

Quorum & GitHub CI

`gh pr comment`

Run a Quorum panel on every pull request from GitHub Actions, and give the Copilot coding agent the same tools.



How the review works

A GitHub Actions workflow sends each pull request's diff to a Quorum panel, reads whether the seats agreed, and posts the result as a comment. A split can block the merge. A re-run of the same commit replays the first answer at no charge when the first call succeeded. The same account can also give the Copilot coding agent the Quorum tools, so the agent checks its own work before it opens a pull request.

Both paths use a Quorum API key. The key bills your organization's API account.

Why a panel on a pull request

A diff review is a judgement task. The reviewer reads a change and decides whether it is correct, whether it is safe, and whether it loses data. A panel gives several independent readings from models built by different labs, and the judge records where they diverged. A defect that one seat sees and another misses appears as a split, and the pull request goes to a person with the passage in dispute attached.

The panel runs next to your human reviewers. It reads each pull request the workflow runs on as soon as it opens or updates, and a person reads the ones that need a person.

What to send

The request has two parts: an instruction and the diff.

The instruction. A short system message names the job and the output. The workflow below asks for correctness, security and data-loss risks, with file and line, and asks the seats to say whether the change is safe to merge. A specific instruction gives the judge something to compare across seats. A general one such as "review this" gives each seat room to comment on style.

The diff. Send the three-dot diff against the base branch, so the panel sees what the pull request adds and not what the base gained since the branch started. The workflow cuts the diff at 24,000 characters. The default input limit is 8000 tokens per request, estimated as characters divided by four, which is about 32,000 characters including the instruction. A request over the limit returns `413 context_length_exceeded`; a 24,000-character diff plus its instruction stays under the default. For larger changes, send one call per directory.

Context. A diff shows the lines that changed. A migration that renames a column reads differently once the panel knows which services read it. Add the pull request title and description to the user message when they carry that information. Exclude files that carry no review signal, such as lockfiles and generated code. A variant of the diff line that does it:

```
git diff "origin/${{ github.base_ref }}...HEAD" -- . ':(exclude)package-lock.json' ':(exclude)*.min.js' | head -c 24000 > diff.txt
```

The diff leaves your repository on every run. Send only what the review needs.

The workflow

Create the secret first: repository Settings, Secrets and variables, Actions, new secret named `QUORUM_API_KEY`. Then save this as `.github/workflows/quorum-review.yml`.

```

name: Quorum review
on:
  pull_request:
    types: [opened, synchronize]

permissions:
  contents: read
  pull-requests: write

jobs:
  review:
    if: github.event.pull_request.head.repo.full_name == github.repository
    runs-on: ubuntu-latest
    timeout-minutes: 10
    env:
      QUORUM_API_KEY: ${ secrets.QUORUM_API_KEY }
      BASE: https://www.quorum.dog/v1
      MODEL: QRUM:STAN
      GH_TOKEN: ${ github.token }
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0

      - name: Build the request from the diff
        run: |
          git diff "origin/${ github.base_ref }...HEAD" | head -c 24000 > diff.txt
          jq -n --arg model "$MODEL" --rawfile diff diff.txt '{
            model: $model,
            messages: [
              {role: "system", content: "You review pull requests. List correctness,
security and data-loss risks in this diff. Cite file and line. Say if the change is safe
to merge."},
              {role: "user", content: $diff}
            ]
          }' > request.json

      - name: Estimate
        run: |
          curl -sS "$BASE/estimate" \
            -H "Authorization: Bearer $QUORUM_API_KEY" \
            -H "Content-Type: application/json" \
            --max-time 30 -d @request.json | tee estimate.json
          if [ "$(jq -r '.deliberation_value.verdict // ""' estimate.json)" =
"likely_hurts" ]; then
            echo "skip=true" >> "$GITHUB_ENV"
          fi

      - name: Deliberate
        if: env.skip != 'true'
        run: |

```

```

curl -sS "$BASE/chat/completions" \
  -H "Authorization: Bearer $QUORUM_API_KEY" \
  -H "Content-Type: application/json" \
  -H "Idempotency-Key: pr-{{ github.event.pull_request.number }}-{{
github.event.pull_request.head.sha }}" \
  --max-time 300 -d @request.json > response.json
jq -e '.choices[0].message.content' response.json > /dev/null

- name: Post the result
  if: env.skip != 'true'
  run: |
    RID=$(jq -r '.quorum.request_id' response.json)
    REPLAYED=$(jq -r '.quorum.replayed // false' response.json)
    if [ "$REPLAYED" = "true" ]; then
      SPLIT=unknown
    else
      SPLIT=$(jq -r '(.quorum.converged == false) or (.quorum.remaining_friction !=
null)' response.json)
    fi
    echo "split=$SPLIT" >> "$GITHUB_ENV"
    {
      if [ "$SPLIT" = "true" ]; then
        echo "**The panel split on this change. A person should read it before
merge.**"
      echo
      jq -r '.quorum.contested_passage | if . then "Contested (\(.seat //
"unknown seat")): \"\(.quote)\"\\n\(.why // "")" else "Contested: see the answer below"
end' response.json
      elif [ "$SPLIT" = "unknown" ]; then
        echo "***Replayed answer from an earlier run of this commit. The panel's
agreement was not stored, so a person should read it.**"
      else
        echo "**The panel agreed.**"
      fi
      echo
      jq -r '.choices[0].message.content' response.json
      echo
      if [ "$REPLAYED" = "true" ]; then
        echo "<sub>Receipt: \`${RID}\` · replayed, no new charge</sub>"
      else
        echo "<sub>Receipt: \`${RID}\` · billed \\\$(jq -r '.quorum.billed_usd //
empty' response.json)</sub>"
      fi
    } > comment.md
    gh pr comment "${{ github.event.pull_request.number }}" --body-file comment.md

```

What each step does

The request step builds the JSON with `jq`, so the diff is escaped correctly whatever characters it holds. The `model` value is the ticker, QRUM:STAN, set once in the job's `env`.

The estimate step is free. The job skips the deliberation when `deliberation_value.verdict` is `likely_hurts`, the verdict for answers that are a passage to reproduce.

The deliberate step runs the panel. `--max-time` is 300 seconds, because a deliberation runs several models. The `jq -e` line exits non-zero when the response has no `choices[0].message.content`, which fails the job on an error response. To see why, print the code before the `jq -e` line:

```
jq -r '.error.code // empty' response.json
```

The post step reads the result, picks one of three headlines and comments.

Idempotent re-runs

The `Idempotency-Key` header is the pull request number and the head commit. GitHub Actions re-runs happen often: a flaky step, a manual re-run, a retried job after a runner failure. Without the header, the API derives a key from your organization, the API key, the Mode and the exact messages, so a re-run that sends the same diff replays too. The explicit header ties the replay to the commit, and it still matches if you edit the prompt text around the diff between runs.

With the key, a re-run of the same commit replays the original answer when the first call succeeded. A replayed response carries `quorum.replayed: true`. It holds the answer and lacks `converged`, `remaining_friction`, `contested_passage` and `billed_usd`, and it bills nothing. A new push has a new head commit and so a new key, which gets a fresh review.

Only calls that succeeded are replayed. A call that ended `capped` is not replayed, so a re-run of it runs again and bills again.

Reading the result

The comment opens with one of three lines.

HEADLINE	CAUSE	WHAT A PERSON DOES
The panel agreed.	<code>converged</code> is <code>true</code> and <code>remaining_friction</code> is <code>null</code> .	Read the answer as one input to your own review.
The panel split on this change.	<code>converged</code> is <code>false</code> , or <code>remaining_friction</code> is not <code>null</code> .	Read the contested passage first, then the full answer.
Replayed answer.	<code>quorum.replayed</code> is <code>true</code> .	Read the answer. The agreement was not stored for a replay, so the job reports no verdict.

The contested passage is an object with `seat`, `quote` and `why`. It is `null` when no single passage carried the split, and the comment then points to the answer below it.

The footer carries the receipt id. `GET /v1/receipts/{request_id}` returns each seat, what the judge scored it and what the call cost, and gives a reviewer the means to tell a real disagreement from seats arriving at the same view. On API-billed seats the receipt has one line per seat with the source, the real cost, the fee rate and the fee.

`quorum.converged` is `null` on the express lane, where no judge read the round. The workflow treats that case as agreement because the comparison is `== false`. If you want a pull request that reached the express lane to get a person, check for `null` and route it.

When to block and when to comment

A comment leaves the merge to the reviewer; a failing check stops it. Start with comments and move to blocking when the team trusts what it sees.

To make a split block the merge, add a final step to the job:

```
- name: Block on a split
  if: env.skip != 'true' && env.split == 'true'
  run: exit 1
```

Then mark the job as a required check in branch protection.

Two details decide how that behaves on a re-run. After a split, the first run fails the check. A re-run of the same commit replays the answer, `split` becomes `unknown`, and the check passes with nobody having read the split. To keep a replayed commit blocked until a person acts, replace the condition with `env.skip != 'true' && env.split != 'false'`. The `skip` test keeps the step from failing on pull requests that the estimate skipped, which never set `split`.

A starting policy:

- Comment on every pull request.
- Block on a split for paths where a defect costs the most, such as migrations, authentication and billing. A `paths` filter on the trigger limits the workflow to those directories.
- Let a follow-up commit clear a blocked check. A new head commit gets a new key and a fresh review.

Cost

An API-key call bills three things. 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 estimate returns `estimated_price_usd`, which is the surcharge alone, so the bill for the real call exceeds it by the token charge and the fee.

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. To set a ceiling for the review job, add `"quorum": {"max_cost_usd": 1.00}` to the request JSON. A response with `quorum.capped: true` ran in full and the bill stopped at the ceiling. The comment footer shows `billed_usd` for each run, so a team sees what each pull request cost.

Three habits keep the total predictable. The workflow triggers on `opened` and `synchronize`, and each push to an open pull request reviews again, so ask contributors to push in batches. The `paths` filter limits runs to the directories that matter. The diff cap bounds the input size.

Secrets and fork pull requests

The key lives in a repository secret and reaches the job through `env`. Three practices keep it safe.

- Create a key for this workflow alone, so you can revoke it without touching other systems.
- Keep the key out of logs. The workflow passes it in a header and never echoes it.
- Store the key as an environment secret with required reviewers if the repository wants a person to approve each run.

GitHub runs workflows triggered by pull requests from forks without the repository's secrets. The `if` on the job line, `github.event.pull_request.head.repo.full_name == github.repository`, limits the job to branches in the same repository, so a fork's pull request skips it and no run fails for a missing secret. A maintainer who wants a panel to read a fork's change can push the commits to a branch in the repository and open a pull request from there.

Copilot coding agent

The Copilot coding agent works on an issue and opens a pull request. It reads MCP servers from the repository's settings, so it can call Quorum while it works. GitHub connects with an API key, so the setup uses a secret in place of a sign-in.

Open Settings, Copilot, Coding agent, MCP configuration, and add:

```
{
  "mcpServers": {
    "quorum": {
      "type": "http",
      "url": "https://www.quorum.dog/mcp",
      "headers": { "Authorization": "$COPILOT_MCP_QUORUM_AUTH" },
      "tools": ["estimate", "deliberate", "get_receipt", "list_modes"]
    }
  }
}
```

Then add a secret named `COPILOT_MCP_QUORUM_AUTH` in the repository's `copilot` environment. Its value is the whole header value: `Bearer` followed by your Quorum API key. The server URL is `https://www.quorum.dog/mcp`.

The `tools` list sets what the agent can call. 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. Lists the Modes you can call, with their prices and the providers behind them.

`deliberate` costs money, so `estimate` and `get_receipt` sit beside it. The agent estimates first and reads the receipt after. The loop is estimate, deliberate, get_receipt, act. Add a line to the repository's agent instructions to tell the agent when to use it:

Before you open or update a pull request that touches migrations, auth or billing, call estimate, then deliberate on the diff. If the seats split, stop and ask for a human review.

With that instruction, the coding agent sends its diff to a panel before it opens the pull request, and the workflow reviews the pull request again. The workflow reviews every pull request, whether a person or an agent wrote it. Each call leaves a receipt.

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.

Checklist

1. Create a Quorum API key for the repository and store it as `QUORUM_API_KEY`.
2. Save the workflow and open a test pull request that touches one file.
3. Read the comment: the headline, the contested passage if there is one, and the receipt id.
4. Re-run the job and confirm the second comment says replayed and bills nothing.
5. Decide where a split blocks the merge, and mark the job as a required check for those paths.
6. Add the Copilot MCP configuration if you use the coding agent.

Next

- Quorum as a pipeline step
- Escalation router
- Errors and rate limits