https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

Can We Trust LLMs for Complex Earth System Model Analysis? Silent Failure and Evidence from Module-Grounded Benchmarking

Tian Zhou[1], Yun Qian[1], and L. Ruby Leung[1]
[1]Pacific Northwest National Laboratory, Richland, WA, USA
Correspondence: Tian Zhou (tian.zhou@pnnl.gov)

Abstract.

Large language models (LLMs) are becoming increasingly capable of complex scientific scripting, but this growing robustness creates a paradox: the more trustworthy their outputs appear, the more easily scientifically incorrect results can pass unnoticed. In Earth system model (ESM) analysis, such silent failures are more dangerous than visible crashes because they produce plausible figures and statistics that may be accepted without detailed inspection. We address this risk with ESFlow, a module-grounded agentic AI framework that constrains the LLM to compose workflows from validated analysis tools rather than generate arbitrary code. The LLM reads an auto-generated, self-describing catalog and outputs a YAML (human-readable data-serialization) workflow, which is then executed by a deterministic engine. We demonstrate this framework with a validated tool library for Energy Exascale Earth System Model (E3SM) land surface hydrology diagnostics in a benchmark spanning seven analysis tasks and six contemporary LLMs. Across both single-attempt runs and runs augmented with automatic self-debugging, the module-grounded approach attains an overall success rate above 80 %, maintains a low and stable silent-failure rate, and reaches 100 % success for the three high-capability models, whereas unconstrained Python code generation succeeds in only about 5 % of runs and sees its silent-failure rate rise from roughly 16 % to about 40 % under self-debugging. These results suggest that increasing LLM capability does not remove the reliability problem in scientific scripting; it makes silent failures more consequential by making incorrect outputs more convincing. The answer to the trust question posed in the title is therefore conditional: unconstrained code generation is not trustworthy for complex ESM analysis, whereas module-grounded workflow composition can be highly reliable for frontier models and remains substantially more robust under iterative self-debugging. By shifting the LLM's role from code generation to the composition of trusted tools, this framework provides a safer, more scalable architecture for AI-assisted scientific discovery that is aligned with FAIR (findable, accessible, interoperable, and reusable) principles.

1 Introduction

Large language models (LLMs) are increasingly being used as agentic AI systems capable of reasoning, planning, and executing multi-step tasks. In Earth system science, these LLM-based agents are beginning to automate analysis workflows that traditionally required weeks of expert scripting. Systems such as EarthLink (Guo et al., 2025) show how LLMs can now plan and execute climate-analysis pipelines by generating and running code on the fly. Geo-OLM (Stamoulis and Marculescu, 2025)

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

couples cost-effective open language models with state-driven workflows for geospatial analysis. PANGAEA GPT (Pantiukhin et al., 2025) showcases multi-agent orchestration for complex environmental data retrieval. By accelerating exploratory analysis and lowering the technical barrier to entry for non-programmers, these systems represent substantial milestones in scientific accessibility.

However, deploying agentic AI in production science introduces a fundamental tension between autonomous capability, apparent robustness, and methodological integrity. This tension arises from the irreconcilable difference between the probabilistic generation of LLMs and the deterministic rigor required by scientific inquiries. While frameworks like EarthLink can version-control their outputs, the methodological logic within those scripts is generated anew for every request, depending on random seed variations or subtle prompt phrasing. As LLMs become better at completing long, multi-step tasks, users are also more likely to trust them, which increases exposure to the most dangerous risk in scientific computing: **silent failure**. Unlike syntax errors, which fail visibly, algorithmic hallucinations — such as mixing up key variables with similar names, selecting an inappropriate regridding scheme, or ignoring leap years — often run to completion without warning. Their outputs may look entirely reasonable, making them more dangerous than crashes because success at the software level can conceal failure at the scientific level. Detecting these silent failures requires the user to carefully inspect AI-generated code, effectively canceling the efficiency gains of automation (Vangala et al., 2025; Siddiq et al., 2025).

Meanwhile, the Earth system model (ESM) community has invested heavily in diagnostic toolkits designed to support reproducible, version-controlled analysis. Packages like E3SM Diagnostics (Zhang et al., 2022), the PCMDI Metrics Package (PMP; Lee et al., 2024), ESMValTool (Righi et al., 2020; Eyring et al., 2020) and the International Land Model Benchmarking (ILAMB) system (ILAMB; Collier et al., 2018) are intended to produce deterministic results but require extensive domain expertise to configure and extend. The gap is therefore clear: LLM agents offer accessibility and flexibility, whereas traditional diagnostic packages offer methodological control.

There is therefore a need for an approach that preserves the accessibility of LLM-based agents while imposing methodological guardrails on scientific analysis. We address this need with ESFlow (**E**arth **S**ystem **Flow**), a module-grounded framework in which the LLM does not write analysis code but instead composes declarative workflows from pre-validated tools. This design shifts the locus of correctness from freshly generated scripts to expert-maintained analysis components, making both execution and provenance reproducible by construction. We evaluate this methodology using a 24-tool library for Energy Exascale Earth System Model (E3SM; Golaz et al., 2022) land surface hydrology analysis, a seven-task benchmark across six LLMs, and an additional iterative self-debug experiment that tests how failures evolve under repeated repair.

2 Approach

2.1 ESFlow, a module-grounded analysis framework

To answer the trust question posed in the title, we developed ESFlow, a framework that constrains what the LLM is allowed to decide during scientific analysis. It separates workflow composition from method implementation. The LLM is allowed to only select validated operations and specify how their inputs and outputs are connected, but the analytical procedures themselves

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

remain fixed in expert-maintained tools. The framework is implemented in Python with minimal dependencies (e.g. NumPy, xarray, pandas), and Fig. 1 illustrates how this constraint changes the end-to-end analysis process.

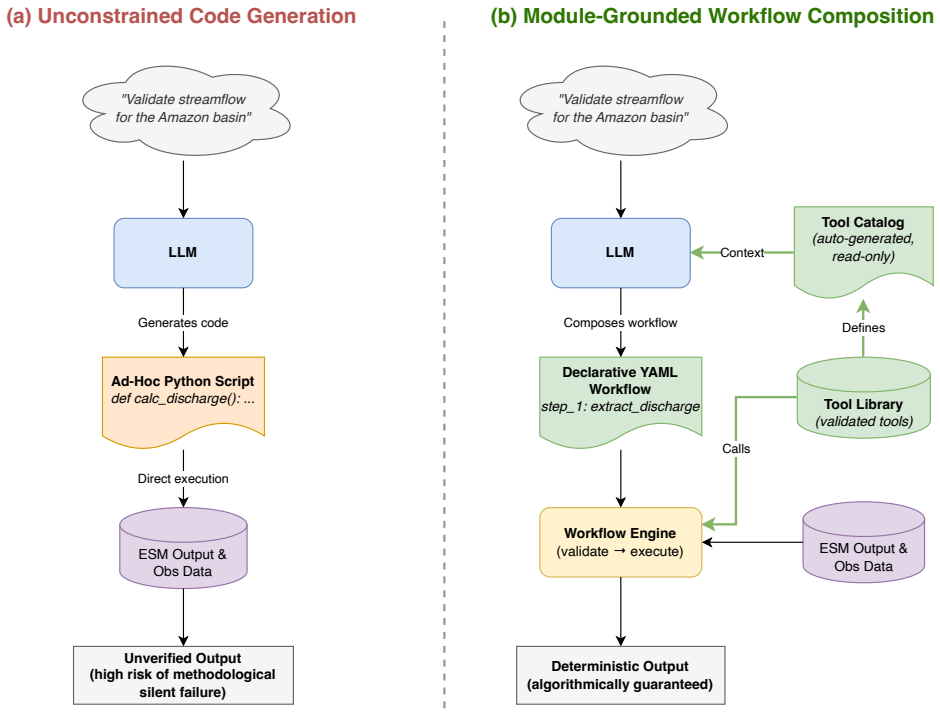

Figure 1. Architectural comparison between unconstrained LLM code generation and ESFlow, the proposed module-grounded framework. (a) Unconstrained code generation: the LLM writes an ad-hoc Python script that directly accesses data, producing unverified output with high risk of silent failure. (b) Module-grounded workflow composition: the LLM reads an auto-generated tool catalog and composes a declarative YAML workflow; a deterministic engine connects each step to a validated analysis tool from the tool library, producing traceable output with recorded provenance.

2.1.1 Architectural design

The key architectural element is the validated tool interface that is exposed to the LLM. Each analysis tool is an ordinary Python function augmented with a small block of structured metadata (a `ToolSpec`) that declares its name, category, input

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

parameters (with defaults and allowed values), and outputs. A catalog generator discovers all registered tools and compiles them into a single machine-readable YAML file, the tool catalog. This catalog is the only tool-definition document the LLM reads. When a researcher issues a natural-language prompt, the LLM receives the catalog as additional context and returns a YAML workflow that references tools by name and wires their parameters together (Fig. 1).

This design has three major features. First, any LLM with sufficient context length can use the tools without fine-tuning — the catalog is the universal interface, so the approach is LLM-agnostic by construction. Second, domain correctness is concentrated in the tools and maintained by domain experts, not distributed across LLM-generated scripts. Third, the cost of onboarding an existing script is minimal (a metadata block of ∼10–15 lines), which means the framework could be built upon a researcher's existing workflow rather than replacing it.

After execution, the engine writes a copy of the workflow augmented with provenance metadata — the git commit hash, a Coordinated Universal Time (UTC) timestamp, and the resolved output directory — so the exact analysis can be reproduced, inspected, or modified by anyone with access to the tool library, without an LLM. This aligns with FAIR data principles (Wilkinson et al., 2016): the workflow is findable, accessible, interoperable, and reusable as a standalone record of the analysis.

2.1.2 Validated tool library

We demonstrate the framework with a 24-tool library designed for E3SM land and river analysis. The tools are organised into six categories (Appendix A): *fetchers* that download observation data from public servers, *loaders* that read and validate local files, *matchers* that perform spatial correspondence between observation locations and model grids, *extractors* that subset model output into analysis-ready formats, *analyzers* that compute statistics and validation metrics, and *plotters* that produce publication-ready figures. Together, the 24 tools expose 92 parameters — an average of fewer than four per tool — keeping the decision space manageable. The underlying implementations total approximately 5 000 lines of Python, meaning a typical 15-line YAML workflow orchestrates hundreds to thousands of lines of validated code that the LLM never needs to generate or reason about.

Several tools encode domain-specific conventions that would be error-prone if left to ad-hoc scripting. For example, the tool `extract_gridded_field` accepts composite variable expressions (e.g., `QVEGE+QVEGT+QSOIL` for the land model's total evapotranspiration). A dedicated fetcher, `fetch_ilamb_data`, downloads gridded observation datasets from the ILAMB data server (Collier et al., 2018) at runtime and caches them locally, so that workflows can be reproduced on any machine without pre-staged observation files. The tool library is designed to be extensible. Adding a new tool requires writing a Python function, adding a `ToolSpec` decorator (Appendix B), and re-running the catalog generator. This means researchers can personalise the library for their own analysis needs — adding tools for atmosphere, ocean, or sea-ice diagnostics, for instance — without modifying the framework itself.

2.1.3 Workflow execution

The workflow engine closes the loop between the LLM's declarative output and deterministic execution. It parses the YAML workflow generated by the LLM, validates each step against the tool catalog, resolves inter-step references, and executes steps

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

sequentially. This validation step is architecturally significant: a workflow either passes schema validation and runs through validated tools with predefined behavior, or fails before execution. If a step fails at runtime, the engine skips all downstream steps that depend on its outputs, so failures propagate visibly rather than producing silently incomplete results. The engine writes all outputs to a structured directory and supports selective re-execution and output caching for iterative development. The framework also includes a Python script that converts a YAML workflow into a workflow diagram for visual inspection. Figure 2 shows an example for a 13-step task that investigates basin-level LND water budget residuals, which requires the workflow to extract the precipitation, runoff, and evapotranspiration fields from the standard E3SM output, aggregate them to basin means, compute the water balance residual, and produce a composite figure with a global map of the residual and per-basin bar charts. Figure 3 shows the resulting output: a global residual map with basin outlines and per-basin breakdowns of precipitation, evapotranspiration, and runoff.

2.2 Benchmark design

2.2.1 Task suite

We evaluate the framework through a benchmark suite of seven tasks (Table 1), tested across six LLMs and compared against a direct code-generation baseline. These tasks probe qualitatively different analytical challenges — from simple tool selection (T1, 3 steps) to multi-source basin water-cycle evaluation (T7, 23 steps) — rather than simply scaling the number of variables. All basin-scale tasks use the same six gauges spanning diverse climates (Amazon, Missouri, Columbia, Danube, Mekong, Orange), so complexity scales with workflow structure rather than domain unfamiliarity. Each task produces deterministic numerical outputs for automated comparison against hand-crafted reference values; complete reference outputs are provided in the Supplement. The sample dataset used in the benchmark comprises five years (1985–1989) of monthly river (ROF) and land (LND) output from an E3SM v3 historical simulation, streamflow gauge observations and basin polygons from the Global Runoff Data Centre (GRDC; GRDC, 2024), and gridded observation products — MODIS evapotranspiration, GPCC precipitation, and LORA runoff — retrieved automatically at runtime from the ILAMB data server (Collier et al., 2018) by the framework's dedicated fetcher tools.

2.2.2 Experimental setup

We test the benchmark across six LLMs spanning three capability tiers: high-capability (Claude Opus 4.6, GPT-5), mid-tier (Gemini 2.5 Flash, o4-mini), and compact/open (Claude Haiku 4.5, Phi-4). Each task–model combination is run four times at the lowest temperature each API permits (`temperature=0` for standard models, `temperature=1` for reasoning models). Claude and GPT models are accessed through Pacific Northwest National Laboratory (PNNL)'s AI Incubator API, Gemini through the Google AI API, and Phi-4 is run locally via LM Studio.

Each LLM is tested under two conditions. In both conditions the LLM receives a prompt comprising two parts: a system message that provides domain context, and a user message containing the natural-language task description. The task descriptions are identical across conditions; the two conditions differ only in their system messages and expected output format. In the

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

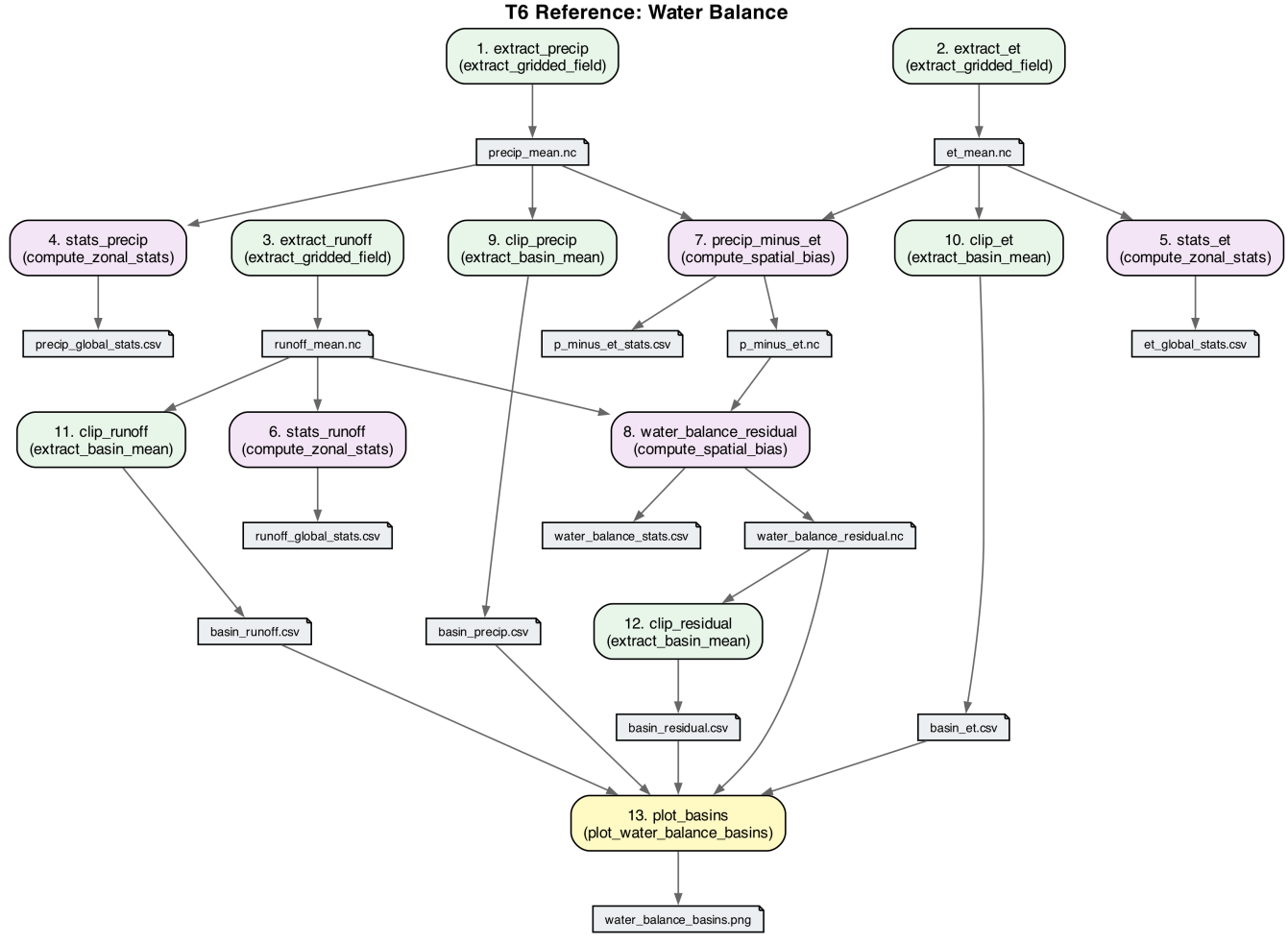

Figure 2. Workflow diagram for the 13-step water balance task (benchmark task T6), automatically generated from the YAML workflow file. Each node shows the step name and the tool it invokes (in parentheses); arrows trace data dependencies through named outputs.

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

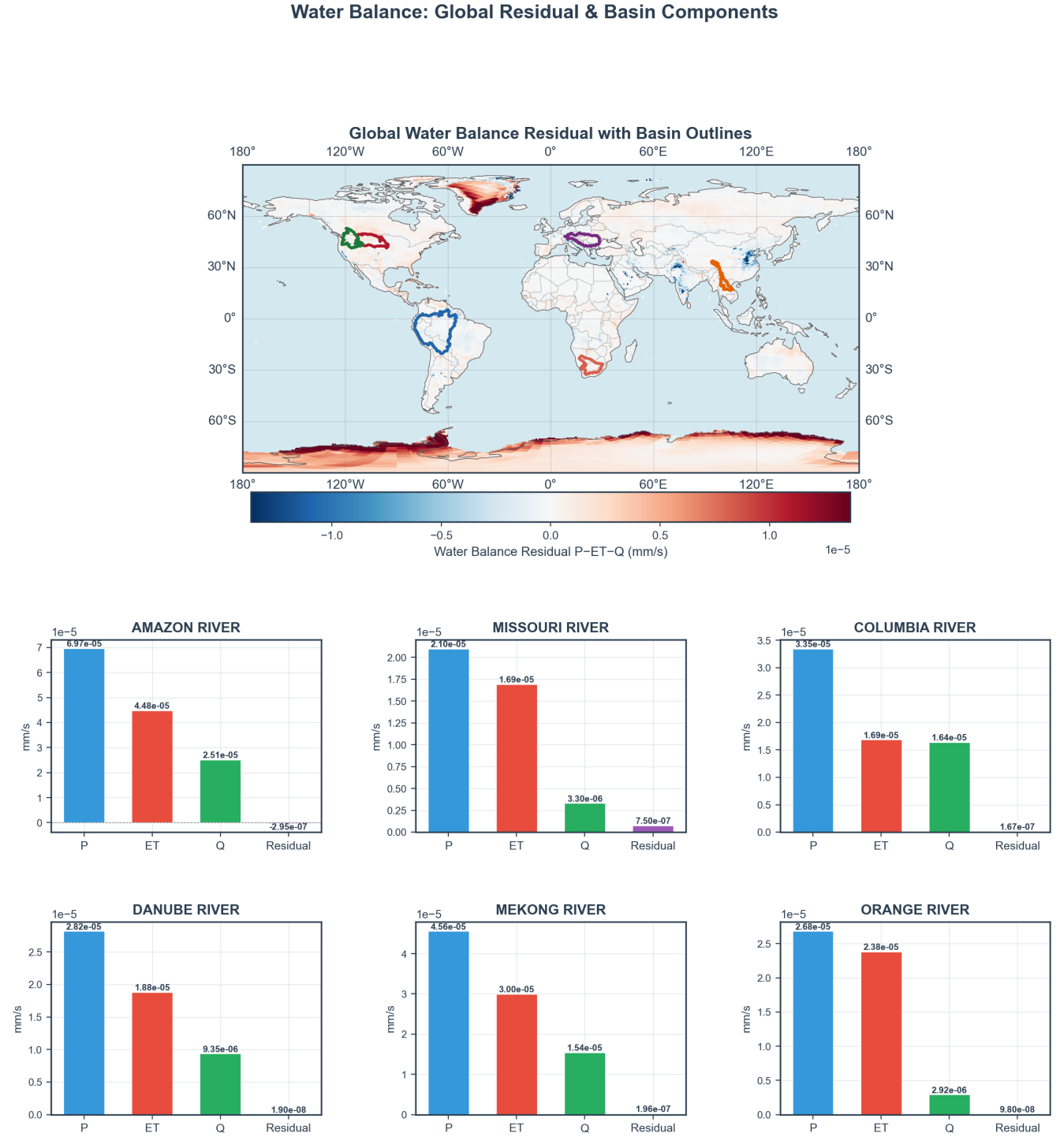

Figure 3. Output of the T6 water balance workflow (Fig. 2): a global water balance residual map ($P - \mathrm{ET} - Q$) with basin outlines, and per-basin bar charts of precipitation, evapotranspiration, runoff, and residual components for six basins spanning diverse climates.

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

Table 1. Benchmark task suite, with tasks ordered by increasing analytical complexity.

Task	Name	Task description	Steps	Key challenge
T1	Observation summary	"Load gauge metadata, extract 1985–1989 streamflow for six named gauges (Amazon, Missouri, Columbia, Danube, Mekong, Orange), compute summary statistics."	3	Tool selection, basic parameterisation
T2	Global runoff map	"Extract ELM QRUNOFF for 1985–1989, compute area-weighted global mean, and produce a map with statistics overlay."	3	Gridded extraction, zonal statistics, map with annotation
T3	ET benchmark	"Fetch MODIS ET from ILAMB, extract 5-year model ET as QVEGE+QVEGT+QSOIL, compute spatial bias and map."	5	Automated data retrieval, composite variable, spatial bias
T4	Basin streamflow	"Match gauges to MOSART grid, extract sim and obs discharge for 1985–1989, compute validation metrics (RMSE, NSE, KGE), and plot basin boundary maps with sim-vs-obs time series for six basins."	6	Basin polygons (GeoJSON), validation metrics, geospatial visualisation
T5	Streamflow FDC	"Match gauges to MOSART grid, extract sim and obs discharge for 1985–1989, compute flow duration curve (FDC) metrics (Wasserstein distance, volume bias, quantile ratios) for all matched gauges, and plot FDC comparisons for six named rivers."	6	Model–obs matching, distributional metrics, FDC plot
T6	Water balance	"Extract climatological P, ET, and QRUNOFF for 1985–1989; compute global means and water balance residual; clip P, ET, Q, and residual to six basins; produce a composite figure with the global residual map and per-basin bar charts."	13	Basin clipping, composite variables, chained residual, multi-panel figure
T7	Basin water cycle evaluation	"For six basins, extract model P/ET/Q and obs P (GPCC)/ET (MODIS)/runoff (LORA), clip to basin means via GeoJSON polygons, compute streamflow FDC metrics, combine into per-basin water budget, and plot model-vs-obs bar charts and radar charts diagnosing bias sources."	23	Basin clipping, four obs sources, unit conversion, budget synthesis, dual visualisation

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

module-grounded condition, the system message contains the full tool catalog, and the LLM generates a YAML workflow. In the code-generation baseline, the system message instead provides dataset paths, variable names, grid structure, and observation metadata — the same domain information encoded in the tool catalog — and the LLM writes a standalone Python script. Both conditions are single-shot with no framework-specific instructions, so the benchmark tests the catalog's self-descriptiveness rather than prompt engineering. The full system messages, task prompts, and prompt templates are provided in the Supplement.

Iterative self-debugging — where the model receives its own error traceback and attempts a fix — is now a standard feature of agentic coding workflows. To assess how the two conditions respond to this common practice, we add a self-debugging test to every run that crashes in the single-shot condition. The captured traceback is returned to the same model together with the original prompt and the failing artifact (YAML workflow or Python script), and the model produces a corrected version. If the corrected artifact crashes again, the new traceback is fed back for up to three rounds. Runs that did not crash are carried over unchanged, since there is no error signal to drive self-repair. Recovered outputs are then graded alongside the single-shot results.

2.2.3 Evaluation criteria

For each task, we produce a reference run: a hand-crafted workflow whose outputs have been manually inspected and approved as ground truth. Every LLM-generated run — under both conditions — is compared against this reference and assigned one of four grades, ranked by consequence:

Success: The most downstream numerical files (e.g. NetCDF fields or CSV statistics) are bit-for-bit identical to the reference and all expected diagnostic plots are present. Note that plots may still differ in cosmetic details (e.g. annotation placement, axis labels, color choices) as these are controlled by optional parameters that the LLM specifies independently for each run.

Silent failure: The workflow or script executes without error and produces plausible, well-formatted output, but numerical values differ from the reference — the most dangerous category because the results appear credible.

Obvious failure: Execution completes and produces the expected deliverables, but the output is visibly wrong (e.g., missing plot elements, incorrect units, absurd values such as central U.S. precipitation comparable to the Amazon, or wrong variable types) — any domain scientist would easily reject it.

Crash: A runtime error terminates execution before completion, or the run completes without producing all required final deliverables (e.g., a missing key figure or output file).

Figures 4 and 5 illustrate the distinction between silent and obvious failures using T6 (water balance) as an example. The reference output for this task is shown in Fig. 3. GPT-5 baseline run 3 (Fig. 4) produces a polished composite figure with the correct layout and basin-mean values within 1–2 % of the reference — the script correctly locates the model's native area variable but omits land-fraction weighting, so coastal grid cells contribute ocean fractions where water balance variables are near zero. Without a known reference, this output would pass visual review. Haiku baseline run 3 (Fig. 5) encounters a polygon-clipping

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

error during basin extraction: 5 of 6 basins fall back to identical default values (P=1.73, ET=1.05, Q=0.57 mm day^{-1}), which any domain scientist would immediately reject — Amazon precipitation cannot equal Orange precipitation.

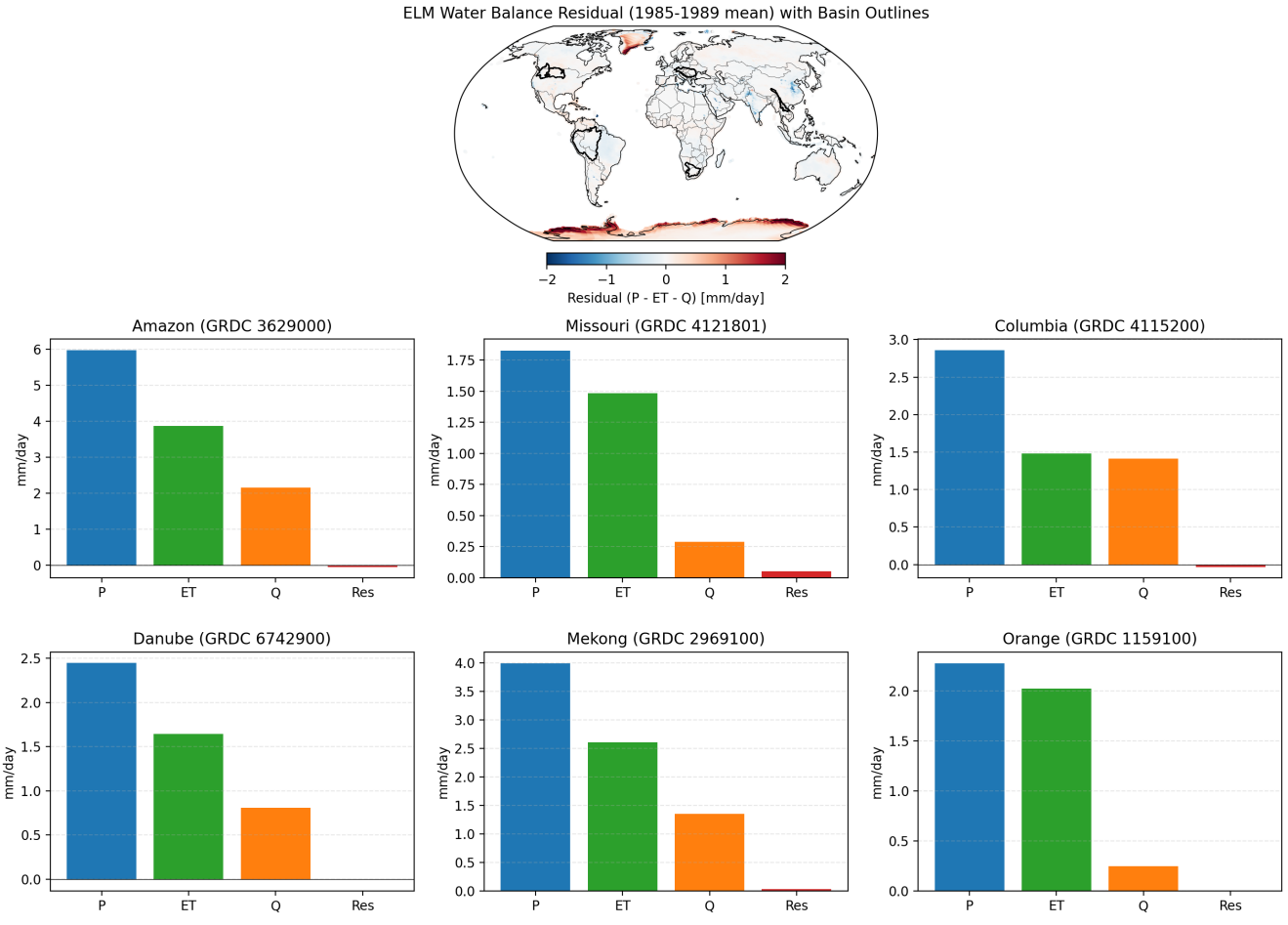

Figure 4. Silent failure in T6 (water balance): GPT-5 baseline run 3. The composite figure has the correct layout and basin-mean values within 1–2 % of the reference (Fig. 3). The script omits land-fraction weighting, so coastal cells contribute their full area weight including ocean fractions, systematically pulling basin means toward zero. Without a known reference, this output would pass visual review.

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

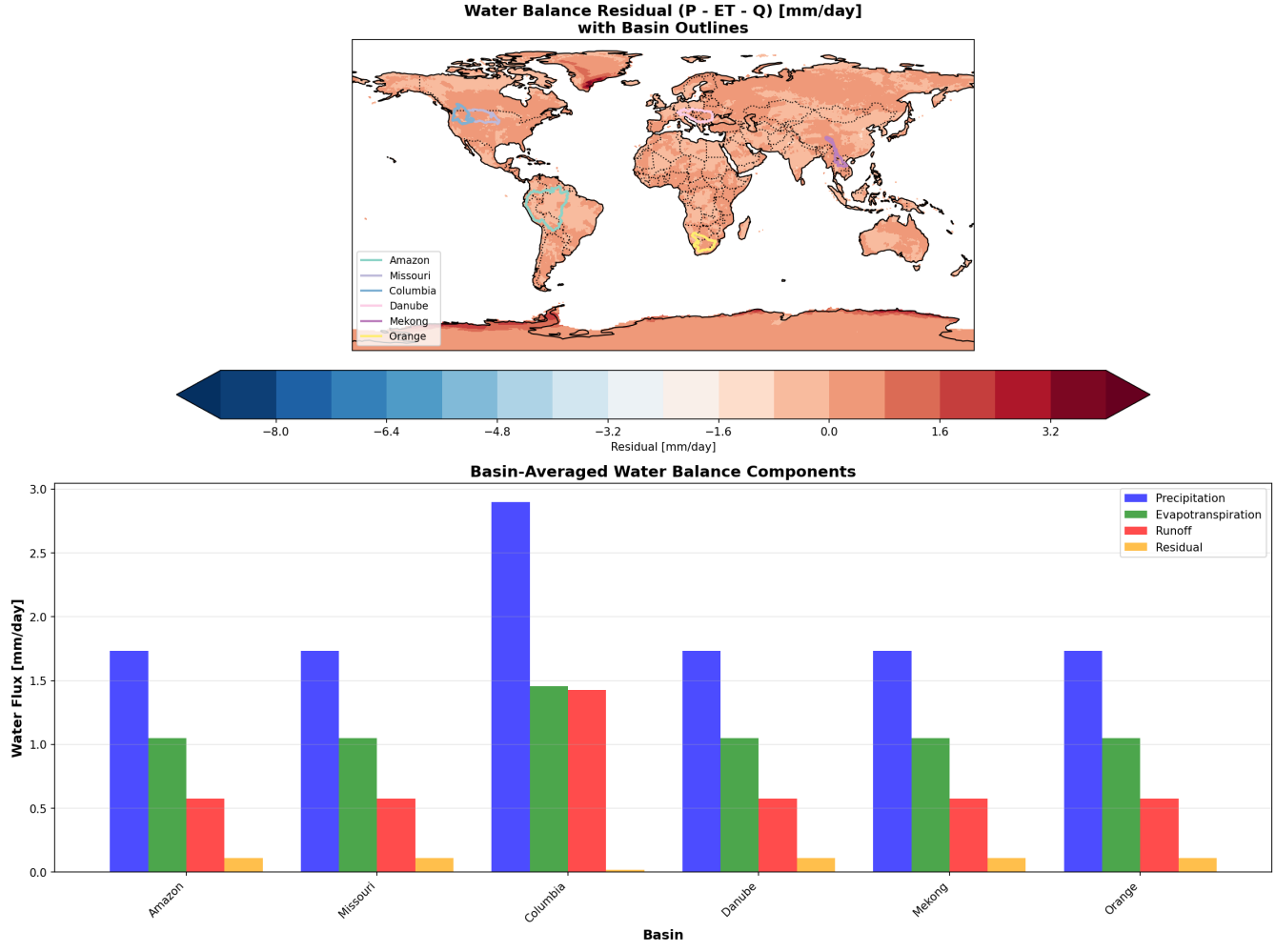

Figure 5. Obvious failure in T6 (water balance): Haiku baseline run 3. A polygon-clipping error during basin extraction causes 5 of 6 basins to return identical fallback values (P=1.73, ET=1.05, Q=0.57 mm day^{-1}), visible as uniform bar heights across basins with diverse climates. Only Columbia, whose polygon geometry is clean enough for clipping, produces distinct values. Compare with Fig. 3 (reference) and Fig. 4 (silent failure).

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

3 Results

3.1 Single-shot results

Under the module-grounded condition (Fig. 6a), 137 of 168 single-shot runs (82 %) produced bit-for-bit identical results to the reference, with zero obvious failures. The remaining runs are 8 silent failures (5 %) and 23 crashes (14 %). Performance follows a clear gradient across both task complexity and model capability. The three strongest models — Claude Opus 4.6, GPT-5, and Gemini 2.5 Flash — together achieve 83 of 84 successful runs (99 %), with the single non-success being a crash in Opus T7 discussed below. All 8 silent failures in the module-grounded condition come from a single model: o4-mini, a small reasoning-focused model. Haiku and Phi-4 handle simpler tasks (T1–T5) well but crash on the most complex tasks (T6–T7), where the number of required steps (13–23) exceeds their compositional capacity.

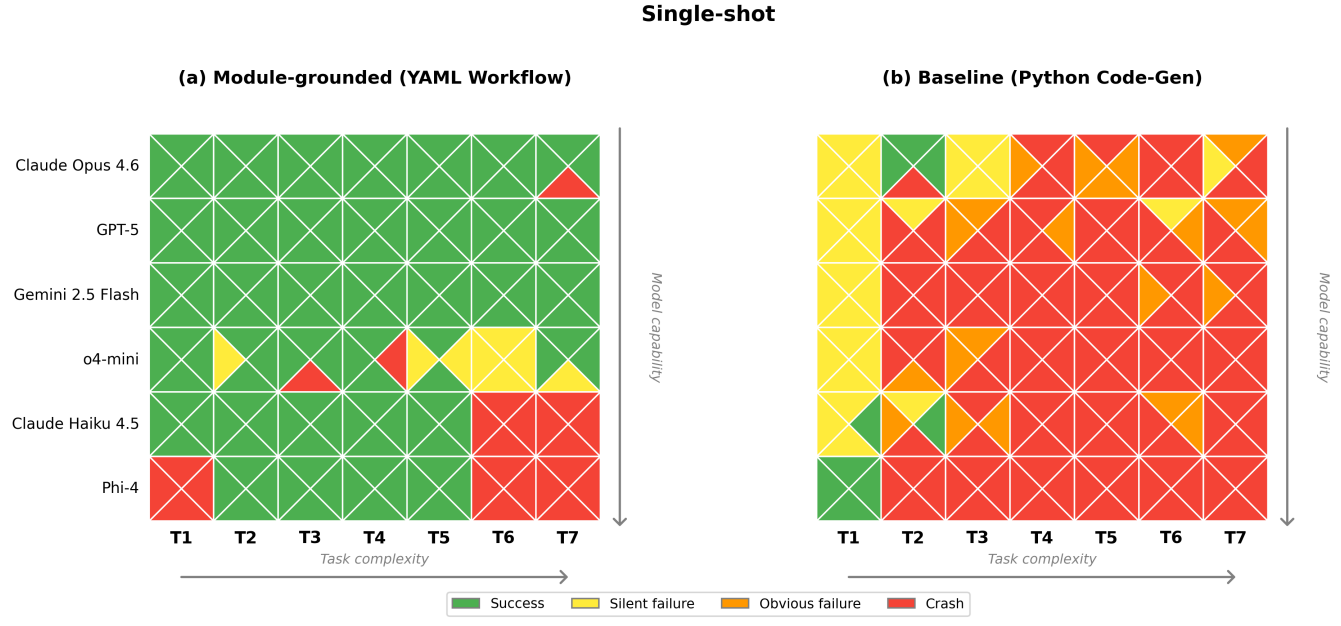

Figure 6. Single-shot benchmark results for six LLMs across seven tasks, comparing the module-grounded approach (a) against the code-generation baseline (b). Each cell is divided into four triangles representing four independent runs, color-coded by outcome: green = success (bit-for-bit match to reference), yellow = silent failure (plausible but numerically incorrect), orange = obvious failure (clearly incorrect output), red = crash (final deliverable missing). Models are ordered by capability tier (top to bottom); tasks by complexity (left to right).

The code-generation baseline (Fig. 6b) exhibits the same complexity–capability gradient but at dramatically worse levels: only 9 of 168 runs (5 %) succeed, while 111 runs (66 %) crash outright and 21 (13 %) produce obviously incorrect output. The crashes share common root causes: calendar-type mismatches in E3SM data, array shape errors from incorrectly read fields, incorrect variable or file path references, and wrong library imports. The obvious failures typically produce deliverables that exist but are functionally useless — obs-only time series for a sim-vs-obs comparison, blank maps, or NaN-filled CSV metrics.

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

Most concerning are the 27 silent failures in the baseline (16 %): these generate plausible output with correct formatting but numerically incorrect values, often from methodological choices that appear reasonable (e.g., bilinear regridding instead of conservative remapping, omitted land-fraction weighting, or inconsistent unit conversions across model and observational datasets). Such failures are dangerous precisely because users may build trust in the LLM after seeing well-formatted output from simpler tasks, then fail to verify the more complex results where silent errors are most likely. Cross-model visual outputs for all seven tasks, along with per-task failure catalogs, are provided in the Supplement.

3.2 Effect of iterative self-debugging

In practice, an analyst who receives a clear crash message would not simply discard the run — the LLM would be asked to fix it. To bring the benchmark closer to this realistic workflow, we extend the single-shot experiment by applying up to three rounds of self-debugging to every crashed run (Fig. 7). Note that runs graded as obvious failures are not re-run because they produce no error signal to trigger repair.

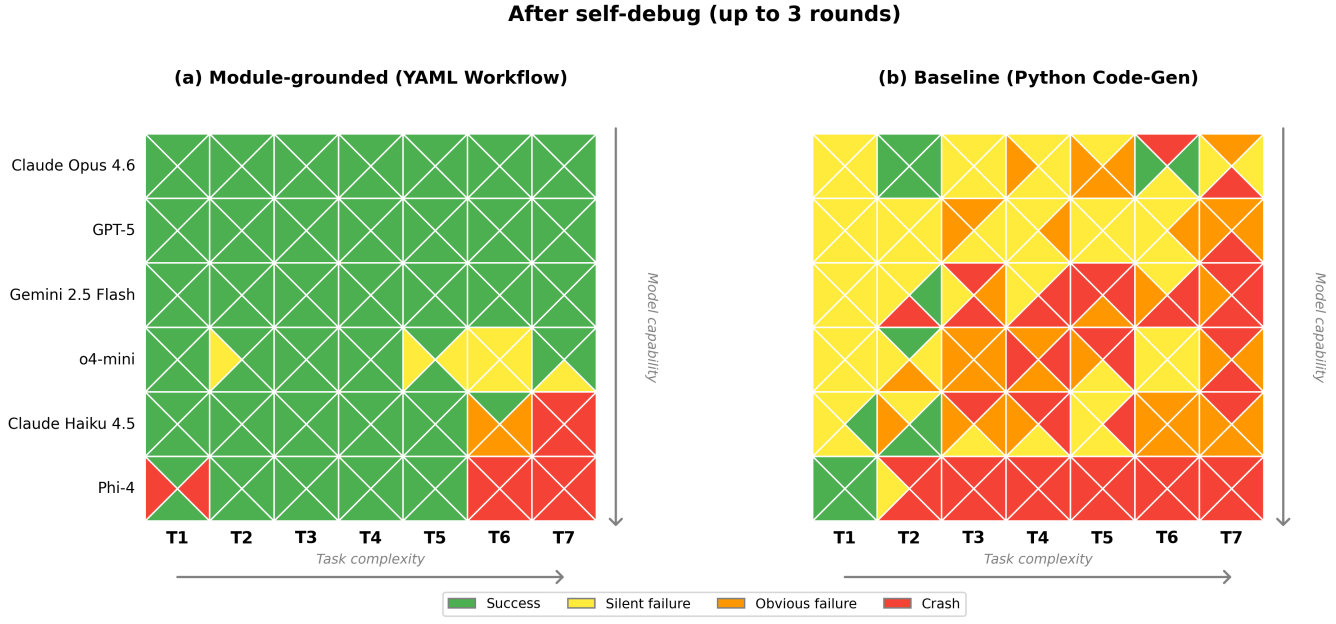

Figure 7. Benchmark results after applying up to three rounds of iterative self-debugging to every single-shot crash. Layout and color coding match Fig. 6.

Under the module-grounded condition, crash counts fall from 23 to 14; the 9 recovered runs split into 6 successes and 3 obvious failures, increasing total successes from 137 to 143 while silent failures remain at 8 (all from o4-mini). Under the baseline condition, crash counts fall much more dramatically, from 111 to 50, but the 61 recovered runs split into only 6 successes, 39 silent failures, and 16 obvious failures. The two conditions therefore send repaired runs in opposite directions:

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

under module-grounded repair, recovered runs predominantly become successes; under unconstrained code generation, they predominantly become silent failures.

To understand why self-debugging changes the two conditions so differently, we classified each non-success run by its dominant root cause (Table C1 in Appendix C; Fig. 8). We classify these into two broad types. *Execution-level* causes include *File/Path error* and *Library/runtime error*. *Methodological* causes include *Units*, *Aggregation/weighting*, and *Algorithm/composition*. In the module-grounded condition we additionally identify *YAML parameter* errors, where a validated tool is invoked with incorrect arguments. This distinction matters because only crashed runs enter the self-debug loop: execution-level causes generate a traceback the model can act on, whereas parameter and methodological errors do not.

Three patterns emerge from the root-cause analysis. First, the module-grounded condition produces zero *Units*, *Aggregation/weighting*, or *Algorithm/composition* errors in either the single-shot or post-debug stage, because temporal resampling, area weighting, and unit conversion are encapsulated inside validated tools rather than written by the LLM. These three methodological categories are precisely the dominant sources of silent failure in the baseline (24 of 27 single-shot silent failures). Second, all 8 module-grounded silent failures belong to a single category — *YAML parameter* — and originate from a single model (o4-mini), reflecting a model-capability limitation rather than a flaw introduced by the debug loop. Self-debugging leaves these 8 runs untouched because incorrect parameters do not raise an exception. Third, in the baseline the self-debug loop nearly doubles the silent-failure count from 27 to 66, and the growth concentrates in *Aggregation/weighting* (24 → 36) and especially *Algorithm/composition* (5 → 34). The growth suggests that fixing a crashed script often introduces or unmasks methodological errors that run silently. By contrast, the module-grounded condition's only post-debug Algorithm/composition failures are three Haiku T6 runs in which the debug loop dropped the runoff-subtraction step from the workflow; the error is in which tools are chained, not in the correctness of any individual tool. Implications of this asymmetry for trust and deployment are discussed in Sect. 4.1.

3.3 Model comparison

In single-shot benchmarking, GPT-5 and Gemini 2.5 Flash both achieve perfect 28/28 successful runs under the module-grounded condition. Claude Opus 4.6 is nearly identical at 27/28, with its single failure being a visible crash (the workflow routed outputs into subdirectories the engine does not create). Together, these three frontier models achieve 83/84 (99 %) success. Under iterative self-debugging, the module-grounded condition repairs the remaining Opus crash into a success (28/28) in one iteration, while the same frontier models under the baseline convert single-shot crashes predominantly into silent failures rather than successes: Opus, GPT-5, and Gemini together produce 26 of the 39 baseline crash-to-silent conversions (67 %). In other words, the most capable models account for two thirds of the additional silent failures introduced by baseline self-debugging, suggesting that higher capability makes the repaired code more convincing but not more correct.

O4-mini, a small reasoning-focused model, achieves 18/28 successes (64 %) but is the sole source of all 8 silent failures under the module-grounded condition. These failures concentrate in tasks with more steps: T1–T3 (3–5 steps) show zero silent failures from o4-mini, whereas T6 (13 steps) shows 4/4 silent failures, driven by a consistent parameter-selection error

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

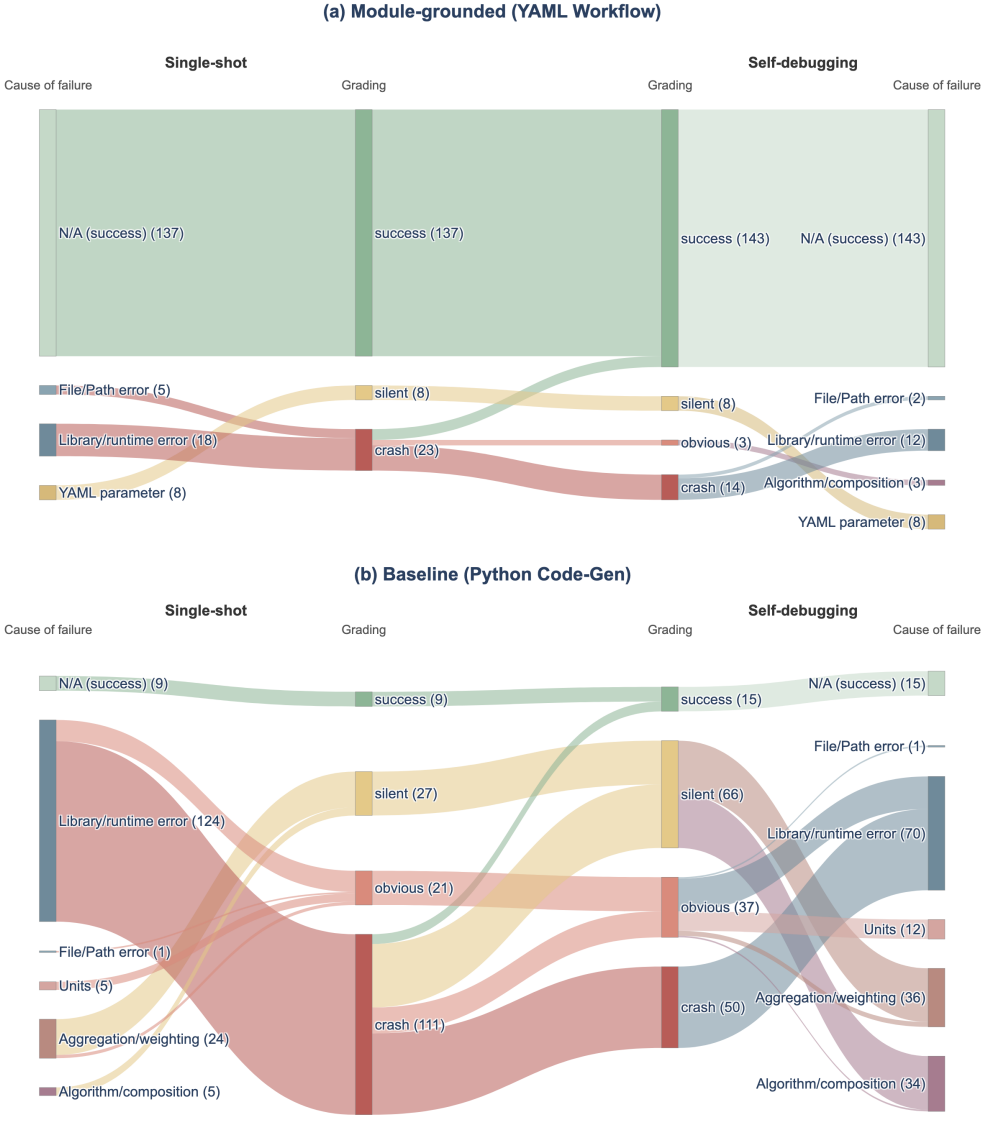

Figure 8. Flow of non-success runs through single-shot grading and post-self-debug regrading under (a) the module-grounded condition and (b) the code-generation baseline. Cause categories are defined in Table C1. Node counts are shown in parentheses; ribbon width is proportional to run count.

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

(`years: [1985, 1989]`, a two-element list instead of the full five-year range). Self-debugging does not touch this failure mode because the error does not raise an exception and therefore generates no traceback to drive repair.

Haiku and Phi-4 handle T1–T5 adequately (mostly success) but crash entirely on T6–T7, where the number of required steps (13–23) and the need to chain intermediate outputs across multiple tools exceeds their compositional capacity. Phi-4's performance is non-monotonic: it crashes on the simplest task (T1, wrong file path) yet succeeds on T2–T5. Under self-debugging, the module-grounded condition recovers a subset of these small-model T6–T7 crashes into successes, whereas the baseline recovers them predominantly into silent or obvious failures.

4 Discussion

Existing approaches to AI-assisted Earth system analysis sit at opposite ends of a methodological spectrum. At one end, agentic systems such as EarthLink (Guo et al., 2025) let the LLM decide everything, generating and executing new analysis code for each request with maximum flexibility but no guarantee of scientific correctness. At the other end, expert-authored diagnostic packages such as E3SM Diagnostics (Zhang et al., 2022) and ESMValTool (Righi et al., 2020) let the LLM decide nothing, providing fixed, trusted workflows with deterministic outputs but requiring domain expertise to configure and extend, and offering no natural-language interface. This study looks for a sweet spot in between, in which the LLM decides what to compute and in what order, but not how each step is implemented. The benchmark suggests that such a sweet spot exists: constraining frontier LLMs to compose validated tools, combined with self-debugging, achieves 100 % success across the three most capable models while producing no methodological silent failures. We acknowledge that baseline failure rates are sensitive to prompt wording, and that different prompts would yield different numbers. The purpose of the baseline, however, is not to optimise code-generation performance through prompt engineering but to illustrate the qualitative gap between the two paradigms. The following subsections discuss what these results imply for failure modes, model capability, and practical deployment.

4.1 Silent failures are correlated across models

Beyond the aggregate failure counts reported in Sect. 3, a qualitative feature of the baseline silent failures deserves attention: they are correlated across models (Wang et al., 2025). Independent LLMs arrive at the same wrong methodological choice when presented with the same task. In T1, five of six models aggregated daily discharge to monthly means before computing extremes. In T3, multiple models applied the $\mathrm{mm\,s^{-1}}$-to-$\mathrm{mm\,day^{-1}}$ conversion to model output but forgot to apply it to observations. These are not random bugs but "reasonable" defaults that many models reach in parallel, which means ensembling or cross-model verification does not protect against them. The module-grounded condition breaks this correlation by construction: unit conversion, temporal aggregation, and gauge matching live inside validated tools, so any model that calls the right tool gets the right method.

Self-debugging is not inherently protective or harmful; its effect depends on whether the analysis logic is free-form or methodologically constrained. Attaching a traceback-driven repair loop to unconstrained code generation converts visible

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

crashes into plausible silent failures, because the loop can only act on errors that raise an exception. Attaching the same loop to module-grounded composition is safe, because a crash there can only come from a wiring or parameter error that the loop is well suited to fix.

4.2 Catalog design and attention limits

The benchmark exposes a tension between catalog completeness and LLM attention capacity. The full tool catalog (∼7000 tokens) fits within all modern context windows, but smaller models struggle to parse it reliably. Phi-4 failed on T1 — the simplest task — by specifying an incorrect file path from the catalog, yet succeeded on T2–T5 where the relevant tools are more distinctive. O4-mini exhibits a similar pattern inside the workflow rather than the catalog: its silent failures cluster on the longer tasks (4/4 on T6), where it consistently interpreted the prompt's "1985–1989" date range as two years (1985 and 1989) rather than the full five-year range. The inverse misinterpretation is also risky in scientific analysis: a user asking the LLM to compute extremes for a few specific years (e.g., "analyse 1985 and 1989") could have the request silently expanded into a full-range average across 1985–1989, which smooths out the extremes being sought but returns output that looks entirely reasonable. Both behaviours are consistent with the "lost in the middle" phenomenon (Liu et al., 2024): attention quality degrades for information buried among many similar entries, whether those entries are catalog items or workflow steps. During development, we also observed that ambiguous catalog entries reliably caused silent errors. When the ILAMB fetcher listed multiple evapotranspiration products under the same CMIP category without specifying internal variable names, every tested LLM guessed the wrong extraction variable. Reducing the catalog to one dataset per variable and annotating the actual NetCDF variable name resolved this problem for all models. This suggests a design principle: catalog entries should be unambiguous and minimal, encoding domain-specific metadata that the LLM cannot infer from context.

4.3 Workflow efficiency and tool library creation.

The module-grounded framework assumes that the user possesses domain expertise: they must know which variables, metrics, and datasets are appropriate for their analysis, and they are responsible for validating that the LLM-generated workflow parameters match what is actually available. This is by design — the framework eliminates software engineering errors, not domain judgment errors. In practice, a researcher replaces hours of writing and debugging analysis scripts (typically 100–300 lines of Python per task) with a natural-language prompt of 50–100 words, while the benchmark results show that the output remains bit-for-bit reproducible against hand-crafted references.

The framework also lowers the barrier to building and sharing a reusable analysis library. Each tool is a self-contained Python function with simple, well-defined inputs and outputs, and existing scripts can be translated, standardized, and registered in the catalog incrementally. Modern LLMs are themselves highly capable of drafting such single-purpose tools: when a researcher needs a new analysis, they can collaborate with an LLM to generate the function, and because each tool has a narrow scope and explicit I/O contract, validation is tractable — a handful of unit tests against known inputs is usually sufficient, in contrast to the much harder problem of validating a multi-step analysis script end-to-end. A well-documented shared catalog then becomes a mechanism for distributing domain expertise across a modeling team: one researcher validates the tools once, and others can

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

compose them through natural-language requests without reimplementing the analysis. Existing workflow YAML files also serve as concrete, executable examples of community conventions. This opens up a potential path for cross-domain use: when a scientist approaches an unfamiliar subfield — for instance, an atmospheric researcher picking up land-surface diagnostics — the LLM can, in principle, consult the catalog together with previously validated workflows to see how another domain community has combined tools, applied units, and handled temporal or spatial aggregation. Demonstrating this cross-domain transfer empirically remains future work. All of the context the LLM consults — tool descriptions, parameter schemas, and example workflows — is stored as plain YAML and Python docstrings, lightweight human-readable formats in the spirit of the markdown-based skill definitions emerging in the AI-assistant ecosystem, so the catalog can be versioned, diffed, and loaded into any LLM context window without custom tooling.

4.4 Limitations and future work

The seven benchmark tasks cover river routing and land-surface hydrology (streamflow validation, runoff climatology, ET benchmarking, water balance closure) but do not test atmosphere, ocean, or sea-ice diagnostics. Key climate analyses — such as computing ENSO indices from sea surface temperatures, evaluating global energy budgets, or diagnosing atmospheric teleconnections — require tools that span multiple ESM components and involve longer, more branching workflows than those tested here. Extending the benchmark to these cross-component analyses is a priority for establishing broader generality. On the other hand, model performance of T7, our most complex 23-step task, already hints at the attention limits of a single-agent approach: Claude Opus 4.6 produced one failure, consistent with degraded attention quality over a long tool catalog. As the catalog grows to cover atmosphere, ocean, and sea-ice components, a multi-agent architecture becomes attractive — an orchestrator agent that interprets the user's question and delegates to domain-specific sub-agents, each operating over a focused subset of the catalog. This would keep each agent's context window manageable while enabling cross-component analyses that no single domain expert could easily script alone.

While the framework validates tool names and workflow structure, and notifies the user the limitations of the toolsets, some LLMs will still attempt to deliver a workflow using available tools even when the requested analysis falls outside the catalog's scope. Verifying parameter values — particularly variable names and dataset paths — to match what is actually available remains an essential step that must be performed by the user (Sect. 4.3).

The tool library itself introduces a maintenance burden. If the underlying ESM changes — new variable names, modified output conventions, or updated file formats — the tools may continue to execute without error but silently produce incorrect results, the same class of silent failure the framework is designed to prevent. Version-pinning the tool library to specific ESM releases and including regression tests against known reference outputs would mitigate this risk, but requires ongoing effort from the tool maintainers.

The benchmark design also has limitations. The self-debug experiment approximates only the simplest realistic repair loop — automatic traceback-driven retry with no human feedback — and is capped at three rounds. In practice, a researcher debugging a baseline script could also intervene on methodology (e.g., noticing that observations and model output are on different time frequencies) rather than only on execution errors, which would partially offset the silent-failure inflation reported here.

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

Conversely, our loop does not attempt to repair single-shot silent or obvious failures because those runs produce no error signal; a user-in-the-loop protocol would cover those cases and likely widen the gap between conditions further. Testing fully interactive multi-turn workflows, including user-guided methodological corrections, remains future work.

On the implementation side, workflow execution is currently sequential; parallel step execution would improve throughput for independent branches. Error recovery is limited — if an intermediate step fails, downstream steps are skipped rather than retried. The YAML schema does not support conditional logic, limiting workflow expressiveness. Future work includes parallel execution, conditional branching, and expansion of the tool library to additional ESM components.

5 Conclusions

Can we trust LLMs for complex Earth system model analysis? Our answer is conditional, and the condition is what the LLM is asked to do. We developed **ESFlow**, a module-grounded framework in which the LLM composes declarative YAML workflows over a validated tool catalog instead of writing analysis code from scratch, and evaluated it against an unconstrained code-generation baseline. The benchmark comprises 336 single-shot runs (6 models $\times$ 7 tasks $\times$ 4 replicates $\times$ 2 conditions), plus an iterative self-debug experiment in which every crashed run received up to three rounds of traceback-driven repair. The main findings are:

- **Unconstrained code generation is unreliable for ESM analysis.** Only 5 % of baseline runs succeeded; 66 % crashed, and 16 % produced silent failures — plausible, well-formatted output that numerically disagrees with hand-crafted references.
- **Silent-failure errors are correlated across models.** Different LLMs independently made the same "reasonable" methodological mistakes (e.g. mishandled unit conversions, missing area weighting), which means cross-model cross-checking is not a safeguard.
- **Module-grounded composition reproduces reference output at high fidelity.** 82 % of module-grounded runs matched the reference bit-for-bit; obvious failures dropped to zero; the three frontier models (Claude Opus 4.6, GPT-5, Gemini 2.5 Flash) together achieved 99 % success (83/84 runs). All 8 residual silent failures originated from a single smaller reasoning model and collapsed to one narrow class of YAML parameter errors.
- **Self-debugging is safe only when the LLM is constrained.** Three rounds of traceback-driven repair improved the module-grounded pipeline (6 of 9 crashes recovered to success, silent failures unchanged at 8) but actively degraded the baseline: 39 of 61 recovered code-generation crashes resurfaced as silent failures, raising the baseline silent-failure count from 27 to 66.
- **The mechanism is structural.** Silent failures in unconstrained code originate where science-critical choices — unit handling, area weighting, algorithm composition — are embedded in generated code. Routing those choices through a

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

catalog that a domain expert has already validated turns the LLM's task from generative to combinatorial and removes the class of errors at its source.

The resulting architecture — a self-describing tool catalog, a declarative workflow artifact, and explicit data wiring — is domain-agnostic. The library demonstrated here targets E3SM water-cycle analysis, but the pattern applies to any field where scientists already maintain trusted scripts, reproducibility matters more than creative exploration, and the cost of a plausible-but-wrong answer is high. The tool library, benchmark suite, and workflow examples are available as open-source software.

Code and data availability. A frozen snapshot of the ESFlow source code (corresponding to the version used in this paper) together with the sample datasets required to run the reference workflows (E3SM model output and GRDC streamflow observations) and the full per-model benchmark outputs is archived on Zenodo at https://doi.org/10.5281/zenodo.19350842 (Zhou, 2026) under a BSD 3-Clause License. The archive includes the benchmark task prompts, reference workflows, and evaluation scripts under `benchmark/`. ILAMB observation data are fetched automatically by the workflow engine from https://www.ilamb.org/ILAMB-Data/ during execution. Ongoing development of ESFlow continues in a public GitHub repository; the URL will be provided in the final version of the manuscript.

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

Appendix A: Tool library details

Table A1 lists all 24 tools by category.

Table A1. Tool library summary. Each tool is a single Python function with typed inputs and named outputs. The 24 tools expose a total of 92 input parameters.

Category	n	Representative tools
Fetchers	1	`fetch_ilamb_data` (18 variable/-dataset pairs)
Loaders	1	`load_obs_metadata`
Matchers	1	`match_to_grid`
Extractors	4	`extract_basin_mean`, `extract_gridded_field`, `extract_e3sm_timeseries`, `extract_obs_timeseries`
Analyzers	7	`compute_basin_budget`, `compute_climatology`, `compute_fdc_metrics`, `compute_metrics`, `compute_spatial_bias`, `compute_summary_stats`, `compute_zonal_stats`
Plotters	10	`plot_basin_timeseries`, `plot_basin_budget_comparison`, `plot_basin_radar`, `plot_bias_comparison`, `plot_fdc`, `plot_gridded_map`, `plot_map`, `plot_scatter`, `plot_timeseries`, `plot_water_balance_basins`

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

Appendix B: Tool registration example

Listing B1 shows a minimal tool registration. The `ToolSpec` declares typed inputs and named outputs; the `@esflow_tool` decorator handles validation, type coercion, and catalog registration.

Listing B1. Minimal tool registration example.

```python
from core.base import esflow_tool, ToolSpec, Param

SPEC = ToolSpec(
    name='compute_zonal_stats',
    description='Area-weighted statistics by region',
    inputs={
        'field_file': Param('path', required=True,
            description='Input NetCDF file'),
        'lat_bands': Param('str', required=False,
            default='',
            description='Bands as "name:south:north"'),
    },
    outputs={
        'stats_file': {'type': 'csv',
            'description': 'Regional statistics'},
    },
)

@esflow_tool(SPEC)
def run(config: dict) -> dict:
    # Implementation here  never seen by the LLM
    ...
```

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

Appendix C: Root-cause categories for non-success runs

Table C1. Root-cause categories used to classify every non-success run. Execution-level causes (top two rows) generate a traceback that the self-debug loop can act on; methodological causes (middle three rows) execute without raising an exception and therefore bypass the loop. YAML parameter errors (bottom row) apply only to the module-grounded condition.

Category	Description and representative examples
File/Path error *(execution-level)*	Crash from a wrong metadata path, missing input file, or unsupported output location. • Wrong gauge-file path (Phi-4, T1) • Output organised into subdirectories the engine does not create (Opus, T7) • Incorrect observation-metadata path (Haiku, T4)
Library/runtime error *(execution-level)*	Language-level or library-level failure during script or tool execution. • Calendar-type mismatch when coercing E3SM no-leap dates (multiple models, T4/T5) • NumPy shape mismatch during gauge-to-grid matching, e.g. (360,) vs. (720,) (GPT-5, T4) • Lazy-array evaluation failure on scalar extraction (Gemini, T7) • Polygon-clipping error on self-intersecting basin geometries (multiple models, T7) • Missing or incorrect import statements (Phi-4, T6)
Units *(methodological)*	Missing or partial unit conversion that produces plausible but numerically wrong output. • Converts some variables to mm day^{-1} but leaves others in mm s^{-1} (multiple models, T7) • Omits MODIS kg m^{-2} s^{-1} to mm day^{-1} conversion (multiple models, T3) • Applies wrong conversion factor ($\times$86.4 instead of $\times$86 400) (o4-mini, T7)
Aggregation / weighting *(methodological)*	Incorrect temporal aggregation, spatial weighting, or land/ocean masking. • Daily-to-monthly aggregation inconsistent between model and observations (multiple models, T2/T4) • Missing land-fraction weighting, causing 1–2 % drift at coastal cells (multiple models, T6) • Area-weighted mean over all cells including ocean zeros, diluting land-only values (multiple models, T3) • Drops cells with negative runoff before averaging, biasing upward (Gemini, T6)
Algorithm / composition *(methodological)*	Wrong analytical method, missing workflow step, or incorrect metric definition. • Bilinear interpolation instead of conservative remapping (multiple models, T3) • Un-normalised Wasserstein distances reported as raw values, e.g. 54 144 vs. reference 0.32 (Opus, T7) • P−ET reported instead of P−ET−Q, omitting the runoff-subtraction step (Haiku, T6) • Hand-rolled spherical area weights instead of model-native area variable (GPT-5, T6) • Longitude convention mismatch (0–360° vs. −180–180°) producing all-NaN fields (Phi-4, T3)
YAML parameter *(parameter; module-grounded only)*	Structurally valid workflow that invokes a validated tool with incorrect arguments; the tool executes normally but on wrong inputs. • `years: [1985, 1989]` (two-element list) instead of the full five-year range, loading only 2 of 5 years (o4-mini, T5/T6/T7) • Wrong variable name passed to a correctly selected extraction tool (o4-mini, T4)

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

Author contributions. TZ: Conceptualization, Methodology, Software, Validation, Investigation, Funding acquisition, Writing – original draft. YQ and LRL: Conceptualization, Writing – review and editing.

Competing interests. The authors declare that they have no conflict of interest.

Acknowledgements. The authors acknowledge the use of the PNNL AI Incubator for providing API access to Claude and GPT models, and the Google AI API for providing access to Gemini models, used in the benchmark evaluation. Manuscript preparation was assisted by Claude Code (Opus 4.6 and Sonnet 4.6 from Anthropic), which was used for LaTeX formatting and scripting assistance. All content was reviewed, edited, and approved by the authors to ensure scientific accuracy and relevance.

Financial support. This research was supported by the U.S. Department of Energy, Office of Science, Biological and Environmental Research program as part of the Interdisciplinary Research for Arctic Coastal Environments (InteRFACE) project under contract Grant Number 89233218CNA000001 to Triad National Security, LLC ("Triad"), and the Energy Exascale Earth System Model (E3SM) project. Pacific Northwest National Laboratory is operated by Battelle Memorial Institute for the U.S. Department of Energy under Contract DE-AC05-76RL01830.

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

References

Collier, N., Hoffman, F. M., Lawrence, D. M., Keppel-Aleks, G., Koven, C. D., Riley, W. J., Mu, M., and Randerson, J. T.: The International Land Model Benchmarking (ILAMB) System: Design, Theory, and Implementation, Journal of Advances in Modeling Earth Systems, 10, 2731–2754, https://doi.org/10.1029/2018MS001354, 2018.

Eyring, V., Bock, L., Lauer, A., Schlund, M., Andela, B., Drost, N., Righi, M., et al.: Earth System Model Evaluation Tool (ESMValTool) v2.0 – an extended set of large-scale diagnostics for quasi-operational and comprehensive evaluation of Earth system models in CMIP, Geoscientific Model Development, 13, 3383–3438, https://doi.org/10.5194/gmd-13-3383-2020, 2020.

Golaz, J.-C., Van Roekel, L. P., Zheng, X., Roberts, A. F., Wolfe, J. D., Lin, W., Bradley, A. M., Tang, Q., Maltrud, M. E., Forsyth, R. M., Zhang, C., Zhou, T., Zhang, K., Zender, C. S., Wu, M., Wang, H., Turner, A. K., Singh, B., Richter, J. H., Qin, Y., Petersen, M. R., Mametjanov, A., Ma, P.-L., Larson, V. E., Krishna, J., Keen, N. D., Jeffery, N., Hunke, E. C., Hannah, W. M., Guba, O., Griffin, B. M., Feng, Y., Engwirda, D., Di Vittorio, A. V., Dang, C., Conlon, L. M., Chen, C.-C.-J., Brunke, M. A., Bisht, G., Benedict, J. J., Asay-Davis, X. S., Zhang, Y., Zhang, M., Zeng, X., Xie, S., Wolfram, P. J., Vo, T., Veneziani, M., Tesfa, T. K., Sreepathi, S., Salinger, A. G., Reeves Eyre, J. E. J., Prather, M. J., Mahajan, S., Li, Q., Jones, P. W., Jacob, R. L., Huebler, G. W., Huang, X., Hillman, B. R., Harrop, B. E., Foucar, J. G., Fang, Y., Comeau, D. S., Caldwell, P. M., Bartoletti, T., Balaguru, K., Taylor, M. A., McCoy, R. B., Leung, L. R., and Bader, D. C.: The DOE E3SM Model Version 2: Overview of the Physical Model and Initial Model Evaluation, Journal of Advances in Modeling Earth Systems, 14, e2022MS003 156, https://doi.org/10.1029/2022MS003156, 2022.

GRDC: GRDC — The Global Runoff Data Centre, 56068 Koblenz, Germany, https://www.bafg.de/GRDC/, accessed: 2026-03-15, 2024.

Guo, Z., Wang, J., Ling, F., Wei, W., Yue, X., Jiang, Z., Xu, W., Luo, J.-J., Cheng, L., Ham, Y.-G., Song, F., Gentine, P., Yamagata, T., Fei, B., Zhang, W., Gu, X., Li, C., Wang, Y., Chen, T., Ouyang, W., Zhou, B., and Bai, L.: EarthLink: Interpreting Climate Signals with Self-Evolving AI Agents, arXiv preprint arXiv:2507.17311, 2025.

Lee, J., Gleckler, P. J., Ahn, M.-S., Ordonez, A., Ullrich, P. A., Sperber, K. R., Taylor, K. E., Planton, Y. Y., Guilyardi, E., Durack, P., Bonfils, C., Zelinka, M. D., Chao, L.-W., Dong, B., Doutriaux, C., Zhang, C., Vo, T., Boutte, J., Wehner, M. F., Pendergrass, A. G., Kim, D., Xue, Z., Wittenberg, A. T., and Krasting, J.: Systematic and objective evaluation of Earth system models: PCMDI Metrics Package (PMP) version 3, Geoscientific Model Development, 17, 3919–3948, https://doi.org/10.5194/gmd-17-3919-2024, 2024.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P.: Lost in the Middle: How Language Models Use Long Contexts, Transactions of the Association for Computational Linguistics, 12, 157–173, https://doi.org/10.1162/tacl_a_00638, 2024.

Pantiukhin, D., Shapkin, B., Kuznetsov, I., Jost, A.-A., and Koldunov, N.: Accelerating Earth Science Discovery via Multi-Agent LLM Systems, Frontiers in Artificial Intelligence, 8, 1674 927, https://doi.org/10.3389/frai.2025.1674927, 2025.

Righi, M., Andela, B., Eyring, V., Lauer, A., Predoi, V., Schlund, M., Vegas-Regidor, J., Bock, L., Brötz, B., de Mora, L., Diblen, F., Dreyer, L., Drost, N., Earnshaw, P., Hassler, B., Koldunov, N., Little, B., Loosveldt Tomas, S., and Zimmermann, K.: Earth System Model Evaluation Tool (ESMValTool) v2.0 – technical overview, Geoscientific Model Development, 13, 1179–1199, https://doi.org/10.5194/gmd-13-1179-2020, 2020.

Siddiq, M. L., Islam-Gomes, A., Sekerak, N., and Santos, J. C. S.: Large Language Models for Software Engineering: A Reproducibility Crisis, arXiv preprint arXiv:2512.00651, 2025.

Stamoulis, D. and Marculescu, D.: Geo-OLM: Enabling Sustainable Earth Observation Studies with Cost-Efficient Open Language Models & State-Driven Workflows, in: Proceedings of the 2025 ACM SIGCAS/SIGCHI Conference on Computing and Sustainable Societies (COMPASS '25), pp. 608–619, ACM, https://doi.org/10.1145/3715335.3735494, 2025.

https://doi.org/10.5194/egusphere-2026-2237
Preprint. Discussion started: 27 April 2026
© Author(s) 2026. CC BY 4.0 License.

Vangala, B. P., Adibifar, A., Malik, T., and Gehani, A.: AI-Generated Code Is Not Reproducible (Yet): An Empirical Study of Dependency Gaps in LLM-Based Coding Agents, arXiv preprint arXiv:2512.22387, 2025.

Wang, Z., Zhou, Z., Song, D., Huang, Y., Chen, S., Ma, L., and Zhang, T.: Towards Understanding the Characteristics of Code Generation Errors Made by Large Language Models, in: Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE '25), https://doi.org/10.1109/ICSE55347.2025.00180, 2025.

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., Gonzalez-Beltran, A., Gray, A. J. G., Groth, P., Goble, C., Grethe, J. S., Heringa, J., 't Hoen, P. A. C., Hooft, R., Kuhn, T., Kok, R., Kok, J., Lusher, S. J., Martone, M. E., Mons, A., Packer, A. L., Persson, B., Rocca-Serra, P., Roos, M., van Schaik, R., Sansone, S.-A., Schultes, E., Sengstag, T., Slater, T., Strawn, G., Swertz, M. A., Thompson, M., van der Lei, J., van Mulligen, E., Velterop, J., Waagmeester, A., Wittenburg, P., Wolstencroft, K., Zhao, J., and Mons, B.: The FAIR Guiding Principles for scientific data management and stewardship, Scientific Data, 3, 160 018, https://doi.org/10.1038/sdata.2016.18, 2016.

Zhang, C., Golaz, J.-C., Forsyth, R., Vo, T., Xie, S., Shaheen, Z., Potter, G. L., Asay-Davis, X. S., Zender, C. S., Lin, W., Chen, C.-C., Terai, C. R., Mahajan, S., Zhou, T., Balaguru, K., Tang, Q., Tao, C., Zhang, Y., Emmenegger, T., Burrows, S., and Ullrich, P. A.: The E3SM Diagnostics Package (E3SM Diags v2.7): a Python-based diagnostics package for Earth system model evaluation, Geoscientific Model Development, 15, 9031–9056, https://doi.org/10.5194/gmd-15-9031-2022, 2022.

Zhou, T.: ESFlow: source code, sample data, and benchmark outputs for "Can We Trust LLMs for Complex Earth System Model Analysis?", https://doi.org/10.5281/zenodo.19350842, 2026.