

Quorum *in* your pipeline

`quorum.converged`

Put a panel at the decision points of an agent pipeline.
Check before an irreversible step, escalate a hard call,
verify an artifact, and send a split to a person.



Where the call goes

An agent pipeline is a chain of steps. Some are mechanical: fetch, parse, convert, write. Others are decisions: ship a migration, merge a change, send a refund, accept an extraction.

Quorum sits at the decision steps. One call returns what several models from different labs concluded independently, and whether they agreed. A split goes to a human.

Picking the steps

Put a panel call where a mistake is expensive to reverse or hard to see. Four questions help decide.

1. Can the step be undone? A database migration, a deploy, an outbound email and a payment are hard to undo. A draft is easy to redo.
2. Will anyone read the output before it takes effect? If a person reviews every result, the pipeline needs a panel less. If the result flows straight into production, it needs one more.
3. Is the answer a judgement? A panel earns its cost on conclusions reached through reasoning: a plan, a diagnosis, a risk assessment, a choice between close options. Lookups, formatting, arithmetic and anything that must be reproduced word for word belong with a single model.
4. What does a wrong answer cost compared to the call? The estimate quotes the Model's surcharge before you spend anything; the bill adds the token charge and the platform fee.

Steps that pass none of these tests keep running on a single model.

The loop

Every Quorum step follows the same loop: estimate, deliberate, get_receipt, act.

The estimate is free and tells you whether to continue. The deliberation runs the panel. The receipt shows who sat in which seat and what each scored. The act step is your pipeline's own branch on the result.

```

import os, requests

BASE = "https://www.quorum.dog/v1"
H = {"Authorization": f"Bearer {os.environ['QUORUM_API_KEY']}" }
MODE = "QRUM:STAN"

# max_usd is an example. Tune it to the Mode you call and what a decision is worth.
def quorum_step(question: str, step_key: str, max_usd: float = 1.00) -> dict:
    msgs = [{"role": "user", "content": question}]

    # 1. estimate: free
    try:
        er = requests.post(f"{BASE}/estimate", headers=H, timeout=30,
                           json={"model": MODE, "messages": msgs})
        est = er.json() if er.ok else None
    except (requests.RequestException, ValueError):
        est = None
    if est is None:
        return {"action": "error", "why": "estimate unavailable"}
    if (est.get("deliberation_value") or {}).get("verdict") == "likely_hurts":
        return {"action": "skip", "why": "a panel would rewrite an exact answer"}
    # estimated_price_usd is the Mode surcharge only. The bill adds the token
    # charge and the platform fee, so leave headroom under max_usd.
    if (est.get("estimated_price_usd") or 0) > max_usd:
        return {"action": "skip", "why": "over budget", "estimate": est}

    # 2. deliberate: runs several models, so a long timeout. Make step_key
    # unique per run and step, e.g. f"{run_id}:{step_name}", so a retry of this
    # run replays instead of billing twice. Replay matches on the key alone and
    # does not compare the question, so a key reused by the next run would
    # return this run's answer to a different question.
    try:
        resp = requests.post(f"{BASE}/chat/completions",
                             headers={**H, "Idempotency-Key": step_key},
                             timeout=300,
                             json={"model": MODE, "messages": msgs,
                                    "quorum": {"max_cost_usd": max_usd}})
    except requests.RequestException as e:
        return {"action": "error", "code": type(e).__name__}
    try:
        r = resp.json()
    except ValueError:
        r = {} # a gateway error can return HTML instead of JSON
    if not resp.ok or "quorum" not in r:
        return {"action": "error", "code": (r.get("error") or {}).get("code") or
        resp.status_code}
    q = r["quorum"]

    # 3. get_receipt: who sat in which seat, what each scored
    receipt = None
    try:

```

```

rr = requests.get(f"{BASE}/receipts/{q['request_id']}", headers=H, timeout=30)
if rr.ok:
    receipt = rr.json().get("receipt") # absent when receipt_available is false
except (requests.RequestException, ValueError):
    pass # the answer stands without its receipt

# 4. act. A replay carries no split fields, so it goes to a person.
replayed = bool(q.get("replayed"))
split = q.get("converged") is False or q.get("remaining_friction") is not None
return {
    "action": "human" if (split or replayed) else "proceed",
    "answer": r["choices"][0]["message"]["content"],
    "contested": q.get("contested_passage"),
    "replayed": replayed,
    "capped": bool(q.get("capped")),
    "request_id": q["request_id"],
    "receipt": receipt,
}

```

The sample calls Standard Mode · QURUM:STAN. The `model` field takes the ticker. `list_modes` over MCP and `GET /v1/models` list the Modes your key can use. Each Mode lists its own surcharge.

Three places a step belongs

Checkpoint

Before an action that cannot be undone, ask the panel whether the plan holds. The agent proposes the migration, the panel reads it, and the pipeline continues when the seats agree. The prompt carries the plan and the context the plan depends on: the schema, the row counts, the rollback path.

A useful checkpoint prompt names what to look for. "List the assumptions in this plan that, if wrong, cause data loss" gives the seats a job. "Is this a good plan" returns opinions.

Escalate

A step the agent is unsure about goes to the panel in place of a single guess. The agent decides it is unsure by its own signal: a low self-reported confidence, a tool result that contradicts its plan, a rule that always escalates a category. The escalation router guide has the routing code that combines the free estimate with those signals.

Verify

After a step produces an artifact such as a diff, a plan or an extraction, hand the artifact to the panel and ask what is wrong with it. The seats review the same artifact independently, so a defect found by one seat and missed by another shows up as a split. The GitHub and CI guide builds this step into a pull request check.

All three jobs share a call shape: the pipeline sends a question and reads the same response fields.

What to do with each outcome

A call returns one of five outcomes. The pipeline needs a branch for each.

OUTCOME	HOW IT SHOWS	PIPELINE ACTION
Agreed	HTTP 200, <code>quorum.converged</code> is <code>true</code> , <code>remaining_friction</code> is <code>null</code>	Proceed with the answer.
Split	<code>quorum.converged</code> is <code>false</code> , or <code>remaining_friction</code> is not <code>null</code>	Pause. Send the contested passage and the receipt to a person.
Replayed	<code>quorum.replayed</code> is <code>true</code>	Treat as unverified. Send to a person or reuse your stored decision.
Capped	<code>quorum.capped</code> is <code>true</code>	Keep the answer, log the bill, and move on by the agreed or split rule.
Error	A non-2xx status with an <code>error.code</code>	Retry with backoff on a 429, a 5xx or a <code>409</code> <code>request_in_progress</code> , then fall back to a person or a single model.

`quorum.converged` is `null` on the express lane, where no judge read the round. A `null` is a single-model answer, and the pipeline treats it as one.

Check the HTTP status and `error.code` before reading `choices`. An error body carries no `choices`, so indexing into it raises an exception.

Agreed

When the seats agree, the receipt shows which models reached the same answer independently. Treat it as an input to your pipeline's decision. Proceed, and store the `request_id` with the action so a later audit can pull the receipt.

Split

When the seats disagree, the disagreement is the result, and a person should read it. Three fields on `quorum` carry it.

FIELD	READ IT AS
<code>converged</code>	<code>false</code> when the panel did not settle on one position.
<code>remaining_friction</code>	Usually <code>factual_conflict</code> , when the judge flagged a factual conflict between seats, or <code>null</code> . It can carry a short sentence from the judge naming the disagreement.
<code>contested_passage</code>	An object <code>{ seat, quote, why }</code> : the seat, the passage it quoted and why it is contested. <code>null</code> when no single passage carried the split.

Route on `converged === false` or a non-null `remaining_friction`. Show the reviewer `contested_passage.quote` and the receipt together with the answer. The receipt lists each seat, so the reviewer can tell a real disagreement from seats agreeing for the same reason.

Design the human step as part of the pipeline. A ticket in a review queue, a message in a channel with approve and reject buttons, or a paused run that a person resumes all work. Record who ruled and what they ruled, and feed that record back into the next prompt when it applies.

Replayed

A repeat of a call that already succeeded returns a replay with `quorum.replayed: true`. The replay holds the answer but lacks `converged`, `remaining_friction`, `contested_passage` and `billed_usd`, and it bills nothing. Only calls that succeeded are replayed.

Replays happen when a pipeline retries with the same idempotency key, which is the correct behaviour after a timeout or a worker restart. A retry that arrives while the first call is still running returns `409 request_in_progress`. Wait, then retry with the same key. Check `replayed` first and read the other fields with a default, as the sample does. A replayed answer has no recorded split state, so the sample sends it to a person. A pipeline that stores its own decision next to the idempotency key can look the decision up and skip the person.

Capped

Each API-key call has a cost ceiling: the higher of \$2.00 and the price of the Mode being called, unless the API key carries its own ceiling. A request can lower it with `quorum.max_cost_usd` and no request can raise it. A response with `quorum.capped: true` ran in full and the bill stopped at the ceiling. The answer is complete. Log the event so you can see which steps run close to the limit.

Do not retry a capped call to get a cheaper result. A capped call is not replayed, so a retry runs the panel again and bills again.

Error

Errors carry an `error.code`. A prompt longer than the input limit returns `413 context_length_exceeded`, and the limit is 8000 tokens per request by default. Shorten the prompt or split the artifact. For other codes, see the errors reference. A pipeline that cannot reach a panel needs a defined fallback, chosen per step: pause for a person at an irreversible step, continue on a single model at a reversible one.

Cost control

Quorum prices a call in three parts. The token charge is \$2.00 per 1M input tokens, \$8.00 per 1M output tokens. The platform fee is 5% on seats that run on Quorum's keys and 0.5% on seats that run on your own key. The Mode adds its surcharge, which `list_modes` shows. The bill is the sum, and the receipt itemises it, with a line per seat on API-billed seats.

The estimate returns `estimated_price_usd`, which is the Mode's surcharge for the question's depth. The token charge and the fee come on top, so compare it with a budget that leaves room.

A pipeline can control spend in four ways.

1. **Estimate first.** The call is free and returns the verdict, the difficulty and the surcharge before any panel runs.
2. **Skip on `likely_hurts`.** That verdict marks answers that are a passage to reproduce, where a rewrite makes the result worse.
3. **Set a ceiling per step.** `quorum.max_cost_usd` bounds the bill of one call. Give a checkpoint on a production migration a higher ceiling than a verification of a draft.
4. **Count what runs.** Log the step name, the `request_id` and the outcome. A weekly total per step shows which placements pay for themselves.

The client timeout for a deliberation is 300 seconds, because a panel runs several models. The estimate response carries a `timing_ms` field, so you can measure its time on your own network.

A constructed example, with illustrative figures: a pipeline runs a checkpoint before each production migration, and a migration is a few times a week. The team sets a ceiling of \$2.00 on the checkpoint step and \$1.00 on a draft verification step. A step whose estimate exceeds

its ceiling is skipped and logged, and a person is asked instead. The ceilings are the team's own numbers and move with what a bad migration costs them.

Writing the question

The panel reads what the pipeline sends. Three habits improve the result.

Send the artifact and the context together. A diff without the schema it touches invites guesses.

Ask for specifics. "Cite the line and say what breaks" gives the judge something to compare across seats.

Ask one question per call. A prompt that bundles five questions produces an answer that averages them, and a split on one of the five hides inside it.

Convene a panel for judgement calls: a consequential assumption, options close enough that the numbers no longer separate them, a diagnosis you cannot check, a decision you will have to defend, or a "what have I missed" check. Use a single model for lookups, syntax, formatting, conversions, arithmetic and anything you need reproduced word for word.

From an MCP client

An agent connected over MCP has the same loop as tools: `estimate`, `deliberate` and `get_receipt`. Quotes the Mode's surcharge for the question and says whether a panel is likely to help, free of charge. The bill adds the token charge and the platform fee. Sends your question to a panel of models and returns where they agree and where they split. Shows which model sat in each seat of a past deliberation, with judge scores, cost and latency. A pipeline that runs inside an agent host such as an editor or a coding agent calls the tools directly and applies the same outcome table to what `deliberate` returns. The connect guide covers setup for each client.

Next

- Escalation router: decide per question whether a panel is worth calling.
- GitHub and CI: a pull request check.

- Agent frameworks: LangGraph, CrewAI and the OpenAI Agents SDK.
- Errors and rate limits.

`quorum.dog/docs`