AI Deliberation & Your Editor

POST /mcp

One URL, your key as a bearer token, six tools. Whatever agent you already run gets somewhere to go when it is not sure — and comes back with where three frontier models split.

A WORKFLOW GUIDE

QUORUM.DOG

What is in here

03 to 05 are the argument and the mechanism. 06 is the one thing you install. 07 to 09 are the three clients — same server, three wrappers. 10 to 12 are the work. 13 is the record it leaves, 14 is the loop. If you are short of time, read 04 and 06.

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 had one model behind your agent, the distinction that matters is this: asking the same model a second time returns the same reasoning in a second coat. Independent models fail in different places. That is the whole mechanism.

A SEAT

One frontier model on the panel. Three seats, 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. `list_modes` returns the ones your key may call.

A RECEIPT

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

WHERE IT RUNS

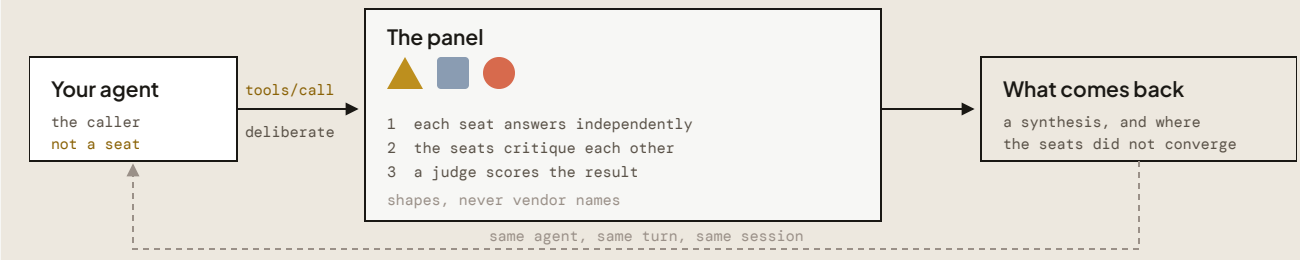
A hosted MCP server on one URL, and an OpenAI-compatible REST API at `quorum.dog/v1`. The same key works against both.

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 for nothing else.

YOUR AGENT IS THE CALLER, NOT A SEAT

Worth being exact about, because you are the one wiring it in. The assistant holding your repo, your context and your standing instruction stays where it is. It calls in a panel of independent models, **none of which is the one you were just talking to**. Its own opinion is not on the panel and should not be counted as if it were.



Inside one call — the caller stays outside the panel, and gets back a synthesis and the split.

04

POST /mcp

What you are connecting to.

A **hosted** MCP server on one URL. Any client that speaks remote MCP can call it with your key as a static bearer token — no package to install, nothing running on your machine.

No OAuth, no app registration, no redirect URL, no refresh. One header is the whole authentication story, which is why the same key works against `quorum.dog/v1` too.

A ROUND TRIP YOU CAN RUN BEFORE YOU TOUCH ANY EDITOR

```
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"}'

{"jsonrpc":"2.0","id":1,"result":{"tools":[
  { "name":"estimate",     "inputSchema":{ ... } },
  { "name":"list_modes",   "inputSchema":{ ... } },
  { "name":"deliberate",   "inputSchema":{ ... } },
  { "name":"get_receipt",  "inputSchema":{ ... } },
  { "name":"run_test",     "inputSchema":{ ... } },
  { "name":"certify_mode", "inputSchema":{ ... } }
]}}
```

Exhibit 01 — A tools/list round trip; input schemas elided for the page. run_test and certify_mode reached the hosted server on 2026-08-22, so a guide that lists four tools predates them.

TOOL	COST	WHAT IT DOES
`estimate`	Free	Classifies and prices a question without running it. §05.
`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	Per-seat model and judge score, difficulty, cost and latency.
`run_test`	—	Runs a mode against a test set.
`certify_mode`	—	Certifies a mode. Bench work rather than turn-by-turn work.

TWO THINGS THAT LOOK LIKE FAULTS AND ARE NOT

A **GET** returns **405**. The server is Streamable HTTP, POST only, and 405 is correct — there is nothing for a GET to stream. And it is **stateless**: no session to keep warm.

WHAT YOU SHOULD SEE IN THE CLIENT

Six tools, and `list_modes` returning real panels. An empty list is the wrapper key in your config, not the server. §07.

05

When your agent should escalate.

Not one language and not one stack. These recur wherever the work is done and the **judgement** is what remains.

01	**A confident diagnosis you are about to act on**	It may be right. It may also cost you an afternoon rebuilding around the wrong lead. **§10 works this one end to end.**
02	**A choice between approaches that are close**	Two designs, three libraries, four migration orders, all defensible. A single model picks one and sounds certain. **§11.**
03	**A change that is hard to reverse**	A schema migration, a published contract, a dependency you will carry for years. Being wrong there is not measured in tokens. **§12.**
04	**What you did not think to ask**	Ask several models what is missing rather than asking one to confirm what is there. Disagreement is the signal.

ESTIMATE — FREE ON EVERY CALL, AND THE GATE YOUR AGENT READS FIRST

```
{
  "deliberation_value": {
    "verdict":  "likely_helps",
    "basis":    "hypothesis",
    "evidence": "per-task pattern, n=36 -- not a measured effect"
  },
  "difficulty_score": 0.78,
  "task_type": "strategy",
  "estimated_price_usd": 0.15,
  "billed_usd": 0
}
```

Exhibit 02 — An estimate response. Field shape abridged, values illustrative.

READ THE VERDICT, NOT THE PRICE

`likely_hurts` fires when the answer is a verbatim artifact a synthesis pass would only rewrite — a lookup, a format, a mechanical transform. `basis: "hypothesis"` is Quorum being exact about what the verdict is: a gate, not a guarantee.

WHAT IT COSTS TO ASK

Classification is a round trip and a cheap one: about **0.9 s**, measured on production. `billed_usd` comes back 0 by design, not as a trial allowance.

06

Teach it once.

Connecting the server gives your agent a capability. This gives it a **rule** — and the decision stops being yours to remember.

Put it wherever your client keeps persistent guidance: a project rules file, a workspace instruction, a skill, or the top of the session you are about to work in. The wording is the same in all three clients.

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. You are the caller, not one of the
seats -- your own opinion is not on the panel.

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

Convene the panel when:
  - you have a confident diagnosis and I am about to act on it;
  - you are choosing between approaches that are close, and the
    choice is hard to reverse later;
  - I ask you to check a design, a migration or a trade-off that I
    will have to defend to someone;
  - I ask what you have missed.

Before you convene it, call estimate. If deliberation_value comes
back likely_hurts, answer the question yourself and tell me why.

When you do convene it:
  - put the actual code, error text, numbers or options into the
    question, not a description of them;
  - report where the seats disagreed and do not reconcile it for me;
  - give me the strongest argument against the option you are
    recommending, and the request_id.

Do not convene it for lookups, syntax, formatting, mechanical
transforms or anything you can simply do. Answer those yourself.

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

BEST PRACTICE

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

THE LINE THAT MATTERS MOST

"Put the actual code into the question." The panel argues about what it is given. An agent left to summarise will write *a caching approach that may have invalidation issues*; you want the function, the TTL and the write path. Same escalation, an entirely different argument comes back.

07

Claude Code.

The next three sections are the same server three times. The URL does not change and the protocol does not change. **The wrapper key does** — and a config with the wrong one fails silently rather than telling you why.

These blocks are written for their own client and are not interchangeable. Copy the one for the editor you are in, not the one you read first.

CLIENT	WRAPPER KEY	WHERE IT GOES	KEY IN THE FILE AS
Claude Code	`mcpServers`	Claude Code's MCP config	`Bearer qk_...`
VS Code	`servers`	.vscode/mcp.json, or your user profile	`${input:quorum-key}`
Cursor	`mcpServers`	.cursor/mcp.json, or ~/.cursor/mcp.json	`${env:QUORUM_API_KEY}`

Exhibit 03 — Same URL, three wrappers. Claude Code and Cursor nest servers under mcpServers; VS Code uses servers. That one word is the difference between a working connector and a silent one.

CLAUDE CODE

```
{
  "mcpServers": {
    "quorum": {
      "type": "http",
      "url": "https://www.quorum.dog/mcp",
      "headers": { "Authorization": "Bearer qk_..." }
    }
  }
}
```

CHECK IT WORKS

Ask your assistant to call `list_modes`. It is free, it requires a valid key, and it returns the panels you are actually entitled to — so a real answer proves the URL, the key and the wiring **all at once**. If you would rather test outside the editor first, that is Exhibit 01.

THEN THE STANDING INSTRUCTION

A connector alone gives the agent a tool it will rarely reach for on its own. Paste §06 into whatever this client keeps as persistent guidance and the escalation becomes automatic. There is a fuller guide to the agent-loop patterns this enables: *Quorum with Claude Code*.

08

VS Code.

VS Code nests servers under **servers**, not **mcpServers**. Everything else is identical to the block on the previous page — and that one word is the whole reason these sections are separate.

The file goes in the workspace at `.vscode/mcp.json`, or in your user profile if you want the server available in every repo you open.

VS CODE · .VSCODE/MCP.JSON (WORKSPACE) · OR YOUR USER PROFILE

```
{
  "servers": {
    "quorum": {
      "type": "http",
      "url": "https://www.quorum.dog/mcp",
      "headers": { "Authorization": "Bearer ${input:quorum-key}" }
    }
  }
}
```

WHY THE KEY IS NOT IN THE FILE

`${input:...}` makes VS Code prompt for the key on first use rather than storing it in a file you might commit. You can hardcode the token instead, and it will work — but a checked-in API key is a bad afternoon, and a workspace file is exactly the kind of file that gets committed by accident.

WORKSPACE OR PROFILE

The workspace file travels with the repository, which is convenient for a team and is the reason to keep the key out of it. The user profile keeps the server with you across every project and never enters version control at all.

CHECK IT WORKS

Ask for `list_modes`. It is free, it needs a valid key, and it returns the panels your key may convene — a real answer proves the URL, the key and the wrapper key at once. An empty tool list with no error is the classic symptom of `mcpServers` pasted into a VS Code file.

THEN THE STANDING INSTRUCTION

§06, in whatever this client keeps as persistent guidance. The connector is the capability; the instruction is what makes it fire.

THE ONLY LINE THAT DIFFERS BETWEEN THE TWO WAYS OF HOLDING THE KEY

```
"Authorization": "Bearer ${input:quorum-key}"   prompts once, stores nothing
"Authorization": "Bearer qk_..."                works, and commits your key
```

09

Cursor.

Back to **mcpServers**, as in Claude Code. What is different here is where the key lives: Cursor interpolates `${env:...}` in both `url` and `headers`.

Project scope is `.cursor/mcp.json`; global scope is `~/.cursor/mcp.json`. Either way the key stays in your environment rather than in the repository.

CURSOR · .CURSOR/MCP.JSON (PROJECT) · ~/.CURSOR/MCP.JSON (GLOBAL)

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

KNOWN CURSOR BEHAVIOUR WORTH KNOWING ABOUT IN ADVANCE

Cursor has a reported issue in which a server that advertises 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. It is here because of the failure mode it produces: if a future version starts advertising them and your Cursor connection breaks while Claude Code and VS Code keep working, this is the first thing to check rather than the last.

CHECK IT WORKS

Ask for `list_modes`. Free, requires a valid key, returns the panels you are entitled to — one call that proves the URL, the key, the interpolation and the wrapper all at once. If it fails, check that the variable is actually exported into the environment Cursor was launched from.

THEN THE STANDING INSTRUCTION

§06, in whatever this client keeps as persistent guidance. Without it the connector sits in the tool list and the agent reaches for it roughly never.

CLOSING THE THREE-CLIENT RUN — WHAT THE SYMPTOMS MEAN

01	**An empty tool list, and no error**	The wrapper key is wrong for this client. Claude Code and Cursor use mcpServers; VS Code uses servers.
02	**A GET to the URL returns 405**	Correct, not a fault. Streamable HTTP is POST only.
03	**Works everywhere but Cursor**	Check the behaviour noted above, then check that the variable is exported into the environment Cursor was launched from.

10

TURN → PLAN → COMMIT

One escalation, end to end.

The agent has read the service, formed a view and has an edit ready to apply. This is the smallest unit of the whole thing: **one turn, one call.**

WHAT THE AGENT SENDS WHEN THE STANDING INSTRUCTION FIRES

```
{"jsonrpc":"2.0","id":7,"method":"tools/call","params":{
  "name":"deliberate",
  "arguments":{
    "mode":"quorum-standard",
    "question":"Read-heavy service, 40k rps, p99 180 ms. Sessions
      live in the app process. We propose moving them to a shared
      cache and keeping a local read-through layer in front of it.
      Is the local layer worth the invalidation risk, or does it
      move the bug somewhere harder to see?"
  }}}
```

Exhibit 04 — The tools/call an agent issues when it escalates. Argument shape illustrative and the question string wrapped for the page; tools/list returns the authoritative schema. The specificity is the point — the panel argues about what it is given.

WHAT COMES BACK

▲ **Worth it, with one condition.** A read-through layer is fine if invalidation is driven by the write path. A TTL here is exactly the bug being described, only slower and harder to reproduce.

■ **Move the sessions and stop there.** A local layer in front of a shared cache is two caches with one invalidation story between them. Ship the shared cache, measure, add the local layer only if p99 still needs it.

● **Both answers assume the premise.** Nothing here establishes that sessions are what makes p99 180 ms. Instrument the hop first; this may be optimising the wrong one.

SEAT POSITIONS ARE ILLUSTRATIVE OF THE PATTERN, NOT A TRANSCRIPT OF A SPECIFIC CALL

The seats did not vote. Two argued about the design; the third asked whether the problem had been established at all — and that is the objection that changes what you do next.

ELAPSED

A panel takes a median of **28.5 seconds**, 88.2 at the ninetieth percentile, on the branch you told it to escalate.

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

11

TURN → **PLAN** → COMMIT

Narrow first, then deliberate.

One turn is easy to reason about. A multi-step piece of work — a migration, an extraction, a rewrite — is where agents start escalating badly, because the obvious move is to hand the panel **the whole plan**.

That is the expensive mistake. You pay three models to read what your agent reads for free, and the question arrives so broad that three models produce three summaries instead of three positions.

The sequence below is four turns in one session. The first three are the agent working alone and cost nothing. Only the fourth convenes a panel — and it convenes it on the *shortlist*, not the plan.

This is the same discipline the standing instruction encodes, applied at the scale of a piece of work rather than a single question.

01	**"What actually has to change?"**	The agent reads the code and lists the steps. This is reading, it is what the agent is good at, and it is free.
02	**"Order them by blast radius."**	Which steps touch data, which touch contracts, which touch only internals. Mechanical. Still no panel.
03	**"Which of those is hardest to reverse?"**	Now the agent is making a judgement call in one voice. Useful — and exactly the point at which a second opinion is worth having.
04	**"Put those two to the panel."**	Three independent reads on the steps that carry the risk. This is the call worth paying for, and it is **one call, not one per step**.

WHY THE ORDER MATTERS

A panel is at its sharpest on a question narrow enough to disagree about. Deliberating on the whole plan is not thoroughness, it is dilution — and it is the single most common way to conclude that panels are not worth it. **Narrow first, then deliberate.**

WHAT YOU ARE LEFT HOLDING

Not "the plan is fine." A named shortlist of the steps that carry the risk, and for each one whether independent reasoners agreed it was reversible. That is a design record, and it did not exist before you asked.

Three of these four turns cost nothing. The one that costs is the one you made narrow enough to disagree about.

WHAT THE SEQUENCE RELIES ON

The connection is stateless: each call sees only its own question. What turns 01 to 03 established has to be written into turn 04.

12

TURN → PLAN → **COMMIT**

The change you cannot take back.

Most decisions are cheap to revisit. A schema migration, a published contract, a deletion — those are not. This is the class where a **split is worth more than an answer**.

```
WHAT A WELL-INSTRUCTED AGENT HANDS BACK

Panel convened (quorum-standard). The seats did not agree.

  Two seats: migrate online. Dual-write, backfill, cut over. The
    schema change is not the risk; the dual-write window is.
  One seat: the dual-write window IS the risk. A short maintenance
    window is cheaper than the class of bug you get from a table
    that is half migrated and still serving.

Strongest argument against the option you are leaning toward:
  a dual-write bug is silent. You find it in support tickets, not
  in CI, and by then both shapes are in production data.

request_id  rq_8f2c41a9      get_receipt for per-seat detail

I have not reconciled these. Both are defensible; the choice is
which failure you would rather be on call for.
```

ILLUSTRATIVE OF THE PATTERN — NOT A TRANSCRIPT OF A SPECIFIC CALL

DO NOT LET IT BE RECONCILED

A split is not a tie to be broken and not a vote to be counted. It is the finding. An agent that averages three positions into one paragraph has handed you a single-model answer that took longer to produce. That is why §06 says **report the split and do not reconcile it**.

AGREEMENT COUNTS TOO

Three independent seats landing in the same place is information, not a wasted call — the difference between "the model said so" and three models from different labs, reasoning separately, finding no objection between them.

The three jobs are one habit at three scales. A turn is a question. A plan is a shortlist of questions. A commit is the question you only get to answer once.

AND WHAT YOU KEEP

The `request_id` in that report is the whole reason the next section exists.

13

The receipt, and what you can do with it.

Every deliberation leaves a record, and `get_receipt` is free to call. It is the difference between "the AI said" and a decision you can hand to somebody else six months later.

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 leave your organisation without carrying somebody else's trademark.

IT EXPORTS

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

IT TRAVELS

That artefact is what goes into the design doc, the RFC or the incident review. The argument travels with the decision instead of evaporating in a terminal scrollback.

IT TUNES

Receipts that keep showing immediate convergence mean your threshold is too low — you are paying for agreement. §14.

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

A receipt records one call. An agent working across a codebase for a year would generate hundreds of them — every judgement it escalated, how the panel split, and what was decided. What Quorum does not do is accumulate them: there is no index across an agent, a repository or a team, no linkage to what actually happened afterwards, and no retrieval beyond a request_id you already hold. It is named here as the direction the receipt makes possible, not as something you can switch on.

Exhibit 05 — A receipt.

14

When the agent runs without you.

Everything so far assumes you are at the keyboard, and for one person on one problem that is right: when the question matters, the price is not what you are weighing.

Unattended work changes the arithmetic. An agent working a queue overnight, or a review firing on every pull request, makes that decision hundreds of times with nobody there to judge it.

That is what **estimate** is for. It classifies a question without running it, free on every call, so the process can decide for itself which items are worth a panel.

The threshold is yours and belongs in your own logic. Quorum's job is to hand you the facts that branch needs, at no cost.

WHAT ESTIMATE HANDS BACK, IN THE ORDER TO READ IT

FIELD	WHAT YOU BRANCH ON
`deliberation_value`	**Read this first.** likely_helps, likely_hurts or unknown — a better gate than a price, and one that carries its own basis.
`difficulty_score`	0 to 1. The dial. Set your threshold from your own measured distribution, then move it with evidence.
`task_type`	What kind of question it is — how you exempt whole categories rather than tuning one number.
`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 — not as a trial allowance.

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 process when one endpoint has a bad minute is worse than no router.

TUNE IT WITH RECEIPTS

If receipts on escalated items keep showing the panel converging with no dissent, raise the threshold — you are paying for agreement. If people keep pushing back on single-model answers, lower it.

Start with one client, one repository and the standing instruction. You will know inside a week which of the four moments is yours.

WHERE TO GO NEXT

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