Artificial Intelligence (AI), Large Language Models (LLMs) and RiverWare

How We Use Generative AI for Software Development

Guidelines and safeguards

- Follow CU AI policies — All use of generative AI must comply with CU policies.
- Protect university and sponsor IP — AI providers may not store or train on our software.

How we use AI today

- Primary uses: Code completion and chat
- Goals: Improve development efficiency and code quality
- Agents: Limited use of generative AI agents

Exploratory Analysis

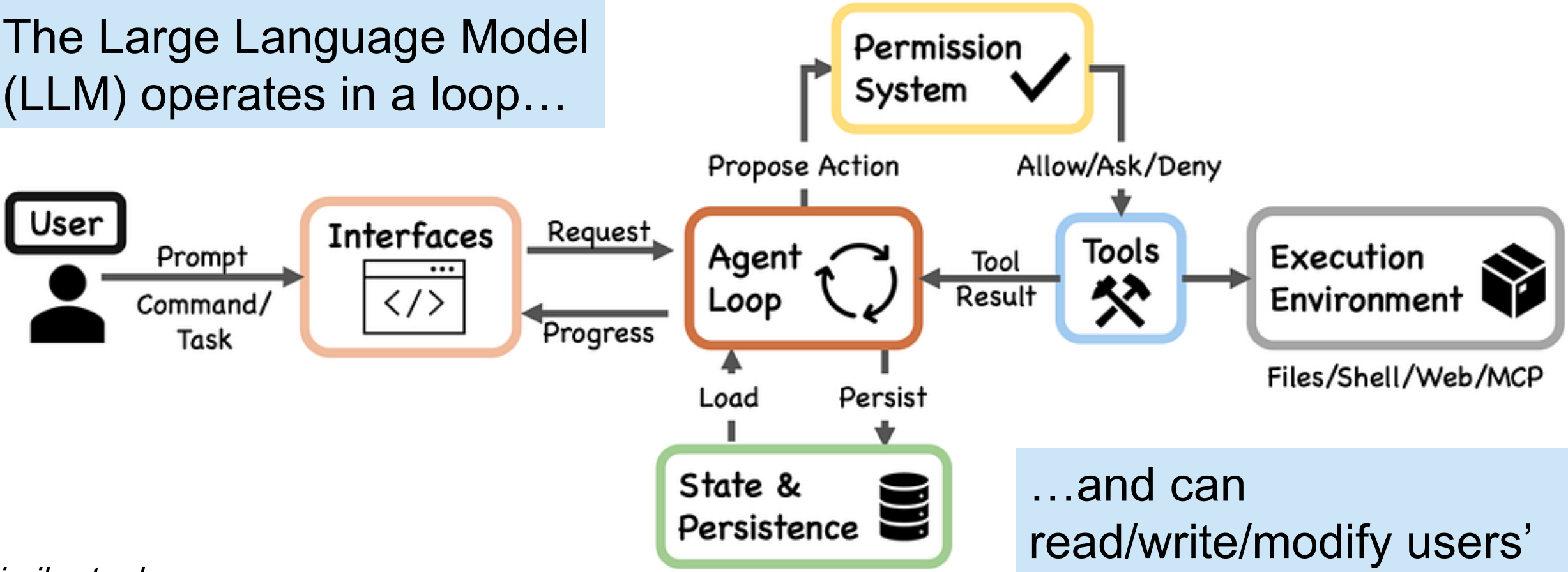
Other coupling of LLMs and RiverWare

- For demonstration purposes
- Not a part of any CADSWES software
- Local files are being read and analyzed by the LLM

https://github.com/jrkasprzyk/RiverWare-AI-Tools

The demonstration uses **Claude Code**, a "harness" connected to an LLM

The Large Language Model (LLM) operates in a loop…



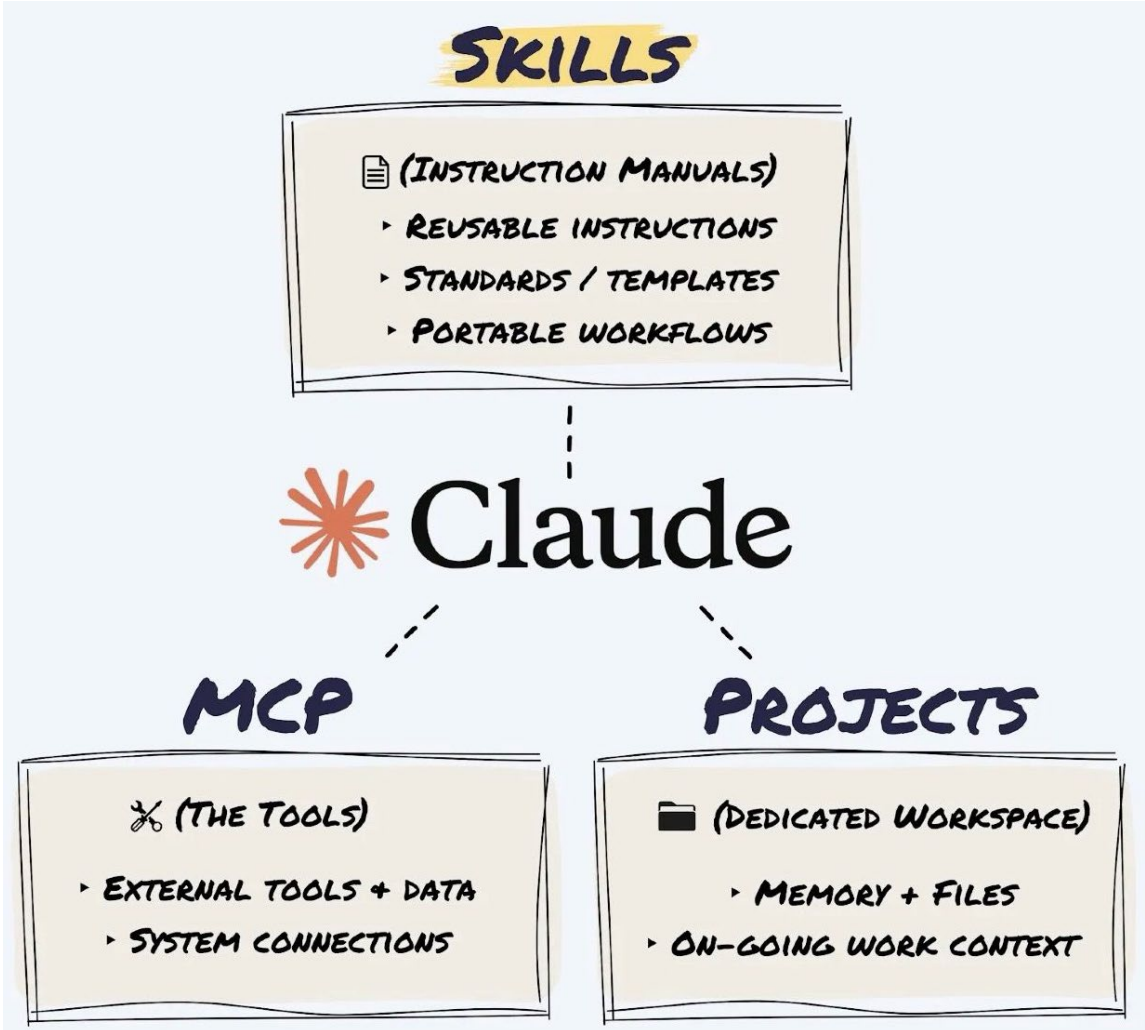
https://arxiv.org/pdf/2604.14228

…and can read/write/modify users' files only with permission

Similar tools:
VSCode + GitHub Copilot
Cursor

https://cobusgreyling.medium.com/98-of-claude-code-is-not-ai-bab2f37dee0e

We created **skills**: reusable instructions that aid the LLM



MCP: Model Context Protocol

Demos

Explain a RiverWare Model

Interactive Help

Write Draft of RPL rules

Create Dashboard

Demos

Explain a RiverWare Model

Interactive Help

Write Draft of RPL rules

Create Dashboard

/visualize-riverware-model

Skill instructions (excerpt)

Step 1 — generate the dashboard

`skills/visualize-riverware-model/digest_to_json.py` (Python 3.10+, stdlib only; it imports the parser from the explain skill, resolved from its own location, so it runs from any working directory).

This writes `<modelname>_dashboard.html` next to the model (`-o path` to choose the destination). Drop `--html` to get the raw JSON digest on stdout instead, which is useful when you want to inspect what the dashboard will show:

- `objects` — every simulation object with type and description
- `links` — object-to-object topology from the model's link definitions, oriented upstream → downstream and classified as water conveyance (`flow`) or data/head exchange (`data`)

Initial user prompt:

```
/riverware-ai-tools:visualize-riverware-model make a dashboard for the model found in [folder]
```

/visualize-riverware-model

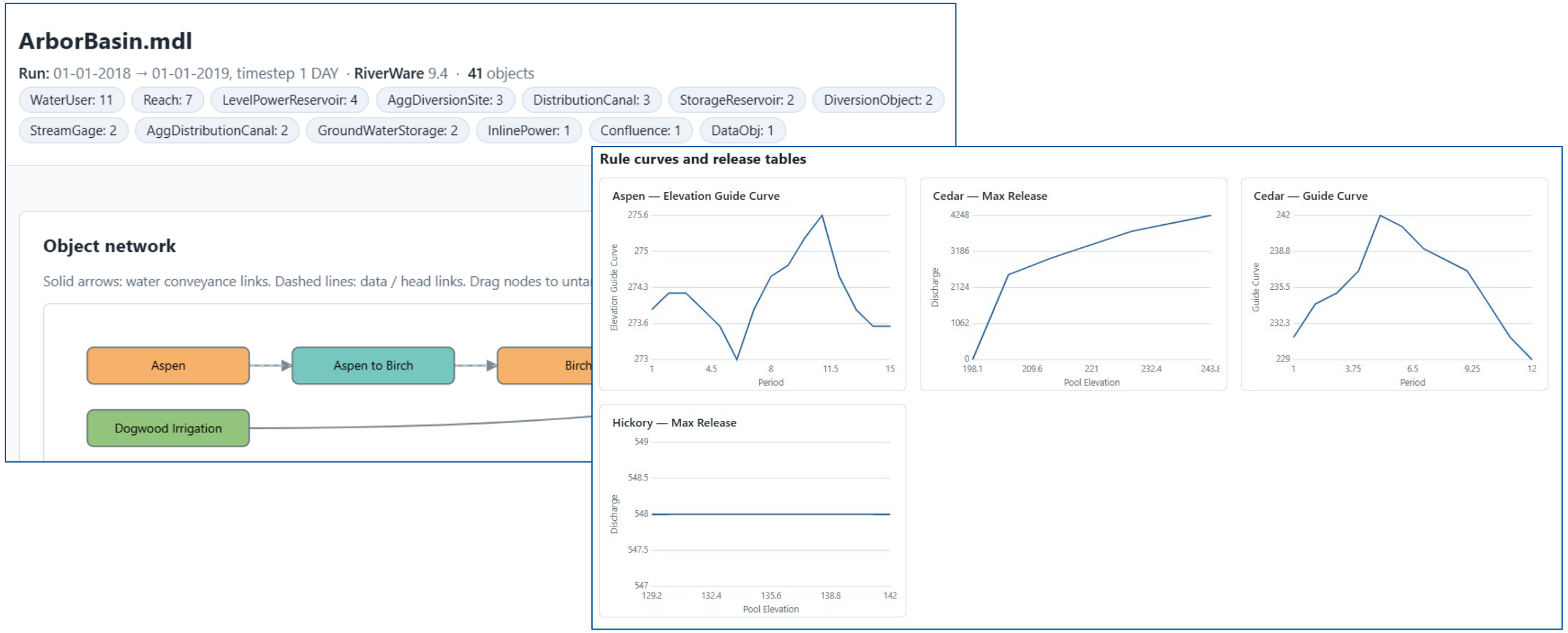
Claude is assessing its work and revising:

```
● Dashboard generated. But min value 2.0 on all three series looks suspicious — checking.

● Bash(python "C:/Users/joka0958/.claude/plugins/cache/riverware-ai-tools/riverware-ai-tools
/0.4.
      3/skills/visualize-riverware-model/digest_to_json.py" "C:/CADSWES/ugm-…)
  └ Res1 Outflow first5: [(1, 29.9344), (2, 31.9989), (3, 29.9344), (4, 92.1059), (5, 97.40
    4)] small vals: [(14, 2.0), (27, 2.0), (40, 2.0), (45, 4.0), (52, 9.81179), (53, 9.9938
    1), (64, 2.0), (77, 2.0), (102, 2.0), (110, 2.0)]
    … +7 lines (ctrl+o to expand)
```

/visualize-riverware-model

Dashboard loaded in browser:



https://jrkasprzyk.github.io/RiverWare-AI-Tools/examples/ArborBasin/ArborBasin_dashboard.html

Demos

Explain a RiverWare Model

Interactive Help

Write Draft of RPL rules

Create Dashboard

/explain-riverware-model

What's in the model file typically?

```
# installed as a plugin, run from anywhere
python "${CLAUDE_PLUGIN_ROOT}/skills/explain-riverware-
model/explain.py" path/to/your/model.mdl
```

Pass a `.mdl` alone, a `.rls` alone, or both together (order does not matter — the extension picks the parser). Add `--json` for machine-readable output.

What it pulls out:

- **From `.mdl`:** RiverWare version, run horizon + timestep, object count; every simulation object with its type, description, *selected* simulation methods (the `None`/`No Method` ones are dropped), and its slots grouped by slot type; the rule-curve / lookup tables (`TableSlot`, `PeriodicSlot`) with column labels, row count, and a 3-row sample; DMIs, embedded RPL sets, and model scripts.
- **From `.rls`:** (if stored in a separate file) ruleset name, agenda order, precision, description; every policy group and utility group with its active flag; every rule and function with its active flag, notes, and full RPL body.

If the output is long, redirect it to a temporary file of your choice and grep it:

```
python skills/explain-riverware-model/explain.py
examples/ArborBasin/ArborBasin.mdl > digest.txt
```

https://github.com/jrkasprzyk/RiverWare-AI-Tools/blob/main/examples/ArborBasin/ArborBasin_explained.md

Skill instructions (excerpt)

/explain-riverware-model

Overview

Arbor Basin is the CADSWES RiverWare training model: a two-basin river system with 41 simulation objects, run at a one-day timestep over calendar year 2018 (1 January 2018 through 1 January 2019). The model solves with rule-based simulation using an embedded ruleset, `Arbor Basin Rules`, and carries supporting initialization and MRM rule sets alongside it.

The system divides into a **west basin** — a five-reservoir stem running Aspen, Birch, Cedar, Dogwood, Elm — and an **east basin** centered on Hickory Reservoir, which supplies irrigation districts, an interstate minimum-flow point, and a canal-and-groundwater conjunctive-use complex. A Transbasin Diversion moves water from Cedar in the west basin to Hickory's inflow in the east, and it is the structural hinge of the model: how much the west basin can spare determines how much the east basin has to work with.

Physical network

West basin: the power cascade

Aspen, at the top of the watershed, is a level power reservoir (pool elevations roughly 273–282) with a 68-row Plant Power Table and a monthly Elevation Guide Curve holding the pool near elevation 274. Its Outflow passes through the Aspen to Birch reach (time-lag routing) into **Birch**, a second level power reservoir (elevations roughly 245–253). The two operate in

https://github.com/jrkasprzyk/RiverWare-AI-Tools/blob/main/examples/ArborBasin/ArborBasin_explained.md

Output (excerpt)

Demos

Explain a RiverWare Model

Interactive Help

Write Draft of RPL rules

Create Dashboard

/riverware-help

Prioritize official sources

Answer RiverWare questions the way a good help desk would: grounded in the official documentation, cited, and — when the question is about the user's own model — connected to what that model actually contains.

The authoritative source is the CADSWES online help, **CurrentVersion** (v9.6 at the time this skill was written):

```
https://www.riverware.org/HelpSystem/CurrentVersion/
```

It is plain static HTML with no authentication; WebFetch works on every page. Answers reflect CurrentVersion semantics. If the user says they run an older RiverWare, say so explicitly in the answer and note that behavior may differ (prior release notes exist under `PriorRelNotes.html`, but deep links into old releases are unreliable — do not chase them unless asked).

Source priority

1. **The user's model** — for contextual questions ("why does my rule not fire", "which method is my reservoir using"), evidence from their `.mdl` / `.rls` beats everything. Get it via the digest driver (`skills/explain-

Skill instructions (excerpt)

https://github.com/jrkasprzyk/RiverWare-AI-Tools/blob/main/examples/sessions/riverware-help_rule-overwrite.md

/riverware-help

Question. I added a rule at the bottom of the `Roberto Rules` group. The Rule Log shows it firing and computing a release, but the final `Roberto.Release` value came from `Flood Control` instead. Why does my rule's value not stick?

Answer.

Your rule is working — it is being **overwritten on purpose**, and the mechanism is rule priority. Per the CurrentVersion help:

Slot Priorities and Flags — "Slots that are set by rules have the priority of the rule that set them," and "Slots which are directly set by a rule are displayed with a special flag, the R flag." Priority 0 is user input: "These values may never be overwritten during a run."

Resetting Slot Values — a proposed R-flagged value overwrites an existing R-flagged value only at **higher or equal priority**. The page's example table: existing `6R` overwritten by proposed `4R` → successful; existing `4R` and proposed `6R` → unsuccessful. (Smaller number = higher priority; rule 1 is the highest-priority rule in the set.)

Your model: the ruleset uses `ASCENDING` agenda order, so the *bottom* rule of the listing fires *first* and has the *lowest* priority. The `Roberto Rules` listing, top to bottom, is:

1. `Prevent Overtopping`
2. `Avoid City Flooding`

The user is encouraged to read the documentation

https://github.com/jrkasprzyk/RiverWare-AI-Tools/blob/main/examples/sessions/riverware-help_rule-overwrite.md

Example results of a session

/riverware-help

Some design guidelines to ponder:

Skill behavior	Where it shows up
"Why is my model doing Y" gets reasoning, not a lookup	The answer reconstructs the priority mechanics end to end
Cite the page, quote the semantics	Two URLs, exact sentences on R flags and the overwrite table
Model evidence beats general knowledge	The actual `Roberto Rules` listing and `ASCENDING` order come from the embedded ruleset
Hand off, don't drift	Ends by pointing at draft-riverware-rules instead of rewriting the rule mid-answer

https://github.com/jrkasprzyk/RiverWare-AI-Tools/blob/main/examples/sessions/riverware-help_rule-overwrite.md

Demos

Explain a RiverWare Model

Interactive Help

Write Draft of RPL rules

Create Dashboard

/draft-riverware-rules

Skill instructions (excerpt):

This skill turns a plain-language policy request ("add a spring flood-control drawdown for Cedar") into a pasteable RPL rule that fits the target model: it references only slots that exist, matches the ruleset's existing idioms, and comes with an explicit statement of where it belongs in the agenda.

The deliverable is a **draft for a modeler to review and load** — never a validated rule. Only RiverWare loading and running the ruleset validates RPL.

https://github.com/jrkasprzyk/RiverWare-AI-Tools/blob/main/examples/ArborBasin/ArborBasin_rule_case_study.md

/draft-riverware-rules

The request supplies no slot for the drawdown offset, so the draft uses a literal `2 "m"` below the Guide Curve. If the offset should be tunable (for example, by an external workflow through the input DMI), Cedar needs a new scalar slot for it first — drafting alone cannot create one.

The draft

```
RULE "Cedar Spring Flood Drawdown";
DESCRIPTION "March-May: hold Cedar 2 m below the Guide Curve to preserve
flood space for snowmelt.";
ACTIVE TRUE;
BEGIN

  $ "Cedar.Outflow" [] := IF ( "GetMonth"( @"t" ) >= 3.00000000 AND
                               "GetMonth"( @"t" ) <= 5.00000000 AND
                               $ "Cedar.Pool Elevation" [] >
                                 $ "Cedar.Guide Curve" [] - 2.00000000 "m" )
 THEN
  "Min"( "SolveOutflow"( % "Cedar", $ "Cedar.Inflow" [],
                         "ElevationToStorage"( % "Cedar",
                             $ "Cedar.Guide Curve" [] - 2.00000000 "m" ),
                         $ "Cedar.Storage" [@"t - 1"], @"t" ),
         "GetMaxOutflowGivenInflow"( % "Cedar", $ "Cedar.Inflow" [], @"t" ) )
 ENDIF;

END
```

https://github.com/jrkasprzyk/RiverWare-AI-Tools/blob/main/examples/ArborBasin/ArborBasin_rule_case_study.md

To try yourself:

In Claude Code:

First, add the marketplace:

```
/plugin marketplace add jrkasprzyk/RiverWare-AI-Tools
```

then from the marketplace, install the plugin:

```
/plugin install riverware-ai-tools@riverware-ai-tools
```

For other tools and full instructions, see:
https://github.com/jrkasprzyk/RiverWare-AI-Tools/blob/main/README.md