.