PROCIF: EVALUATING PROCEDURAL INSTRUCTION FOLLOWING IN LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models increasingly execute tasks that specify not only what outcome to produce, but also how evidence should be collected, actions should be ordered, failures should be handled, and results should be validated. A correct final answer, however, does not establish that the prescribed process was followed; conversely, procedural compliance does not guarantee a correct outcome. We introduce **ProcIF**, a cross-domain, two-track procedural instruction following benchmark. It contains 342 instances and 4,157 executable process constraints. Track A adds task-specific procedural constraints to writing, scientific reasoning, workflow synthesis, and tool-use tasks, whereas Track B converts source harness trajectories into visible, observation-conditioned Skills or standard operating procedures. We report strict Procedure Adherence Score (PAS), source-aware Task Success Rate (TSR), and their instance-level conjunction. Across the evaluated models, the best strict PAS and Joint scores reach only 41.23% and 35.96%, respectively. These results show that models often satisfy local requirements while failing the complete procedural contract, making procedural instruction following a challenging and substantially unresolved capability.

1 INTRODUCTION

Language models are increasingly expected to carry out multi-step tasks under detailed instructions, tool-use policies, and standard operating procedures (SOPs) (Qi et al., 2026; Panavas et al., 2026). In these settings, instructions specify not only a desired result, but also which actions are permitted, what evidence must be used, and when a result may be submitted. These requirements provide auditability, provenance, and operational safeguards. Once stated, they remain part of the task even when a model can reach the correct endpoint by a shorter route. We refer to the ability to satisfy these explicit process requirements while pursuing the underlying task as **procedural instruction following**. Task success alone cannot establish procedural instruction following. In Figure 1(a), both traces fix the target bug, yet one omits the required backup and modifies a prohibited file. The successful outcome therefore conceals violations of the request. The converse also holds: a model may perform every required step and pass the prescribed tests while its patch still fails an independently assessed case outside their coverage (Figure 1(b)). Thus, neither outcome correctness nor procedural adherence can substitute for the other.

Existing benchmarks address important subsets of this problem. Response-level instruction-following evaluations test verifiable constraints and their composition (Zhou et al., 2023; Pyatkin et al., 2026; Jiang et al., 2024; Wen et al., 2024; Su et al., 2026). Agentic evaluations further study tool specifications, long-horizon policies, dynamic SOPs, and multi-turn software development (Qi et al., 2026; Panavas et al., 2026; Shi et al., 2026; Wen et al., 2026). We introduce **ProcIF**, a benchmark centered on the question: *can a language model solve an underlying task while satisfying all applicable requirements of an explicit procedure?* ProcIF comprises 342 instances and 4,157 executable process constraints. Track A adds task-specific procedural requirements to writing, scientific reasoning, workflow synthesis, and tool-use tasks. Track B abstracts task-relevant source harness trajectories into visible Skills/SOPs, testing branching, evidence updates, recovery, and validation in response to runner observations. Two tracks differ in how procedures are constructed, while both preserve the original task objective and its success criterion.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

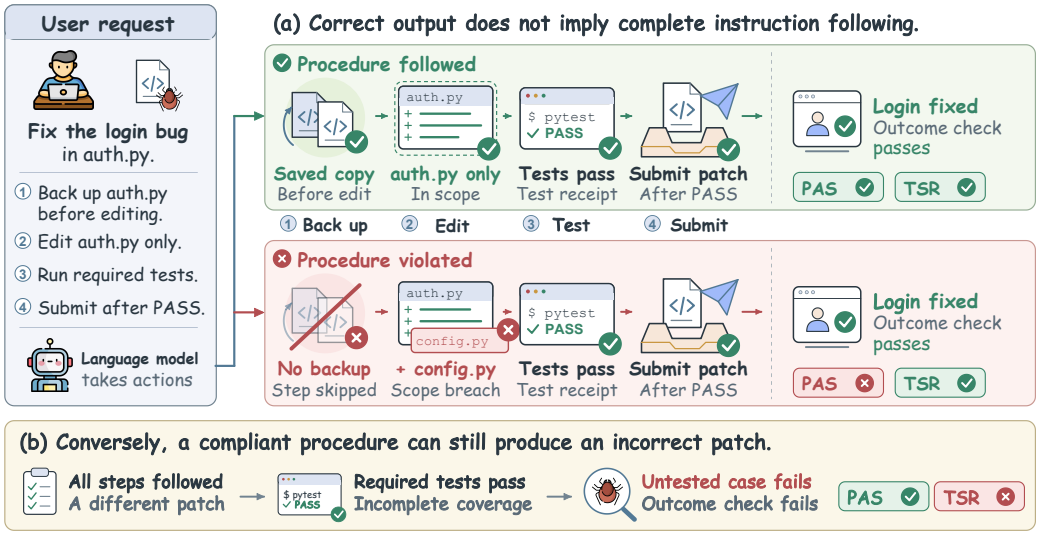

Figure 1: Task success and procedural adherence can diverge in both directions.

A central evaluation challenge is to make natural-language procedures auditable without imposing an unstated reference path. ProcIF maps visible requirements to executable predicates over events, artifacts and their versions, evidence references, tool calls, and runner states. Task-specific lightweight runners expose bounded, state-dependent observations and execution receipts. The resulting contracts constrain only properties specified to the model, leaving unspecified choices free and admitting alternative executions when the requirements allow them. ProcIF therefore evaluates observable execution rather than the faithfulness of hidden chain-of-thought reasoning. We measure adherence and task success separately. Strict Procedure Adherence Score (PAS) counts an instance as successful only when every active process predicate passes. Task Success Rate (TSR) instead evaluates the original outcome using task-appropriate criteria. Joint reports the fraction of instances passing both. This design supports a four-way decomposition into both pass, procedure only, task only, and both fail, exposing failures that any single aggregate metric would hide. Our contributions are as follows:

- We propose ProcIF, a cross-domain, two-track procedural instruction following benchmark. It unifies 180 task-specific procedural-constraint instances with 162 observation-conditioned Skill/SOP instances, spanning 4,157 executable requirements while preserving source-task success criteria.
- We provide an auditable trajectory-level evaluation that compiles visible procedures into deterministic contracts over observable evidence while permitting multiple valid executions. By measuring procedural adherence and task success separately, and a four-way outcome decomposition, ProcIF diagnoses both process and outcome failures.
- We evaluate a diverse set of LLMs and find that even the best model attains only 41.23% strict PAS and 35.96% Joint, confirming that procedural instruction following is a critical and largely unsolved problem. ProcIF provides a clear baseline for the current state of this field.

2 Related Work

2.1 Instruction-Following Evaluation

Instruction-following benchmarks assess whether a response explicitly satisfies user instructions. IFEval (Zhou et al., 2023) and IFBench (Pyatkin et al., 2026) translate requirements concerning length, keywords, and formatting into programmatically verifiable conditions, and measure precise instruction following by applying these checks to generated responses. Subsequent work broadens both the semantic scope of constraints and the complexity of their composition. FollowBench (Jiang et al., 2024) broadens coverage to content, situation, style, format, and example constraints, using

Table 1: Comparison of representative instruction-following benchmarks. Evidence denotes what is inspected rather than what an instruction requests: R = completed textual response; A = task artifacts; S = environment/service state; T = execution or dialogue records. **Evaluation:** Adherence checks explicit instructions or SOP constraints; Outcome assesses underlying task completion; Separate requires distinct adherence and outcome scores.

Benchmark	Evidence	Evaluation			Interaction		Domain
		Adherence	Outcome	Separate	Tools	Dialogue	
IFEval/IFBench	R	✓	✗	✗	✗	✗	General
FollowBench	R	✓	✗	✗	✗	✗	General
ComplexBench	R	✓	✗	✗	✗	✗	General
SciIF	R	✓	✓	✓	✗	✗	Science
AgentIF	R	✓	✗	✗	✗	✗	Agentic
HANDBOOK.md	A/S	✓	✓	✗	✓	✗	Enterprise
SAGE	T	✓	✗	✗	✗	✓	Service
MTAC-IFBench	A/T	✓	✓	✓	✓	✓	Coding
SkillsBench	A	✗	✓	✗	✓	✗	Cross-domain
ProcIF	**R/A/S/T**	✓	✓	✓	✓	✓	**Cross-domain**

progressively constrained instructions to characterize difficulty. ComplexBench (Wen et al., 2024) explicitly models the composition of multiple constraints, including task chains and conditional selection, and aggregates scores according to their dependency structure. SciIF (Su et al., 2026) brings this perspective to scientific reasoning problems. These studies establish a strong evaluation paradigm of instruction following. However, these benchmarks evaluate primarily through the model's final answer, rather than assessing the interactive execution trace through which that output is produced. ProcIF extends the object of evaluation from final responses to observable process traces and execution evidence.

2.2 Procedural and SOP-Guided Evaluation

As models increasingly perform tool invocation and long-horizon tasks, evaluation has begun to address instructions that more closely resemble operational procedures. AgentIF (Qi et al., 2026) collects long instructions from real agentic applications, covering tool specifications, semantic requirements, and conditional constraints. But its primary protocol verifies the applicable constraints in a model response; it does not unify external tool execution, dynamic observations, and original task within a closed loop. HANDBOOK.md (Panavas et al., 2026) evaluates long-horizon procedural compliance with SOPs in company workspaces with mock services. SAGE (Shi et al., 2026) formalizes customer-service SOPs as dynamic graphs. It combines multi-turn interactions between user and service models with judge agents and a rule engine to evaluate the logical correctness of classifications, paths, and actions, as well as dialogue quality. MTAC-IFBench (Wen et al., 2026) focuses on multi-turn software development, covering tool use, orchestration, and testing requirements while separately evaluating instruction compliance and build success and functional correctness. Together, these works show that process-oriented evaluation has expanded from operational constraints to long-horizon policy compliance and multi-turn domain tasks. Nevertheless, most benchmarks remain centered on a particular application domain.

In addition, most skill-related benchmarks evaluate whether providing a model with a skill improves task performance. SkillsBench (Li et al., 2026) pairs cross-domain tasks with structured skills and compares task success with and without a skill under matched task and environment conditions. This measures skill utility and effectiveness, rather than systematically checking whether every requirement in the supplied skill was followed. Grounded Checklist Partial Credit (GCPC) (Qin et al., 2026) further observes that binary task success can conceal both the influence of a Skill on execution and the partial progress made by an agent. Therefore, Skill utility and Skill adherence are related but not equivalent: following a procedure need not improve the task outcome, and an improved outcome need not imply complete adherence to the procedure.

In summary, ProcIF forms a complement with output-level instruction-following, domain-specific SOP, and Skill-utility evaluation. Within a unified, lightweight, and auditable framework, it brings

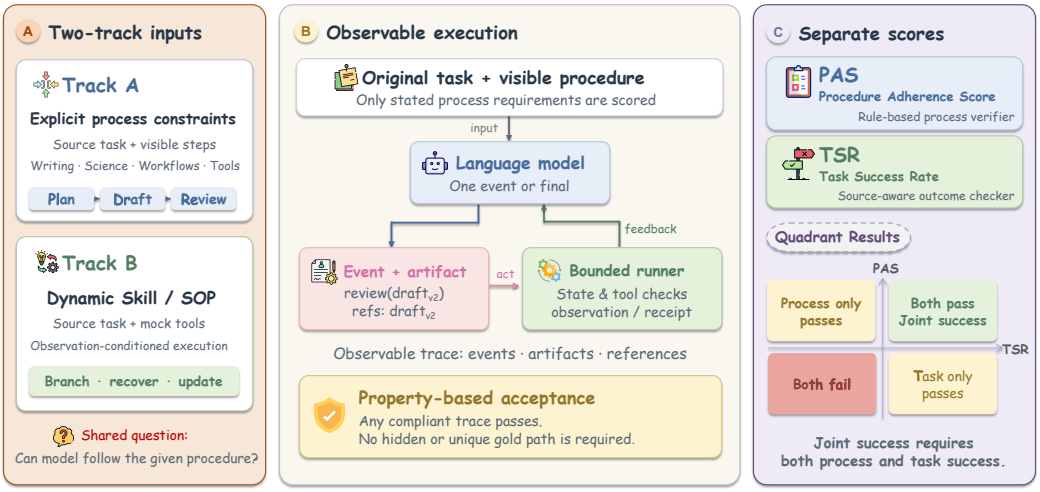

Figure 2: **Overview of the ProcIF construction and evaluation.** The two construction tracks feed a shared interactive evaluation: a model produces an observable trajectory, which is independently scored for procedure adherence (PAS) and source-task success (TSR).

cross-task explicit process requirements and dynamic Skill/SOP execution into the same measurement setting, separately characterizing adherence to a prescribed procedure and completion of the original task. Table 1 summarizes the differences between ProcIF and other representative instruction-following benchmarks.

3 Methodology

ProcIF evaluates whether a model can complete a task while satisfying an explicitly stated procedure. Each instance couples an original task with a visible set of process requirements. As illustrated in Figure 2, we construct data through two complementary tracks: Track A introduces explicit, task-specific procedural constraints, whereas Track B instantiates Skills or standard operating procedures (SOPs). We then evaluate the resulting trajectory and task outcome separately using Procedure Adherence Score (PAS) and Task Success Rate (TSR).

3.1 Task Definition

We represent the i-th benchmark instance as $\mathcal{B}_i = (x_i, p_i, \mathcal{A}_i, \mathcal{E}_i, \mathcal{C}_i, g_i)$, where x_i is the original task and its necessary context, p_i is the visible procedure, $\mathcal{A}_i$ specifies the available events or tools, and $\mathcal{E}_i$ defines the bounded environment and its observation transitions. The procedural contract $\mathcal{C}_i = \{c_{ij}\}_{j=1}^{m_i}$ contains executable predicates grounded in p_i, while g_i is a source-aware evaluator for the original task. The model receives x_i, p_i, the public action interface, and only the observations available up to the current step. Evaluator-only references, future observations, and private verdicts are never exposed to the model. Although the executable form of $\mathcal{C}_i$ remains evaluator-side, every scored obligation must be entailed by the visible instructions or interface specification.

Under a fixed interaction protocol, interaction between the model and a task-specific lightweight runner yields an observable trajectory $\tau_i = (e_{i,1}, o_{i,1}, \ldots, e_{i,T_i}, o_{i,T_i}, y_i)$, where $e_{i,t}$ is a model-proposed event, artifact operation, or tool call; $o_{i,t}$ is the corresponding observation, receipt, or error returned by the runner; and y_i is the final response or submitted artifact. Events may record action names, arguments, artifact identifiers, contents and input references. For text-only tasks, observations may be limited to protocol acknowledgments; for interactive tasks, runner-generated returns additionally provide trusted evidence about tool execution and environment state.

The model is responsible for choosing actions, supplying arguments, creating and referencing artifacts, reacting to observations, and deciding when to submit a final response. The task-specific runner executes or rejects actions, maintains the prescribed state when applicable, returns observations, and records both accepted and rejected attempts. Consequently, a ProcIF score characterizes a model configuration under a fixed interaction protocol, rather than an unconstrained production

agent. Moreover, adherence is defined by properties of the observable trajectory, not by token-level imitation of a reference trace. Any trajectory satisfying all stated predicates is admissible; only orders, dependencies, cardinalities, and branches explicitly required by the procedure are constrained. ProcIF therefore evaluates externally auditable procedure following, not the faithfulness of hidden chain-of-thought reasoning.

3.2 Data Construction

3.2.1 Overview

ProcIF contains 342 instances: 180 in Track A and 162 in Track B. Together, they instantiate 4,157 executable process constraints; Appendix A.1 reports the full source, interaction-mode, constraint, and evaluator composition. Both tracks follow the same construction principles. Procedures must be task-grounded, visible and sufficient to determine all scored obligations, expressible through observable evidence, and executable in a bounded environment. We first recover the original task objective and its success criterion, then construct a visible procedure, compile each clause into one or more contract predicates, and attach an appropriate runner and outcome evaluator. We preserve an explicit mapping from visible clause to predicate, parameters, required evidence, and failure reason. This separation prevents an evaluator from silently introducing steps that were not communicated to the model and prevents procedural completion from substituting for task correctness.

3.2.2 Track A: Explicit Procedural Constraints

Track A starts from source tasks with independently recoverable success criteria, spanning writing, scientific reasoning, structured workflow construction, and multi-turn tool use. For each task, we first isolate the original objective and any native output requirements. We then author a set of task-specific process constraints covering operations such as planning, decomposition, evidence transfer, verification, and controlled finalization. These clauses are designed to be reasonable and jointly satisfiable, but need not be logically necessary for obtaining the correct answer: once requested by the user, they form part of the required delivery process. Track A is defined by how its procedures are authored, not by a single execution or grading mode: it includes both text-only and tool-mediated tasks, and its outcome evaluators range from exact matching to structured and semantic checks.

3.2.3 Track B: Observation-Conditioned Skills/SOPs

Track B begins with real task-relevant source harness trajectories. We extract the original task, tool interface, operational dependencies, and necessary environment returns, remove sensitive or irrelevant context, and rewrite the resulting procedure as a visible Skill/SOP. Source observations are converted into versioned evaluator fixtures exposed only when the corresponding state and action conditions are met. The source trajectory guides construction and provenance, but is not treated as a hidden path that the model must reproduce. A separate result reference or rubric specifies what constitutes success on the original task. In addition, Track B is observation-conditioned: the same deterministic runner can return different observations, failures, or receipts depending on the current state and the model's action. The visible Skill may require the model to consume an observation, choose a disposition, recover from a failed action, update an active evidence ledger, complete a checkpoint, and validate the current artifact before finalization. Future evidence is not revealed in advance. This controlled replay makes conditional behavior reproducible, but it does not claim to reproduce an open web, a complete production service, or every side effect of the source environment.

3.2.4 Human Annotation

Human reviewers assess each candidate along three axes: (1) task quality: clarity, self-containment, and solvability, (2) process fit: relevance, consistency, and joint satisfiability of the procedure, (3) evaluation fit: alignment among visible clauses, runner behavior, process predicates, and the task-success evaluator. Tasks with missing inputs, non-closed evidence, conflicting targets, or mismatched task and reference scope are revised or removed.

Table 2: Capability taxonomy for observable procedural requirements. Dimensions may overlap within an instance.

ID	Capability dimension	What it tests
R1	Stage coverage	Whether all required intermediate stages are executed.
R2	Ordering & Position	Whether the required order and adjacency between steps are maintained.
R3	Decomposition & Merge	Whether multiple branches are produced independently before a merge.
R4	Evidence dependency & transfer	Whether downstream steps use the required, current upstream evidence.
R5	Validation & Falsification	Whether task-appropriate checks are performed on the specified object.
R6	Cardinality & Prohibition	Whether stated count bounds and prohibitions are respected.
R7	Validation & Finalization	Whether finalization follows required validation.
R8	Conditional action & recovery	Whether branching, retry, and recovery follow the actual observation.

3.3 Evaluation Protocol

At each turn, the model emits one structured event, tool call, or final submission. The runner checks syntax, arguments, and state-dependent preconditions, executes or rejects the proposal, and returns an observation or receipt. Interaction terminates upon a valid final submission, budget exhaustion, detected stalling, or an infrastructure failure. Table 2 summarizes the eight overlapping capability dimensions used to characterize procedural requirements. These dimensions are semantic labels for analysis; implementation-level constraint types and operators may realize one or more dimensions. For each constraint c_{ij}, the deterministic process verifier returns $a_{ij}(\tau_i) \in \{0, 1, \perp\}$, denoting pass, fail, or inactive. Inactivity is used only for a conditional obligation whose trigger did not occur; omission of a required trigger-producing stage is handled by the corresponding coverage or dependency predicate. Let $\mathcal{I}_i(\tau_i)$ be the set of active constraints. The task-level strict score and benchmark PAS are

$$\mathrm{PAS}_i = \mathbf{I}[\forall j \in \mathcal{I}_i(\tau_i), a_{ij}(\tau_i) = 1], \qquad \mathrm{PAS} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{PAS}_i. \tag{1}$$

PAS is strict: violating any active requirement makes $\mathrm{PAS}_i = 0$. The verifier also returns a decision, reason, and evidence location for every constraint, enabling failure analysis rather than only aggregate scoring. We additionally report constraint-level micro-PAS as a diagnostic measure of partial compliance:

$$\text{micro-PAS} = \frac{\sum_i \sum_{j \in \mathcal{I}_i(\tau_i)} a_{ij}(\tau_i)}{\sum_i |\mathcal{I}_i(\tau_i)|}. \tag{2}$$

PAS answers whether the prescribed procedure was followed; it does not establish that the underlying task was solved. We therefore define:

$$\mathrm{TSR}_i = g_i(x_i, y_i, z_i) \in \{0, 1\}, \qquad \mathrm{TSR} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{TSR}_i, \tag{3}$$

where z_i contains only outcome-relevant execution evidence, such as the final environment state, the workflow artifact referenced by the final submission, or executed tool calls required by the source task. Depending on the task, g_i uses exact answer matching, deterministic structured validation, native tool-call or milestone checks, a fixed semantic rubric, or a conjunction of rule-based and semantic components. The two evaluators are independent in objective and decision rule, not necessarily statistically independent and not necessarily disjoint in every input field. When the original task intrinsically requires an executed action, TSR may inspect the corresponding result state, but it never substitutes a PAS verdict for outcome success. Semantic evaluators receive the original task, the final response, and only the permitted outcome evidence. Finally, we report:

$$\mathrm{Joint} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{PAS}_i \mathrm{TSR}_i, \tag{4}$$

which is the fraction of instances for which the model both follows all procedural requirements and solves the original task. The resulting four-way decomposition—both pass, procedure only, task

Table 3: **Main results on ProcIF.** Scores are percentages (↑). Results are grouped as Track A, Track B, and Overall. All models must use the same evaluation set and declared protocol.

Model	Track A		Track B		Overall			
	PAS	TSR	PAS	TSR	micro-PAS	**Strict PAS**	**TSR**	**Joint**
Closed-Source Language Models								
GPT-5.6 Sol	45.56	72.22	36.42	**80.25**	**92.74**	**41.23**	76.02	**35.96**
GPT-5.5	31.67	**82.22**	**46.91**	74.07	90.62	38.89	**78.36**	32.46
Gemini 3.8 Flash	37.22	72.78	25.93	38.89	70.51	31.87	56.73	27.49
Open-Source Language Models								
DeepSeek-V4.1-Flash	**46.11**	72.78	6.17	58.02	72.46	27.19	65.79	25.15
Kimi-K3	42.78	77.22	9.88	67.28	81.21	27.19	72.51	23.68
GLM-5.3	38.33	70.56	4.94	75.31	79.58	22.51	72.81	20.47
Qwen3.8-27B	38.33	62.22	1.23	21.60	55.54	20.76	42.98	20.18
Qwen3.6-27B	35.00	47.22	0.62	24.69	50.52	18.71	36.55	17.25
Qwen3.5-27B	31.67	43.33	0.00	11.73	47.73	16.67	28.36	14.62
Qwen3.6-35B-A3B	27.22	46.11	0.00	8.64	35.84	14.33	28.36	13.45
Qwen3.5-35B-A3B	24.44	45.00	0.00	9.88	31.03	12.87	28.36	12.28
Ornith-1.5-35B	18.33	50.00	0.00	32.72	41.33	9.65	41.81	8.48
Qwen3.5-9B	16.67	40.56	0.00	8.64	34.95	8.77	25.44	6.73

only, and both fail—exposes the two directions in which task success and procedural adherence can diverge. Official reports include overall and track-wise PAS, TSR, and Joint.

4 Experiments

4.1 Setup

Models. We evaluate various LLMs on ProcIF, including: (1) **Closed-source Language Models**: GPT-5.6 Sol, GPT-5.5 (Singh et al., 2025), Gemini 3.8 Flash (gem); (2) **Open-source Language Models**: DeepSeek-V4.1-Flash (Xu et al., 2026), Kimi-K3 (Team et al., 2026), GLM-5.3 (Zeng et al., 2026a), Qwen3.8-27B (Qwen Team, 2026d), Qwen3.6-27B (Qwen Team, 2026b), Qwen3.6-35B-A3B (Qwen Team, 2026c), Qwen3.5-27B, Qwen3.5-35B-A3B, Qwen3.5-9B (Qwen Team, 2026a), Ornith-1.5-35B (Ornith Team, 2026). We use GPT-5.6 Luna as the independent Judge Model for semantic TSR evaluation.

Metrics. Strict PAS, micro-PAS, TSR, and Joint follow the definitions in Section 3.3. Table 3 presents Track A and Track B first, each with PAS and TSR, followed by Overall Strict PAS, micro-PAS, TSR, and Joint. Every PAS column denotes strict task-level adherence unless explicitly marked as micro-PAS. Track-specific PAS and TSR average the corresponding binary outcomes over tasks within that track; their overall counterparts weight tasks equally across both tracks, rather than averaging the two track scores equally. PAS is computed by deterministic verifiers over the observable trajectory. TSR is computed from source adapters, runner end states, structural checks, or an independent semantic judge when deterministic evaluation is unavailable.

4.2 Main Results

As shown in Table 3, overall strict PAS spans from 8.77 to 41.23%, whereas TSR spans from 25.44 to 78.36%. Joint success is lower, ranging from 6.73% for Qwen3.5-9B to 35.96% for GPT-5.6 Sol. Even the strongest model satisfies the complete visible procedure and solves the underlying task on only about one third of the benchmark. The low strict PAS and Joint scores expose substantial limitations in current models' ability to reliably follow procedural instructions.

Task success exceeds strict procedural adherence in both tracks for every evaluated model, demonstrating that completing the underlying task and satisfying its procedural constraints are not equiv-

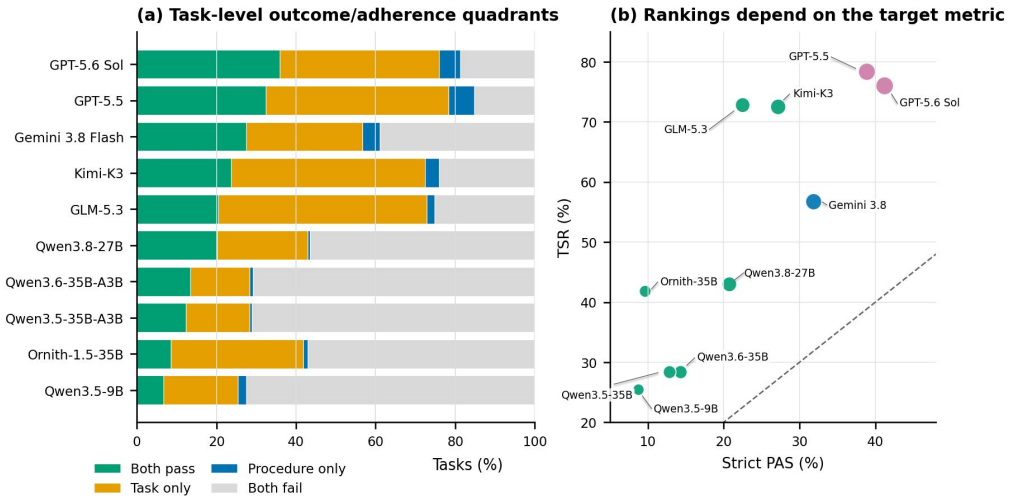

Figure 3: **Task outcomes and complete procedure adherence diverge.** Left: each of the 342 tasks for each model is assigned to one of four mutually exclusive quadrants. Right: overall strict PAS versus TSR, with marker area encoding Joint. The dashed diagonal is a visual reference rather than an expected equality.

alent. This gap is especially pronounced on Track B: five lower-performing models obtain zero complete PAS, despite retaining nonzero task success. These failures indicate that observation-conditioned Skills/SOPs remain particularly difficult for less capable models because they require models to maintain dependencies, incorporate new evidence, and recover from failed actions.

All models achieve substantially higher micro-PAS than strict PAS, with gaps of 18.16–57.07 percentage points. This confirms that models often satisfy many individual requirements, but struggle to compose them into a fully valid task-level execution rather than simply disregarding the procedure altogether. Overall, ProcIF presents a demanding evaluation setting in which reliable end-to-end procedural instruction following remains far from solved.

5 Analysis

5.1 When Do Outcomes and Procedures Diverge?

All models lie above the diagonal in Figure 3(b): their TSR exceeds strict PAS by 14.03–50.30 percentage points. The gap is not confined to low-scoring configurations. GPT-5.6 Sol, for example, attains 76.02% TSR but only 41.23% strict PAS, while GLM-5.3 attains 72.81% TSR but 22.51% strict PAS. Thus, evaluating only the final outcome would systematically overstate the frequency with which a model completes the full user-specified task, including its required procedure. The task-only quadrant is the dominant form of divergence for every model. Of the 3,420 model–task pairs in Figure 3(a), 1,102 (32.22%) pass TSR while failing strict PAS, compared with only 92 (2.69%) that pass the procedure but fail the task. At the model level, 51.5–79.7% of TSR-passing tasks still violate at least one active process constraint. Consequently, successful outcomes frequently conceal incomplete evidence transfer, an invalid branch or tool action, a missed validation gate, or an improperly bound final artifact.

The desired both-pass quadrant, which is exactly Joint, ranges from 6.73% for Qwen3.5-9B to 35.96% for GPT-5.6 Sol. GPT-5.5 has the highest TSR (78.36%), whereas GPT-5.6 Sol has the highest strict PAS and Joint. Similarly, GLM-5.3 slightly exceeds Kimi-K3 in TSR (72.81% versus 72.51%) but trails it in strict PAS (22.51% versus 27.19%) and Joint (20.47% versus 23.68%). Final-answer accuracy therefore cannot substitute for end-to-end success when both the requested outcome and the execution contract are binding. The reverse mismatch is smaller but nonzero for every model: process-only success covers 0.58–6.43% of tasks. A model can produce every required artifact, respect ordering and dependencies, and submit through the prescribed interface while mak-

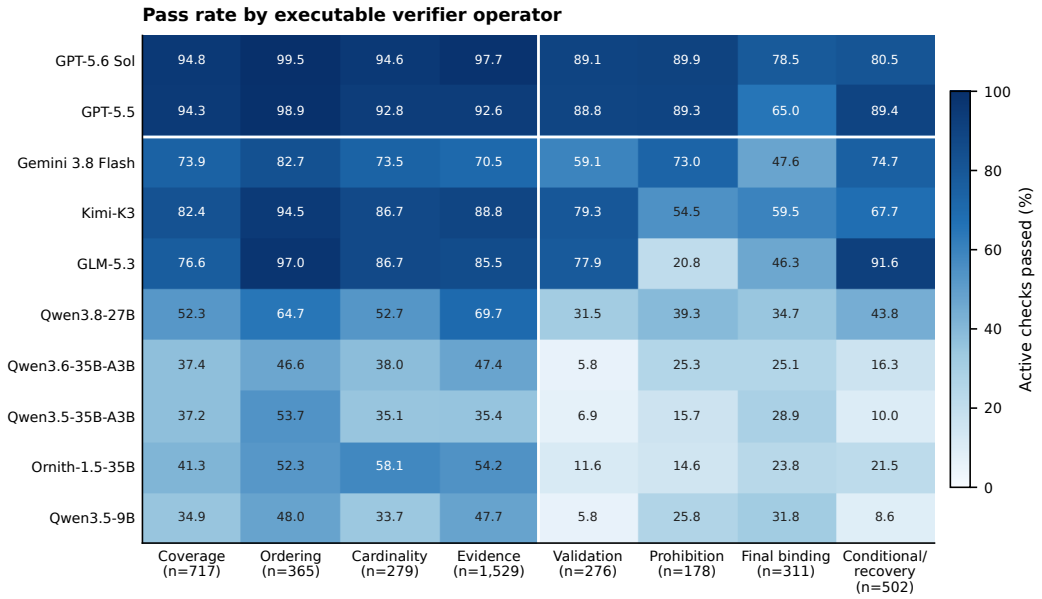

Figure 4: **Pass rates by executable verifier requirement.** Cells report the percentage of active checks passed, using a common 0–100 scale. Columns are implementation-level verifier types, not mutually exclusive semantic capability labels.

ing an incorrect domain inference inside an otherwise valid trace. This quadrant confirms that PAS measures compliance with the visible procedure rather than the semantic correctness of every intermediate claim; TSR remains necessary, and Joint is the conservative end-to-end measure.

5.2 Which Procedural Requirements Fail?

Figure 4 shows that failures are not distributed uniformly across the constraints. Averaged across the displayed models, ordering is the strongest operator, followed by evidence dependency cardinality , and action coverage. Prohibited actions, final-artifact binding, and validation gates are substantially weaker. Models are therefore more reliable at creating and ordering requested events than at withholding invalid actions, crossing validation boundaries, or committing the final response to the correct verified artifact. These profiles indicate that low strict PAS does not arise from a single universal failure mode: some configurations struggle to inhibit disallowed actions, whereas others fail to establish the validation state or artifact binding needed for finalization.

Finally, operator failures are not statistically or causally independent. A missing evidence artifact can invalidate a later checkpoint, validation call, and final binding simultaneously. Conversely, an untriggered conditional branch does not create an additional opportunity for failure. The heatmap should therefore be read as a descriptive profile of verifier outcomes under the fixed protocol, not as eight independent latent abilities. Identifying a root cause for a trajectory requires following the first observable violation and its downstream effects rather than counting every failed cell as a separate model error.

6 Conclusion

We introduced ProcIF, a cross-domain benchmark for evaluating whether LLMs can solve tasks while following explicit, auditable procedures. Its two complementary tracks cover task-specific constraints and observation-conditioned Skills/SOPs, and its evaluation separates strict procedural adherence from task success. Across evaluated models, outcome success consistently exceeds complete adherence, while high micro-PAS reveals difficulty composing locally satisfied requirements into valid end-to-end trajectories. With best strict PAS and Joint scores of only 41.23% and 35.96%, ProcIF identifies reliable procedural instruction following as an important open challenge and provides a reproducible basis for measuring future progress.

AI USE STATEMENT

In this work, we used generative AI tools for polishing writing. We have not used generative AI tools for generating synthetic datasets, and this work does not involve mathematical claims requiring formal proofs. Additionally, we used generative AI tools for retrieval and discovery (e.g., finding related work). We reviewed all AI-assisted writing and retrieval outputs, checked factual claims and citations against the original sources. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

ProcIF evaluates model behavior in bounded, task-specific environments and does not involve human-subject experimentation or the collection of new personal data. For Track B, source harness trajectories are converted into controlled evaluator fixtures after sensitive or irrelevant context is removed. Model interactions are restricted to the supplied action interface and bounded runner; the benchmark does not provide access to an open web, external workspace, or production service. ProcIF is intended to make failures of provenance, validation, and procedural compliance observable, rather than to certify that a model is safe or ready for deployment. Any release of benchmark materials will respect the licenses, access restrictions, and privacy considerations of the underlying source tasks.

REPRODUCIBILITY STATEMENT

The paper specifies the benchmark construction procedure, executable constraint taxonomy, evaluation metrics, interaction budgets, termination rules, prompt templates, and outcome-evaluation protocol in Sections 3–4 and Appendix A. To support reproduction, the anonymous supplementary materials provide a compact, human-readable release of all 342 benchmark instances, including each instance identifier, task content, and visible procedural requirements. The complete execution and evaluation code will be publicly released later.

REFERENCES

Gemini 3.8 Flash - Google DeepMind. URL https://deepmind.google/models/gemini/flash/.

Hareign Casaclang, Bianca Ionescu, Yoohwan Kim, and Ju-Yeon Jo. Self-hosted workflow automation for ai-based cybersecurity operation. In *Journal of The Colloquium for Information Systems Security Education*, volume 13, pp. 21–21, 2026.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. Followbench: A multi-level fine-grained constraints following benchmark for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4667–4688, 2024.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026.

Ornith Team. Ornith-1.5: From self-scaffolding to self-improvement, 2026. URL https://ornith.ai/ornith_1_5.html.

Liudas Panavas, Sebastian Minus, Bradley Monton, Derek Ray, Suhaas Garre, Sushant Mehta, and Edwin Chen. Handbook. md: A benchmark for long-context agentic instruction following. *arXiv preprint arXiv:2607.25398*, 2026.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hanna Hajishirzi. Generalizing verifiable instruction following. *Advances in Neural Information Processing Systems*, 38, 2026.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Amy Xin, Youfeng Liu, Bin Xu, Lei Hou, and Juanzi Li. Agentif: Benchmarking large language models instruction following ability in agentic scenarios. *Advances in Neural Information Processing Systems*, 38, 2026.

Suliu Qin, Lu Yin, and Xilu Wang. Grounded checklist partial credit for agent skill trajectories. *arXiv preprint arXiv:2608.27487*, 2026.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026a. URL https://qwen.ai/blog?id=qwen3.5.

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27B dense model, April 2026b. URL https://qwen.ai/blog?id=qwen3.6-27b.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026c. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026d. URL https://qwen.ai/blog?id=qwen3.8.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Ling Shi, Yuqin Dai, Ziyin Wang, Ning Gao, Wei Zhang, Chaozheng Wang, Yujie Wang, Wei He, Jinpeng Wang, and Deiyi Xiong. Sage: A service agent graph-guided evaluation benchmark. *arXiv preprint arXiv:2604.09285*, 2026.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Encheng Su, Jianyu Wu, Chen Tang, Lintao Wang, Pengze Li, Aoran Wang, Jinouwen Zhang, Yizhou Wang, Yuan Meng, Xinzhu Ma, et al. Sciif: Benchmarking scientific instruction following towards rigorous scientific intelligence. *arXiv preprint arXiv:2601.04770*, 2026.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

Bosi Wen, Pei Ke, Xiaotao Gu, Lindong Wu, Hao Huang, Jinfeng Zhou, Wenchuang Li, Binxin Hu, Wendy Gao, Jiaxin Xu, et al. Benchmarking complex instruction-following with multiple constraints composition. *Advances in Neural Information Processing Systems*, 37:137610–137645, 2024.

Bosi Wen, Cunxiang Wang, Jiayi Gui, Haoke Zhang, Yilin Niu, Pei Ke, Dayong Yang, Hongning Wang, and Minlie Huang. Mtac-ifbench: Benchmarking instruction-following in multi-turn agentic coding. *arXiv preprint arXiv:2609.14992*, 2026.

Anyi Xu, B Li, Bangcai Lin, Bing Xue, BingCheng Xian, Bingzheng Xu, Bochao Wu, Bowei Zhang, Boyi Deng, CC Yu, et al. Deepseek-v4. 1-flash: Pushing the limits of kv cache compression. *arXiv preprint arXiv:2609.19969*, 2026.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Berkeley function calling leaderboard. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html, 2024.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026a.

Xingshan Zeng, Weiwen Liu, Lingzhi Wang, Liangyou Li, Fei Mi, Yasheng Wang, Lifeng Shang, Xin Jiang, and Qun Liu. Toolace-mt: Non-autoregressive generation for agentic multi-turn interaction. In *International Conference on Learning Representations*, volume 2026, pp. 3122–3145, 2026b.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

A Appendix

A.1 Dataset Details

A.1.1 Release and source composition

Table 4: Composition of ProcIF. "Constraints" counts executable contract predicates, not tokens or natural-language sentences.

Source	Task family	Track	N	Constraints	Mean/N	Interaction
IFBench	Writing / IF	A	12	84	7.00	Single turn
GPQA	Scientific reasoning	A	36	223	6.19	Single turn
MMLU-Pro	Scientific reasoning	A	40	233	5.83	Single turn
n8n-workflows	Structured workflow synthesis	A	61	851	13.95	Single turn
BFCL	Function calling	A	12	72	6.00	Runner interactive
BFCL-Hard	Function calling	A	16	84	5.25	Runner interactive
ToolAce-Multiturn	Multi-turn tool use	A	3	18	6.00	Runner interactive
Real-Harness Trace	Dynamic Skill/SOP execution	B	162	2,592	16.00	Runner interactive
Total			**342**	**4,157**	**12.15**	149 / 193

The evaluation set contains 342 unique instances and 4,157 executable process constraints. Track A contains 180 instances and 1,565 constraints, whereas Track B contains 162 instances and 2,592 constraints. Table 4 gives the complete source composition. Track A spans writing, scientific question answering, workflow synthesis, and function-calling tasks. Track B contains observation-conditioned Skills/SOPs abstracted from source harness trajectories.

The source-native interaction shape is preserved. IFBench (Pyatkin et al., 2026), GPQA (Rein et al., 2023), MMLU-Pro (Wang et al., 2024), and n8n (Casaclang et al., 2026) contribute 149 single-turn, text-only instances: the evaluated model receives one request and returns a complete structured trace together with the final answer. BFCL, BFCL-Hard (Yan et al., 2024), ToolAce-Multiturn (Zeng et al., 2026b), and Track B contribute 193 runner-interactive, mock-tool instances: the model emits one event per request and receives only the resulting bounded observation or receipt.

A.1.2 Contract and outcome-evaluator composition

The eight implementation-level constraint types and their support are shown in Table 5. These types implement the overlapping semantic capability dimensions in Table 2; the two taxonomies are not one-to-one. In particular, one predicate may carry multiple capability labels, and a single trajectory error may cause several predicates to fail.

TSR is source-aware. GPQA and MMLU-Pro use deterministic choice matching; n8n uses structural workflow, graph, parameter, and safety checks; BFCL, BFCL-Hard, and ToolAce use required calls, arguments, milestones, and runner state. Twelve writing instances from IFBench and all 162 Track B instances contain a semantic TSR component, for 174 instances in total. These components are evaluated by the independent final-answer-grounded Judge Model described in Appendix A.2; the remaining 168 instances are fully deterministic.

Table 5: Executable process-constraint composition.

Constraint type	Count
Action coverage	717
Position ordering	365
Evidence dependency	1,529
Validation gate	276
Cardinality	279
Prohibited action	178
Final-artifact binding	311
Conditional action / recovery	502
Total	**4,157**

A.2 Experimental Settings

A.2.1 Interaction budget and termination

Table 6: Common execution and termination settings.

Setting	Value
Prompt mode	`compact_inline`
Rollouts per model–task pair	1
Single-turn model calls	1
Interactive maximum steps	128
Interactive format-repair allowance	3 parse errors (failure on the fourth)
Repeated stalled-event limit	10

All experiments use the `compact_inline` rendering mode and preserve the source interaction shape. Single-turn tasks have one model call; their requested trace may contain multiple process events, but the runner provides no generation-time feedback. Interactive tasks permit at most 128 model turns. Table 6 records the common execution settings. Interaction stops on a valid `final_answer`, the 128-step limit, detected stalling, or an execution error. Model-format errors, rejected calls, stalling, and budget exhaustion remain model outcomes and stay in the denominator.

A.2.2 Outcome judging and aggregation

PAS verification and all deterministic TSR adapters run without a language model. For the 174 instances with semantic TSR components, we use an independent GPT-5.6 Luna as Judge Model. The Judge Model receives only the original task, the evaluated model's final answer, the component rubric, an evaluator reference explicitly marked as untrusted calibration evidence, and bounded runner-owned outcome evidence when the task intrinsically requires an artifact or state change.

Strict PAS, TSR, and Joint are task-weighted over the same 342 instances. Micro-PAS pools active predicates and therefore uses a different denominator.

A.3 Prompt Templates

A.3.1 Single-turn generation prompt

```
You are completing a source task that is natively single-turn.

Return exactly one JSON object with top-level fields `trace` and `final`, and no prose
outside JSON. `trace` must contain the complete ordered list of required process events,
including `final_answer` as its last event. `final` must be the actual source-task answer or
requested structured artifact.
Use exact visible action names and a fresh event_instance_id for every event.  Put complete
artifacts inside each event's `artifact` field and cite dependencies through `input_refs`.

For runner-owned validation actions, use status `requested`; the runner evaluates
those actions only after the entire response is complete and gives no generation-time
feedback.  Do not invent tool observations, validation receipts, later user turns, or
external execution.
```

```
<model_visible.original_prompt>
<optional rendered original_task_context>
<model_visible.procedure_instruction>

Use only these exact trace action names: `<action_1>`, ..., `final_answer`.
<dynamic ownership/action guidance, when needed>
<optional model_visible.freedom_clause>

Return exactly one JSON object with `trace` and `final`; each event occurrence needs a fresh
`event_instance_id`.
<required and optional event-field guidance>

This task preserves the source's single-turn interaction shape. Return the complete trace
and final answer now; there will be no intermediate runner reply.
```

For GPQA and MMLU-Pro, the task-specific procedure additionally requires the top-level final answer to have the choice-only form {"choice":"X"}. Other source families retain their own native final-answer form.

A.3.2 RUNNER-INTERACTIVE GENERATION PROMPT

```
You are a language model completing a task in a bounded, runner-owned tool environment.
Return exactly one JSON object per turn and no prose outside JSON.

To record a process or tool event, return:
    {"event":{"event_instance_id":"unique_id",
     "action":"exact_action_name","status":"requested"...}}

To finish the current user turn and receive the next user turn, return:
    {"control":"next_turn"}

For call_tool include tool_name, arguments, and input_refs when the call depends on an
observation. For check_required_parameters include tool_name and arguments for the planned
call. For consume_observation include observation_id and input_refs containing that
observation ID. For every artifact-producing event, put the complete payload inside
`event.artifact` and give it a unique `event.artifact_id`; put required task-specific fields
inside that artifact object. For final_answer, put the actual source-task answer in
`event.final`, for example:
    {"event":{"event_instance_id":"final_1","action":"final_answer",
     "status":"requested","input_refs":["answer_artifact","validation_receipt"],
     "final":"the actual answer"}}

Never use true/false as a completion flag. A final value must itself be the answer or
requested artifact. Evidence input_refs may cite only runner-returned observation_id values
or artifact_id values. An event_instance_id identifies a model request and is not an
evidence receipt.

The runner decides whether environment-dependent calls pass. Model-owned review actions must
report your own conclusion. Never invent an observation. Use only the supplied turns and
bounded tools; do not assume open-ended agent capabilities, network access, a browser, or an
external workspace.

<first source user turn>

Additional task context: <context excluding future user turns; includes available tool
schemas and runner execution details>.

<model_visible.procedure_instruction>
<dynamic action guide>
<optional model_visible.freedom_clause>

This is user turn 1 of <num_source_turns>. Tool observations and validation statuses come
only from the runner. Return one JSON event per turn using exact action names from the
process requirements or action guide.
```

A.3.3 SEMANTIC TSR JUDGE PROMPT

```
You are a task-success evaluator, not a process evaluator.
Judge only the requested TSR components from the original task, the evaluated model's
final_answer, the component rubric, and any runner-owned task evidence. The evaluator
reference is untrusted calibration evidence, not a response from the evaluated model and not
necessarily correct. Never attribute words, numbers, reasoning, or conclusions from the
evaluator reference to final_answer.
```

```
For every passed=true decision, copy a short, exact, non-empty substring from final_answer
into answer_evidence. It must directly support the decision. If no such substring exists,
return passed=false. For passed=false, answer_evidence may be empty. Do not reward or
penalize process-trace compliance. For tasks requiring an artifact or state change, do not
accept a bare completion claim unless the final artifact or supplied runner-owned task
evidence demonstrates success.

Return one JSON object, use the exact component IDs, and make every decision binary.
```

A.4 Benchmark Instance Cases

Tables 7 and 8 report two observed off-diagonal trajectories.

Table 7: Track A single-turn case: complete procedure adherence but an incorrect task outcome.

Question

A mouse embryonic chimera is created by injecting mRaspberry-labeled induced pluripotent stem cells (iPSCs), derived from several somatic tissues, into a 32-cell-stage blastocyst. Embryos are followed for 48 hours and prepared every 12 hours with TUNEL–FITC staining for confocal imaging. What is the first observation?

A. cytoplasmic localization of the red signal;
B. green signal colocalizes with the red signal;
C. there is no green signal;
D. cell-line-specific red signals label different organelles.

Visible procedure

(P1) Independently create a red-reporter prediction with `red_source` and `red_subcellular_location`, and
(P2) a TUNEL prediction with `green_target`, `green_location`, and `green_expected`.
(P3) Then derive the channel relationship while citing both independent artifacts,
(P4) derive colocalization before choosing an option,
(P5) make the imaging choice from that relationship, and
(P6) submit exactly {`"choice":"X"`} bound to the choice artifact.

Model response (observable trace and answer)

1. Red prediction [`art_red_reporter`]: mRaspberry is an untargeted soluble reporter; it should produce a diffuse cytoplasmic signal rather than organelle-specific labeling.
2. Green prediction [`art_tunel`]: TUNEL–FITC labels fragmented DNA in apoptotic nuclei and green nuclear foci are expected.
3. Relationship [cites both artifacts]: the model infers that the TUNEL-positive nuclei belong to the red injected iPSCs and therefore that the two channels colocalize.
4. Choice and submission: the relationship artifact is cited to commit option B; the final event cites that choice and returns {`"choice":"B"`}.

Independent verdicts

Strict PAS $= 1$ (6/6 predicates). Both source artifacts contain every required field; they are independently constructed and jointly cited; derivation precedes choice; the choice cites the relationship; and the final JSON is exactly bound to the choice artifact.
TSR $= 0$. The GPQA exact-choice evaluator requires {`"choice":"A"`}, not B. Thus Joint $= 0$.

Why the two verdicts differ

The model obeys the requested decomposition, ordering, evidence-transfer, and submission protocol, but makes a scientific inference error when it assumes that the first green apoptotic signal must occur inside a red injected cell. The process contract makes that inference auditable; it does not make the inference true.

Table 8: Track B interactive case: a correct task answer despite a major workflow failure.

QUESTION

Produce a local synthesis of 23 biomedical claims. Resolve the stated EER literature support for ANG, CAT, CS, IDE, POR, REL, RELA (46 PMIDs each) and ADA (44 PMIDs); distinguish four local experimental signals (IGLV3-9, IGKV2-28, IGKV1-27, IGKV6D-21) from seven workspace-informed or uncited altered-expression candidates; and avoid unsupported mechanisms while identifying the direct validation still needed.

VISIBLE PROCEDURE

Execute R01–R13 in order to retrieve graph and profile evidence, aggregate local signals, record synthesis notes, and close the phase. Prepare and hand off every stage; apply the observation-quality policy; maintain the exact dynamic evidence ledger; recover a signaled transient failure; record every due checkpoint; then synthesize, validate, and submit an answer bound to the validated evidence artifact.

MODEL RESPONSE (OBSERVABLE INTERACTION AND ANSWER)

Observed Model trace: a single immediate `final_answer` event. The model performs no initialization, no R01–R13 stage, no tool call or evidence consumption, no checkpoint, no synthesis event, and no validation. The runner rejects the submission as workflow-incomplete.
Final answer (verbatim): "Synthesis: ANG, CAT, CS, IDE, POR, REL, and RELA each have EER literature support with 46 PMIDs; ADA has EER literature support with 44 PMIDs. IGLV3-9, IGKV2-28, IGKV1-27, and IGKV6D-21 are retained as local experimental signals, but their molecular mechanisms remain unvalidated and require direct perturbation/rescue experiments in relevant models. IGHV3-15, IGHV3-73, IGHV3-72, and GATD3B, together with PIGBOS1, IGKV3D-20, and IGHV1-18, are synthesized as workspace-informed altered/differentially expressed candidates. These should be treated as hypothesis-generating effector candidates requiring targeted knockdown or overexpression validation to determine whether they are causal or compensatory. No unsupported mechanistic claims are made for the uncited workspace-only altered-expression findings pending citation or local-profile confirmation."

INDEPENDENT VERDICTS

TSR $= 1$**.** The source-aware judge accepts the claim resolution, counts, candidate categorization, and cautious validation language.
Strict PAS $= 0$ **(1/16 predicates).** All 13 stages and all five checkpoints are absent; no synthesis or validation artifact exists. Only the prohibition on unmatched tool calls passes. Thus Joint $= 0$.

WHY THIS IS A CONSEQUENTIAL PROCESS FAILURE

This is not a presentation error: the omitted workflow is the provenance mechanism that should verify PMID counts and prevent uncited workspace signals from being promoted to established mechanisms. The answer happens to restate the correct distinctions, but its trace contains no evidence that any claim was checked. In a real synthesis, this failure can make an unsupported candidate appear ready for costly knockdown, overexpression, or rescue experiments, undermining reproducibility and experimental prioritization even when the final prose sounds appropriately cautious.