CodeWikiBench Rubric and Evaluation Overview

November 18, 2025

1 Documentation Sources

Many benchmark runs only include alternative documentation drops (e.g., `codewiki` or `deepwiki`) rather than the official `original` folder. The pipelines described below accept any parsed source as long as `docs_tree.json` and `structured_docs.json` are present under `data/<repo>/<source>`. When `original` is missing, simply pass the folder name via `--docs-source` (rubrics) or `--reference` (evaluation).

2 Rubric Generation Pipeline

For each requested model, `run_rubrics_pipeline.sh` locates the documentation source, launches `generate_rubrics.py`, and stores per-model hierarchies. The Python entry point performs the following steps:

1. Call `detect_docs_source()` to locate a folder containing `docs_tree.json`.
2. Load the tree, embed it into the prompt, and instantiate the `pydantic_ai.Agent` (optionally with the `docs_navigator` tool).
3. Run the agent, slice the JSON array from the raw response, and persist `rubrics/<model>.json`. Each leaf already includes weights in {1,2,3} and references back to documentation paths.
4. Invoke `visualize_rubrics()` for quick inspection.

After all models finish, `combine_rubrics.py` loads every JSON, performs an Anthropic-powered semantic merge (falling back to name-based dedupe if needed), and emits `rubrics/combined_rubrics.json` along with statistics (node counts, depth, weight distribution).

3 Evaluation Pipeline

The evaluation entry point mirrors the generation flow:

1. Choose a reference docs folder via `--reference` (auto-detected when omitted).
2. Load `rubrics/combined_rubrics.json` (or a per-model file supplied via `--rubrics-file`).
3. Collect leaf requirements and evaluate them with `judge/judge.py`. Each leaf gets a binary score and evidence.
4. Propagate scores upward via `calculate_scores_bottom_up`: every parent score is a weight-normalized sum of its children's scores.
5. Combine multi-model evaluations through `combine_evaluations.py` and optionally visualize the aggregated report.

Table 1: IEEE-style pseudocode for rubric generation.

Algorithm 1: Rubric Generation with Alternate Documentation Sources

Input: repository identifier R, docs source override s, model list $\mathcal{M}$, tool flag *use_tools*
Output: per-model rubric JSON files H_m and combined hierarchy $\mathcal{H}$

1:	$d \leftarrow \textsc{DetectDocsSource}(R, s)$	Honor `codewiki`/`deepwiki` when `original/` is missing.
2:	$\textsc{EnsureDocsTree}(d)$	Abort unless `docs_tree.json` exists.
3:	**for each** $m \in \mathcal{M}$ **do**	
4:	$T \leftarrow \textsc{LoadDocsTree}(d)$; $S \leftarrow \textsc{LoadStructuredDocs}(d)$	
5:	$A \leftarrow \textsc{InstantiateAgent}(m, T, S, use\_tools)$	
6:	$H_m \leftarrow \textsc{RunRubricAgent}(A)$	Produce weighted hierarchy with documentation references.
7:	$\textsc{PersistRubric}(H_m, R, m)$	Write `rubrics/<m>.json` to disk.
8:	**end for**	
9:	$\mathcal{H} \leftarrow \textsc{CombineRubrics}(\{H_m\})$	Anthropic semantic merge with deterministic fallback.
10:	$\textsc{VisualizeRubrics}(\mathcal{H})$	Optional hierarchy plots and statistics.

4 End-to-End Algorithms

Both pipeline orchestrations follow the same high-level pattern described in Issue `2025-11-18-12-20-22`: locate a parsed documentation folder, fan out per model, and persist JSON artifacts before any visualization or aggregation happens.

4.1 Rubrics Pipeline Algorithm

1. **Docs source detection.** `run_rubrics_pipeline.sh` scans `data/<repo>/` for a folder that exposes `docs_tree.json`. Operators can override the auto-detected folder via `--docs-source` to target `codewiki`, `deepwiki`, or any other parsed drop when `original/` is missing.

2. **Model fan-out.** Once the source folder is confirmed, the script loops over the requested models (via `--models`) and calls `rubrics_generator/generate_rubrics.py` per model so that failures are isolated.

3. **Agent execution.** The generator loads `docs_tree.json`, optionally wires in the `docs_navigator` tool, and asks the Pydantic-AI agent to return a weighted hierarchy with embedded documentation references.

4. **Persistence and visualization.** Each output is saved to `rubrics/<model>.json`, after which `combine_rubrics.py` may run to build `combined_rubrics.json`, collect statistics (counts, depth, weight histogram), and optionally render a visual summary.

4.2 Evaluation Pipeline Algorithm

1. **Reference folder selection.** `run_evaluation_pipeline.sh` mirrors the earlier detection logic but reads `--reference` (default `original`). Any folder that exposes `docs_tree.json` + `structured_docs.json` can be supplied here.

2. **Leaf scoring.** For each model, `judge/judge.py` loads the rubrics (combined or per-model), extracts every leaf requirement, and queries the model for a binary score with reasoning/evidence.

3. **Score propagation and aggregation.** `calculate_scores_bottom_up` normalizes each parent's score by the children's weights, `combine_evaluations.py` fuses multi-model runs, and `visualize_evaluation.py` can be used to export a narrative report.

4.3 Handling Missing `original/`

Neither algorithm depends on the folder name; they only require the canonical parsed outputs. When `data/<repo>/original` is absent, run the rubrics pipeline with `--docs-source <existing_folder>` and the evaluation pipeline with `--reference <existing_folder>`. Both scripts abort early if `docs_tree.json` is missing, which makes misconfigurations obvious.

Mermaid-style Summary. The following block reproduces the execution sketch from the issue for quick embedding into README or wiki contexts:

```
```mermaid
flowchart TD
    S([Docs sources]) -->|tree + structured docs| P[Parsed folder]
    P -->|--docs-source| R[run_rubrics_pipeline.sh]
    R -->|per model| G[generate_rubrics.py]
    G --> RC[combine_rubrics.py]
    RC --> RUB[rubrics store]
    RUB -->|--reference| E[run_evaluation_pipeline.sh]
    E --> J[judge/judge.py]
    J --> CE[combine_evaluations.py]
    CE --> COMB[combined evaluation JSON]
    COMB --> VIS[visualize_evaluation.py]
```
```

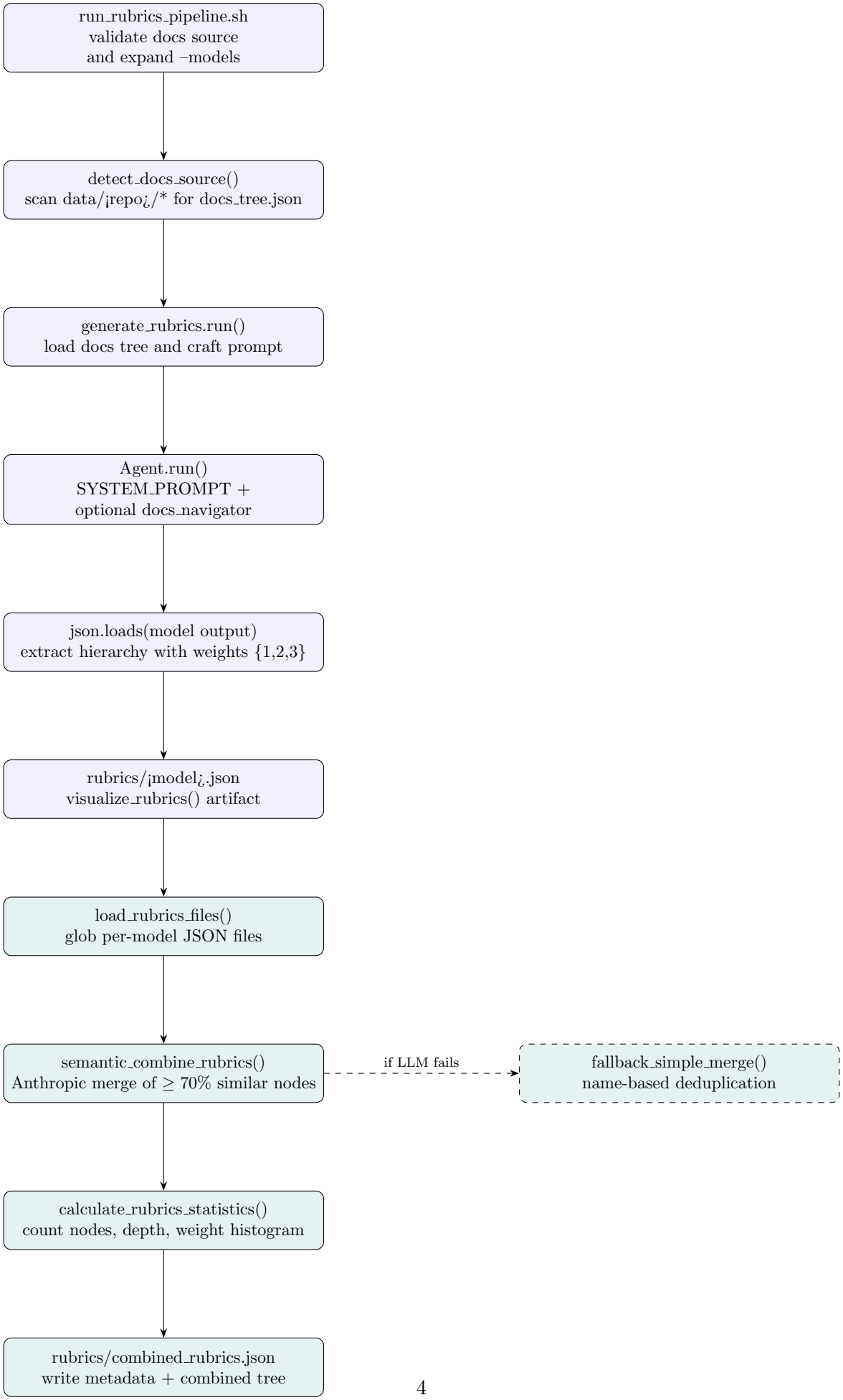
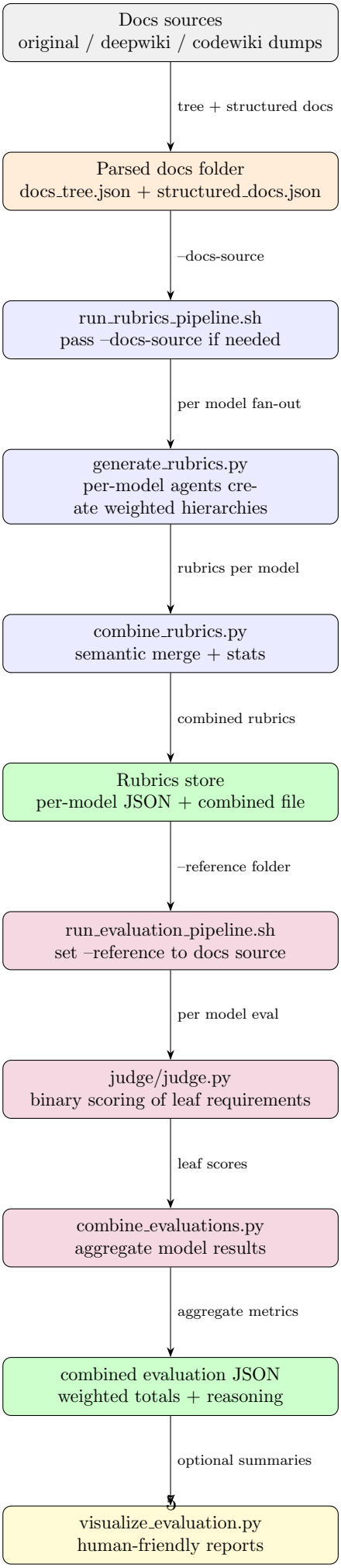


Figure 1: Rubric generation and combination pipeline


Docs sources
original / deepwiki / codewiki dumps
tree + structured docs
Parsed docs folder
docs_tree.json + structured_docs.json
–docs-source
run_rubrics_pipeline.sh
pass –docs-source if needed
per model fan-out
generate_rubrics.py
per-model agents cre-
ate weighted hierarchies
rubrics per model
combine_rubrics.py
semantic merge + stats
combined rubrics
Rubrics store
per-model JSON + combined file
–reference folder
run_evaluation_pipeline.sh
set –reference to docs source
per model eval
judge/judge.py
binary scoring of leaf requirements
leaf scores
combine_evaluations.py
aggregate model results
aggregate metrics
combined evaluation JSON
weighted totals + reasoning
optional summaries
visualize_evaluation.py
human-friendly reports