Auditable Evidence Trails for Pedagogy-Grounded LLM Judging

Anonymous ACL submission

Abstract

Large language models have substantial latent STEM knowledge, but educational marking also requires local pedagogical context: what was taught, what evidence counts, which shortcuts should not receive credit, and how partial credit should be assigned. Most LLM grading pipelines expose only a final score and a short justification, making it difficult for a rubric author to determine whether the model has actually followed their marking standards. We propose a pedagogy-grounded judging framework that makes this context explicit and auditable. For each submitted artefact, a rubric-construction agent first builds a response-specific sub-rubric anchored to the instructor's high-level rubric and course policies, without assigning marks. A separate marking agent then scores that sub-rubric and emits an evidence trail: per-subcriterion marks, maxima, source spans, rationales, and review flags. An orchestrator verifies the trail using source-presence, score-bound, attribution, and incomplete-response checks, then aggregates marks and deterministic tests. The same trace supports a lecturer-as-builder workflow in which the rubric author inspects failures, encodes targeted prompt, rubric, or verifier changes, and re-runs a calibration set. Agreement metrics serve as sanity checks rather than the main claim: on ASAP-SAS the pipeline reaches 66–91% of human inter-rater agreement across five prompts, and on SAS-Bench it obtains QWK ≥ 0.70 on 10 of 12 held-out buckets. More importantly, evidence trails reveal otherwise hidden failures, including fabricated or unsupported evidence, credit for instructor-provided starter code, methods that pass mechanical tests but fail under closer artefact inspection, invalid subcriterion maxima, and cohort-scale patterns invisible to per-script review.

1 Introduction

LLM judges are increasingly used to evaluate open-ended text, from preference judgements (Zheng et al., 2023) to educational assessment (Kortemeyer, 2023; Xiao et al., 2025). The common implementation is a single inference call: provide a rubric and a response, ask for a score and a justification, and treat the result as the judgement. This is attractive because modern LLMs often know the relevant domain content. For STEM assessment, however, domain knowledge is not enough. A marker must also know the local pedagogy: what was emphasised in lectures, which notational conventions were taught, what evidence is acceptable, when missing submitted artefacts matter, and which partial-credit distinctions the lecturer intends to preserve.

The central risk is therefore not only that an LLM may assign the wrong score. It is that the system may appear plausible while silently applying a different marking logic from the one intended by the rubric author. A final score and generic explanation do not show whether the model credited instructor-provided starter material, mistook a claim for evidence, ignored a course-specific policy, or stretched a rubric criterion beyond its intended scope.

This paper treats rubric-conditioned LLM judging as a problem of *pedagogical operationalisation*. The goal is not to replace expert judgement with a fixed scoring machine. It is to let the rubric author build their expertise into the judging process, and then obtain an audit log showing whether that expertise was followed. We propose a two-stage framework. First, a rubric-construction agent reads the submitted artefact and constructs a response-specific sub-rubric anchored to the high-level rubric, course guidance, and calibration examples. This step records how the rubric is being interpreted for this particular response, but assigns no

marks. Second, a separate marking agent applies the sub-rubric and emits an evidence trail: sub-criterion labels, maxima, assigned marks, quoted or otherwise source-grounded evidence, rationales, and review flags. An orchestrator runs the overall workflow, recomputes marks from the structured trace, verifies quoted evidence, checks score bounds, applies deterministic tests where available, and records cases requiring human review.

The evidence trail is the key artefact. It is not merely an explanation appended to a score; it is a structured intermediate representation that can be inspected, queried, and revised. In a postgraduate programming coursework, evidence-trail inspection surfaces failures in source grounding, rubric adherence, and pedagogical fit that final scores would not reveal. In public short-answer benchmarks, the same traces revealed scoring-rule defects and valid off-rubric answers. These observations become inputs to a refinement loop: the rubric author updates prompts, rubric clauses, or verifier checks, then re-runs the same calibration set to determine whether the failure has been resolved.

We evaluate the framework in three settings: a real postgraduate programming coursework with deterministic tests and judgement-heavy code review; ASAP-SAS short-answer scoring prompts; and SAS-Bench step-level grading tasks. The public benchmarks are used to show that the evidence-verifiable structure is not a one-off course implementation and does not collapse agreement with human scores. The main contribution, however, is methodological: the framework exposes the otherwise hidden path from instructor intent to model judgement. This shifts the evaluation target from final-score agreement alone to auditable and repairable judgement: a system must not only produce a plausible mark, but expose whether that mark was grounded in student-authored evidence, bounded by the rubric, and consistent with the lecturer's intended marking logic.

Contributions. We make three contributions. First, we introduce a two-stage, rubric-conditioned judging framework that separates response-specific rubric interpretation from score assignment and records an evidence trail for each marking decision. Second, we define a lecturer-as-builder calibration workflow in which pedagogical standards are encoded as prompts, rubric policies, deterministic checks, and verifier rules, then audited through evidence trails. Third, we report deployment-observed failure modes and refinements across a live postgraduate coursework deployment and public benchmarks, showing how structured traces make deployment failures localisable and correctable without retraining.

2 Related Work

LLM-as-judge and educational scoring. LLM-as-judge methods have been applied to preference evaluation and generation assessment, with work such as MT-Bench, Chatbot Arena, and G-Eval showing high aggregate alignment with human judgements while also exposing biases and reliability concerns (Zheng et al., 2023; Liu et al., 2023). Surveys and benchmark studies emphasise that aggregate agreement is insufficient without checks for consistency, bias, reproducibility, and grounding (Gu et al., 2026; Bavaresco et al., 2025). In education, LLMs have been studied as formative feedback systems, grading assistants, and co-grading tools (Kortemeyer, 2023; Kortemeyer et al., 2024; Xiao et al., 2025). Closest to our setting, AutoSCORE decomposes automated scoring into structured component recognition followed by scoring (Wang et al., 2026). Our framework shares the intuition that interpretation should be separated from scoring, but differs in its response-specific sub-rubrics and its emphasis on deployment-time audit and refinement rather than benchmark optimisation alone. More broadly, decomposing judgement into an explicit intermediate step before scoring relates to plan-then-execute strategies in LLM reasoning (Wei et al., 2022), though our intermediate artefact is a structured sub-rubric rather than a reasoning chain.

Rubric interpretation and pedagogy. Educational measurement work motivates our response-specific treatment of rubrics. Sadler (2009) argues that preset criteria are indeterminate in use: expert markers must interpret them in the context of particular responses. Bloxham et al. (2016) similarly shows that assessment criteria do not remove judgement, but relocate it into situated interpretation. This matters for LLM judging because a prompt containing a high-level rubric does not by itself guarantee that the model applies the lecturer's course-specific interpretation. We make that interpretation explicit by asking the model to produce a sub-rubric before any marks are assigned.

Evidence grounding. Hallucination is a known failure mode of generation systems (Ji et al., 2023). In rubric-conditioned judging, hallucination can enter not only through the final score but through the evidence used to justify it. Requiring a justification is therefore insufficient unless the cited evidence can be checked. Unlike factuality verification in open-domain generation, our setting has a closed source: the submitted artefact is the only valid evidence base, making source-presence checking both feasible and sufficient for the mechanical layer. Our framework verifies source presence and arithmetic consistency mechanically, while leaving full semantic entailment between evidence and mark as an explicit limitation.

3 Framework

Figure 1 gives an overview of the framework. The orchestrator runs the outer workflow: it collects artefacts, dispatches agents, runs deterministic checks, verifies evidence trails, aggregates marks, and records review flags. The submitted input is an *artefact bundle*, not always a single "student response": in our settings it can include answer text, code files, notebooks, test reports, commit logs, or linked repository material. The evidence trail is tied to the sub-rubric application: every judgement-heavy subcriterion receives its own mark, maximum, source span, rationale, and flags.

3.1 Instructor context as the anchor

The examiner supplies a high-level rubric with criteria, maxima, and descriptions of what the assessment values. In our deployment this is augmented with course-specific guidance: how the topic was taught, which artefacts count as evidence, which starter files should not receive credit, how to treat missing repository material, and examples from a calibration set. We treat this context as an anchor and constraint, not a fixed template to be filled identically for every response.

This distinction is important. Modern LLMs may know the underlying STEM content, but they do not automatically know the lecturer's intended partial-credit scheme or pedagogical priorities. A response-specific sub-rubric makes the model's interpretation of the instructor context visible before evaluation begins.

3.2 Bespoke sub-rubric construction

For each artefact bundle, a rubric-construction agent reads the submitted material and emits a bespoke sub-rubric. Each subcriterion contains a label, a maximum mark, and a short description of what full credit would require for this response under the instructor context. No marks are assigned at this stage. The prohibition on scoring is deliberate: the first stage should expose how the rubric is being interpreted, rather than silently constructing criteria to rationalise a score.

3.3 Evidence trails and verifier checks

A separate marking agent receives the artefact bundle, instructor context, and sub-rubric. It assigns marks to each subcriterion and emits an evidence trail. The minimum record for a judgement-heavy subcriterion is

$$\langle \text{label}, \max, \text{mark}, \text{source span}, \text{rationale}, \text{flags} \rangle.$$

The source span may be a quotation from text, a code token or line range, a notebook output, or a structured result from a deterministic test. The orchestrator then verifies properties that do not require another LLM call: marks may not exceed declared maxima; quoted material must be present in the submitted source; source-dependent claims must point to the relevant artefact; and incomplete or valid-but-unmapped responses must be flagged for review rather than silently absorbed into ordinary scoring.

3.4 Hybrid deterministic and judgement-based marking

Some assessment components admit deterministic checks. In programming coursework, the orchestrator can run unit tests, inspect notebook execution state, or parse commit logs. Judgement-heavy criteria, such as code quality, documentation, packaging, explanation quality, or analysis of results, are passed to the agent pipeline. The final mark combines deterministic outputs with verified evidence-trail judgements, so deterministic correctness and instructor-grounded judgement remain separate in the audit log.

4 Lecturer-as-Builder Workflow

4.1 Calibration before deployment

The intended use is calibration before deployment. The rubric author reviews a small calibration set, runs the pipeline on the same responses, and inspects the sub-rubrics, evidence trails, verifier failures, and review flags. When a systematic failure appears, the author encodes a targeted change in

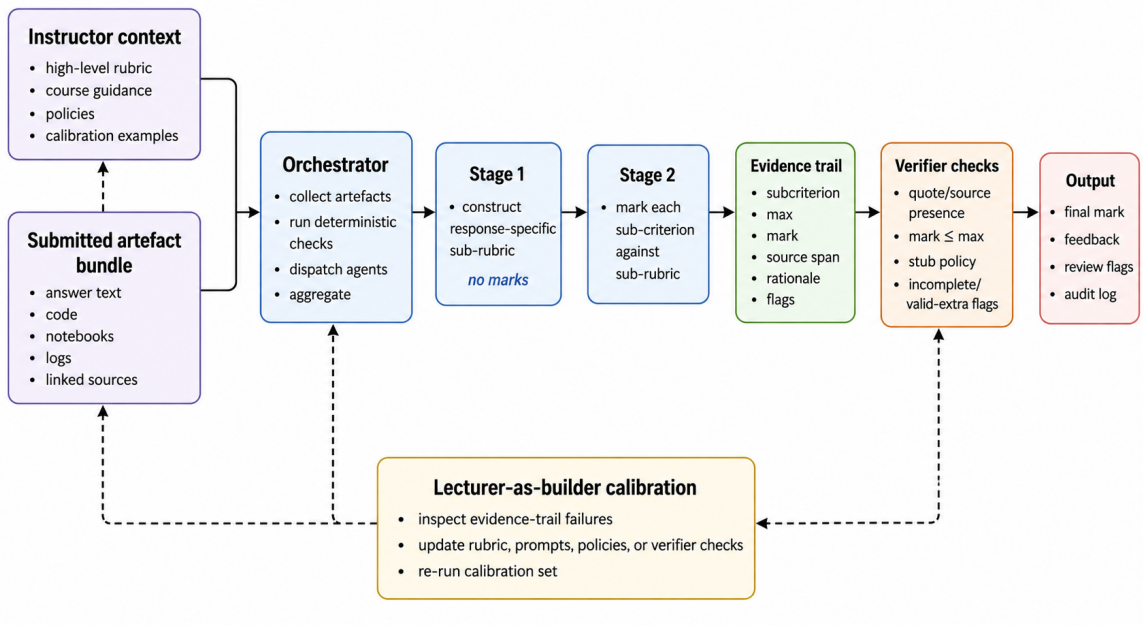

Figure 1: Pedagogy-grounded judging workflow. The lecturer or rubric author provides high-level pedagogical context and calibration examples, while submitted artefacts provide the evidence base for marking. A rubric-construction stage first derives a response-specific sub-rubric without assigning marks. A separate marking stage then applies the sub-rubric and produces an evidence trail containing sub-criterion marks, maxima, quoted evidence, and rationales. Rule-based validation checks the trail for source grounding, score bounds, incomplete responses, and domain-triggered policies. The dashed loop shows calibration before deployment: the rubric author inspects evidence-trail failures, refines prompts, rubric clauses, policies, or checks, and re-runs the calibration set before full-cohort deployment.

one of four places: the high-level rubric, the prompt guidance, the deterministic/orchestrator checks, or the verifier policy. The calibration set is then re-run to check whether the change repaired the failure without creating a new one.

This is the lecturer-as-builder loop. The lecturer does not merely write a rubric for a static automated marker. They progressively convert expertise into inspectable and testable machinery: examples, policies, caps, source requirements, and checks. The evidence trail then audits whether those components have been used.

4.2 What the evidence trail audits

The evidence trail supports three levels of audit. First, it audits *source grounding*: did the cited evidence actually appear in the student's artefact? Second, it audits *rubric adherence*: did each mark stay within its declared maximum and follow explicit policies such as starter-code exclusion? Third, it audits *pedagogical fit*: did the model interpret the response in a way consistent with how the lecturer intended the topic to be assessed? The first two levels can often be checked mechanically. The third requires human inspection, but the evidence trail makes the inspection local: the reviewer examines a subcriterion, its evidence, and the policy that produced it, rather than reverse-engineering a final score.

Table 1 summarises the current refinement catalogue. The set is not intended as a complete taxonomy of grading failures. Rather, it illustrates how deployment-observed failures can be converted into local, testable changes. Different courses will surface different domain-triggered policies while inheriting general checks such as quote verification and score-bound validation.

5 Evaluation Setup

We evaluate the framework in three settings chosen to stress different aspects of rubric-conditioned judging.

NPP coursework. The real-course deployment is a postgraduate Numerical Programming in Python coursework with ethics approval and informed consent from participants. Unlike the public benchmarks, this setting involves genuine student submissions assessed under institutional grading standards, providing an ecologically valid test of

ID	Level	Observed failure	Encoded change
R1	Evidence	Fabricated quoted spans	Source/quote verification
R2	Attribution	Credit for instructor starter material	Stub-inheritance check
R3	Policy	Implementation does not support what the artefact claims	Claim-vs-artefact verification
R4	Constraint	Mark exceeds declared maximum	Score-bound validation
R5	Review	Valid answer outside enumerated rubric	Valid-extra flag
R6	Review	Truncated answer treated as off-topic	Incomplete-response flag

Table 1: Deployment-observed refinements. R1 and R4 are mechanical checks over the evidence-trail structure. R2, R3, R5, and R6 encode assessment policies surfaced by calibration and deployment. None requires retraining.

whether the framework's audit mechanisms surface failures that matter in practice. The real-course deployment is a postgraduate Numerical Programming in Python coursework. Submissions are GitHub repositories containing Python packages, tests, documentation, packaging files, commit histories, and an executed research notebook. This setting tests the hybrid workflow: deterministic unit tests and notebook checks are combined with judgement-heavy code-quality and documentation criteria.

ASAP-SAS. ASAP-SAS (Riordan et al., 2017) provides public short-answer scoring prompts with different rubric forms, including item-counting, joint multi-part scoring, and holistic scoring. We use approximately 200 responses per prompt, stratified by human score, without labelled training examples. This setting tests whether the same two-stage evidence-trail structure transfers beyond the course deployment.

SAS-Bench. SAS-Bench (Lai et al., 2025) provides step-level or criterion-level grading tasks across multiple subject buckets. Maximum scores range from 5 to 34, making exact match hard to interpret across buckets. We therefore report quadratic weighted kappa, accuracy, mean absolute error, and evidence-trail quote-failure rates.

Metrics and interpretation. We evaluate the framework along two axes. The first is agreement: whether the pipeline remains a plausible marker relative to human scores. We report quadratic weighted kappa (QWK), accuracy, mean absolute error (MAE), and, where available, comparison with human inter-rater reliability. These metrics are treated as sanity checks rather than the primary claim. The second axis is audit utility: whether the evidence trail exposes local, reviewable, and repairable failures that a score-only pipeline would hide. We inspect quote-verification failures, score-bound violations, review flags, valid-extra cases, stub-inheritance cases, claim-vs-artefact mismatches, and before/after effects of targeted refinements. This reflects the paper's central claim: evidence trails should make judgement failures discoverable and correctable, rather than merely produce a competitive aggregate score.

6 Results

6.1 Agreement as a sanity check

Table 2 summarises the current public-benchmark results. On ASAP-SAS, the framework obtains mean QWK of 0.705 across five prompts, outperforming direct scoring on four of five prompts and reaching 66–91% of the human inter-rater agreement ceiling. On SAS-Bench, it reaches macro QWK of 0.810 and achieves QWK ≥ 0.70 on 10 of 12 held-out buckets. These results are not presented as a leaderboard claim. They show that the audit-oriented decomposition remains compatible with reasonable agreement across multiple rubric formats.

6.2 What the NPP evidence trails exposed

The NPP deployment illustrates why a score-only evaluation is insufficient. Four failure modes were visible only because the system produced subcriterion-level evidence: a verifier preventing the agent from being misled by aspirational comments (R1), credit-for-stub-material (R2), claim-vs-artefact mismatches (R3), and a cohort-scale test-suite gap.

Anti-fabrication verifier (R1). R1 is a verifier-stage check requiring every quoted token in the evidence trail to physically appear in the supplied source. In the NPP deployment R1 acts preventatively: by enforcing source-presence on each quote, the agent cannot credit method behaviours that appear in docstrings or comments but are absent from the implementation. This

Setting	n	Main metric	Range / detail	Audit signal
ASAP-SAS	1052	mean QWK 0.705	C3 outperforms direct scoring on 4/5 prompts; 66–91% of human inter-rater agreement	Post-hoc corrections recover +0.097 QWK from stored evidence fields (Appendix G)
SAS-Bench	427	macro QWK 0.810	10/12 held-out buckets reach QWK $\geq$ 0.70	2.8% span-level quote-verification failures under exact matching
NPP	61	total-mark MAE 7.11; median $\lvert\Delta\rvert$ 4.5	52% within 5 marks; 79% within 10 marks; 97% within one letter-grade band (38% exact-letter match — boundary-straddling cases dominate). Rigorously analysed subset of the wider course cohort.	R1–R4 cases plus cohort-scale DMI test-suite gap (Table 3)

Table 2: Evaluation summary. Agreement metrics are treated as sanity checks for an auditable judging workflow; the primary analysis is whether evidence trails expose localisable failures.

Audit signal	N	%
Mechanical verifier checks		
R1 quote-verification failure (post-fix)	0	0
R4 score-bound violation (*mark* > *max*)	0	0
Install failure	0	0
Policy / cohort flags for review		
R2 stub-inheritance cap fired	21	34
R3 claim-vs-artefact flag (notebook + plot)	7	11
Slow-but-correct driver (test recovered)	7	11
GenAI-fingerprint review flag	12	20
DMI broken-by-symmetry (cohort test-suite gap)	11	18
Empty submission	1	2
Operational		
Repo unavailable at marking time	2	3

Table 3: Audit-yield over the rigorously analysed 61-submission subset of the NPP cohort. Mechanical verifier checks (R1, R4) catch and reject failures the orchestrator can resolve automatically. Policy flags surface rows the lecturer inspects manually. R2 (stub-inheritance) is the most frequent firing; the DMI cohort-scale pattern — implementations that pass the instructor's unit tests by symmetry coincidence while differing from the rubric's stated formula — is visible only because the evidence trail compares each implementation against the rubric rather than against test pass/fail alone.

matters because students sometimes leave aspirational comments (e.g. "`# uses np.einsum for efficiency`") on simpler code, and the agent must not award the bonus on the basis of the comment alone. A final score alone would not have shown that the agent's evidence was grounded in the implementation rather than the surrounding prose.

Attribution to starter material (R2). In the programming coursework, some apparently valid code, documentation, or packaging content was inherited from the instructor-provided stub. Crediting it would overstate student work. Stub-inheritance detection therefore became an explicit assessment policy: evidence that is identical or near-identical to instructor starter material does not count as student-authored evidence for judgement-heavy criteria.

Claim-vs-artefact verification (R3). R3 handles cases where the artefact's apparent behaviour does not match what it claims. Two NPP examples illustrate. A research notebook displayed what looks like a skyrmion simulation result, but the driver cell was never executed — the visualisation is the hand-constructed initial spin state, not simulation output. Separately, a `Spins.plot` method hardcoded a synthetic vector field (`u=x/r; v=y/r; w=-np.exp(-r**2)`) rather than reading `self.array`. In both cases the agent's evidence trail decoupled the cited visualisation from the code path that would produce it, and the orchestrator flagged the row for human review. The policy is that any judgement-heavy credit must be supported by the code path that produces the cited evidence.

Test-suite gap detection. Evidence trails also surfaced a cohort-scale test-suite gap. 18% of marked NPP submissions implemented the DMI energy with a single cross-product component for both link directions — mathematically wrong, but coincidentally producing the correct magnitude on the instructor's unit test (which asserts `abs(...)` and uses a spin pattern that yields identical absolute values for the correct and broken formulations). The agent's evidence trail compared each implementation against the rubric's stated DMI formula

Component removed	Failure no longer localised	Evidence from NPP	Likely outcome
No source / quote verifier	Unsupported implementation evidence could be credited from comments, docstrings, or surrounding prose rather than executable code.	0/61 post-fix failures (preventative check)	Aspirational comments could earn quality bonuses the implementation does not support
No stub-inheritance policy	Instructor-provided starter material could be treated as student-authored work.	21/61 R2 flags	21/61 students would receive credit for instructor-provided baseline material on documentation or packaging
No claim-vs-artefact verification	Notebook or plotting claims could be accepted without an executable code path supporting the claimed result.	7/61 R3 flags	7/61 plausible-looking artefact claims would propagate to final marks unchallenged
Test-only marking	Implementations that pass unit tests by symmetry coincidence would be accepted despite violating the rubric formula.	11/61 DMI cases	11/61 students would receive full DMI credit despite implementing the wrong formula
No timeout-recovery on deterministic tests	Slow-but-correct implementations fail the per-file pytest budget and are mechanically zeroed despite being correct.	7/61 driver-test recoveries via 600 s re-run	Up to 7 marks per affected student lost to test-budget alone, with no audit trail to surface the cause

Table 4: Counterfactual audit yield over the rigorously analysed 61-submission subset of the NPP cohort. Rather than treating this as a benchmark ablation, we ask which deployment-observed failures would no longer be localised if a verifier, policy, or evidence-trail component were removed.

rather than against test pass/fail alone, and flagged every case. This illustrates that evidence trails audit not only the model's interpretation, but also the gap between what a test asserts and what the rubric describes.

6.3 Public benchmarks as portability tests

The public benchmarks test whether the evidence-trail structure transfers beyond the NPP coursework. On ASAP-SAS, Table 5 compares three zero-shot conditions on the same five prompts: direct scoring without a rubric (C1, mean QWK 0.674), rubric-conditioned single-pass scoring (C2, 0.662), and the full two-stage pipeline (C3, 0.705). C3 outperforms C1 on four of five prompts, with the largest gains on prompts whose rubrics require locating or gating specific elements (P4: +0.096; P10: +0.056). C2 outperforms C3 on P3, the only prompt with a holistic depth-of-explanation rubric where Stage 1's decomposition into sub-criteria introduces systematic over-scoring at the Score 1/Score 2 boundary.

The prompt-type interaction is informative. For evidence-grounded prompts (P4, P10), Stage 1's extraction of relevant elements and Stage 2's application of gating logic provides structured context

Prompt	n	C1 Direct	C2 Rub+Ev	C3 Full	IRR
P1 (sci, 0–3)	201	.690	.696	**.718**	.946
P2 (read, 0–3)	202	.836	.854	**.855**	.936
P3 (read, 0–2)	202	.725	**.746**	.683	.821
P4 (lang, 0–2)	245	.455	.353	**.551**	.829
P10 (data, 0–2)	202	.663	.647	**.719**	.887
Mean		.674	.662	.705	.884

Table 5: ASAP-SAS QWK under three zero-shot conditions (same model, no labelled examples). C1: direct scoring without rubric. C2: rubric-conditioned single-pass. C3: full two-stage pipeline. Bold = best per prompt. C3 outperforms C1 on 4/5 prompts.

that a single pass does not retain; C2 regresses on P4 (0.353) precisely because its conditional rubric logic is difficult to apply without explicit element extraction. For holistic prompts, the pattern is mixed: C3 matches or exceeds C2 on P1 and P2, where element identification is helpful, but underperforms on P3, where the rubric rewards combined depth of explanation rather than the presence of identifiable components.

The three conditions also differ in what they retain. C1 and C2 produce a score and a paragraph of free-text justification. C3 produces a structured

evidence trail with per-subcriterion marks, maxima, quoted spans, and Boolean flags, supporting mechanical verification and post-hoc correction. This auditability is orthogonal to aggregate accuracy: even where C3's QWK matches C2, the structured output enables diagnosis and repair that free-text justification does not.

SAS-Bench (Lai et al., 2025) provides a complementary test because it contains structured step-level grading with varied score ranges. Exact-match accuracy varies widely across buckets because hitting the exact integer score is inherently harder for 25–34 point tasks than for 5–6 point tasks. QWK and normalised error are therefore more informative. The held-out results show strong transfer on 10 of 12 buckets, while the two weaker buckets remain diagnosable: Geography short-answer shows a train-to-validation drop after refinement, and English gap-filling shows systematic over-marking that likely needs a stricter answer-matching policy.

7 Discussion

The framework reframes LLM grading from "can the model assign a mark?" to "can the lecturer inspect and control the marking logic?" This is important for an EMNLP audience because the technical object is not an educational product alone; it is an intermediate representation for LLM judgement. The response-specific sub-rubric externalises rubric interpretation. The evidence trail externalises score justification. The verifier checks externalise parts of the trust boundary that should not be left to generation.

This also clarifies the role of metrics. We report agreement against human scores as a sanity check, not as the primary contribution: two human markers on the same rubric will themselves disagree, and a system that matches one human exactly may have arrived at the score for a different reason. The central empirical claim is that structured traces make wrong-reason failures visible — the same final score can arise from very different chains of judgement, and only the intermediate fields surface which. The NPP and ASAP-SAS fixes show this directly: scoring-rule defects, fabricated evidence, and policy mismatches could be repaired because the system retained the intermediate fields needed to locate the error.

The lecturer-as-builder workflow is the pedagogical contribution. Modern LLMs bring latent domain knowledge and consistency, but they do not know the local assessment culture unless it is encoded. The framework gives the lecturer places to encode it: high-level rubrics, calibration examples, prompt guidance, deterministic tests, source policies, and verifier checks. The evidence trail then gives the lecturer a way to verify that these choices have actually shaped the judgement.

Finally, evidence trails help distinguish model failures from rubric or test-suite failures. In human marking, a cohort-wide disagreement with the rubric may be dismissed as student misunderstanding. In this deployment, the structured traces made it cheap to ask the alternative question: is the failure on the student's side, or in the rubric or its mechanical checks? The cohort-scale DMI pattern (§6.2) was an instance of the latter. That bidirectional audit is a useful property of intermediate representations in high-stakes judging.

8 Conclusion

We introduced a pedagogy-grounded LLM judging framework that separates rubric interpretation from scoring and records source-grounded evidence trails for each judgement-heavy decision. The framework is designed for lecturer control rather than full automation: instructors encode pedagogical standards as rubric clauses, prompts, calibration examples, deterministic checks, and verifier policies, then inspect whether those standards are followed. Public benchmark results indicate that the structure remains viable across short-answer and step-level grading tasks, while the course deployment shows the main benefit: evidence trails expose local, correctable failures that final scores would hide. Further work includes multi-course deployment and fuller second-marker calibration.

Limitations

The lecturer-as-builder workflow assumes that the rubric author can revise prompts, rubric clauses, or orchestrator checks. We have not studied settings where these components are maintained by a separate technical team, nor settings in which policy changes require formal approval before deployment.

The refinement catalogue is not exhaustive. R1–R6 reflect failures observed in our deployments and calibration sets. Other courses, models, response formats, or institutional policies may surface different failure modes, so production use should include

a defect-discovery phase before full-cohort marking.

The real-course deployment has not yet reported full independent second-marker calibration. This limits interpretation of model–human disagreement on judgement-heavy criteria.

Quote verification only checks source presence. It does not prove that the cited source semantically supports the assigned mark. Semantic support remains a human-auditable but not fully mechanical property in the current framework.

ASAP-SAS and SAS-Bench test portability across public short-answer and step-level grading tasks, but they do not reproduce all complexities of coursework submissions, including missing files, linked repositories, instructor starter material, late commits, and course-specific integrity policies.

Ethical Considerations

This study has formal research ethics approval at the contributing institution. All participants received a Participant Information Sheet and gave informed consent before any of their work was processed for research. Several protections were maintained to separate the AI workflow from grade determination: human marking and feedback release were completed first; AI-generated marks and feedback were then produced using the same rubric, after human marking was final; human markers were not shown any AI output; and no student's module mark was set or revised on the basis of AI output. Coursework and feedback used for research were pseudonymised, with the identifier-to-code mapping held separately and securely. The paper reports only aggregate results and anonymised examples. Inference was performed using institution-approved enterprise-licensed LLMs under terms that prevent student work from being retained for training of public models. Public benchmark data (ASAP-SAS, SAS-Bench) were used only under their respective dataset access terms. The released code and supplementary material exclude private coursework submissions, student repositories, candidate identifiers, and course-specific private infrastructure.

Use of AI assistants. AI assistants were used during the preparation of this paper for language editing, code debugging, and drafting support. The authors checked and revised all generated text, code, and analysis outputs, and remain responsible for the final content of the paper.

References

Anna Bavaresco, Raffaella Bernardi, Leonardo Bertolazzi, Desmond Elliott, Raquel Fernández, Albert Gatt, Esam Ghaleb, Mario Giulianelli, Michael Hanna, Alexander Koller, Andre Martins, Philipp Mondorf, Vera Neplenbroek, Sandro Pezzelle, Barbara Plank, David Schlangen, Alessandro Suglia, Aditya K. Surikuchi, Ece Takmaz, and Alberto Testoni. 2025.

LLMs instead of Human Judges? A Large Scale Empirical Study across 20 NLP Evaluation Tasks.

In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 238–255, Vienna, Austria. Association for Computational Linguistics.

Sue Bloxham, Birgit den Outer, Jane Hudson, and Margaret Price. 2016.

Let's stop the pretence of consistent marking: exploring the multiple limitations of assessment criteria.

Assessment & Evaluation in Higher Education, 41(3):466–481.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Zhouchi Lin, Bowen Zhang, Lionel Ni, Wen Gao, Yuanzhuo Wang, and Jian Guo. 2026.

A survey on LLM-as-a-Judge.

The Innovation, 7(6):101253.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Delong Chen, Wenliang Dai, Ho Shu Chan, Andrea Madotto, and Pascale Fung. 2023.

Survey of hallucination in natural language generation.

ACM Computing Surveys, 55(12):248:1–248:38.

Gerd Kortemeyer. 2023.

Toward AI grading of student problem solutions in introductory physics: A feasibility study.

Physical Review Physics Education Research, 19(2):020163.

Gerd Kortemeyer, Julian Nöhl, and Daria Onishchuk. 2024.

Grading assistance for a handwritten thermodynamics exam using artificial intelligence: An exploratory study.

Physical Review Physics Education Research, 20(2):020144.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023.

G-Eval: NLG evaluation using GPT-4 with better human alignment.

In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages

2511–2522, Singapore. Association for Computational Linguistics.

Peichao Lai, Kexuan Zhang, Yi Lin, Linyihan Zhang, Feiyang Ye, Jinhao Yan, Yanwei Xu, Conghui He, Yilei Wang, Wentao Zhang, and Bin Cui. 2025.

SAS-BENCH: A fine-grained benchmark for evaluating short answer scoring with large language models.

arXiv preprint arXiv:2505.07247.

Brian Riordan, Andrea Horbach, Aoife Cahill, Torsten Zesch, and Chong Min Lee. 2017.

Investigating neural architectures for short answer scoring.

In *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 159–168, Copenhagen, Denmark. Association for Computational Linguistics.

D. Royce Sadler. 2009.

Indeterminacy in the use of preset criteria for assessment and grading.

Assessment & Evaluation in Higher Education, 34(2):159–179.

Yun Wang, Zhaojun Ding, Xuansheng Wu, Siyue Sun, Ninghao Liu, and Xiaoming Zhai. 2026.

AutoSCORE: Enhancing Automated Scoring with Multi-Agent Large Language Models via Structured Component Recognition.

Proceedings of the AAAI Conference on Artificial Intelligence, 40(48):40898–40906.

Changrong Xiao, Wenxing Ma, Qingping Song, Sean Xin Xu, Kunpeng Zhang, Yufang Wang, and Qi Fu. 2025.

Human-AI Collaborative Essay Scoring: A Dual-Process Framework with LLMs.

In *Proceedings of the 15th International Learning Analytics and Knowledge Conference*, pages 293–305, Dublin, Ireland. Association for Computing Machinery.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023.

Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena.

In *Advances in Neural Information Processing Systems*, volume 36, Datasets and Benchmarks Track.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Deng Zhou. 2022.

Chain-of-thought prompting elicits reasoning in large language models.

In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837.

A Evidence-trail schema

Table 7 shows the per-subcriterion fields emitted by the Stage 2 marking agent. The orchestrator consumes this structure to recompute the total mark, run R1 (source-presence) and R4 (score-bound) checks, and surface review flags. A typical NPP submission generates 30–60 such rows across 13 marking tasks; ASAP-SAS responses produce 3–8 rows per prompt.

B Prompting and orchestration protocol

For each artefact bundle, the orchestrator runs the following protocol:

1. **Stage 1: sub-rubric construction.** The model receives the instructor context, high-level rubric, calibration notes, and submitted artefacts. It outputs a response-specific sub-rubric with subcriterion labels, descriptions, and maximum marks. It is explicitly instructed not to assign marks.

2. **Stage 2: evidence-trail marking.** The model receives the same artefact bundle together with the Stage 1 sub-rubric. It assigns marks to each subcriterion and emits the evidence-trail schema in Table 7.

3. **Deterministic verification.** The orchestrator recomputes task totals, enforces score bounds, checks quoted spans or cited source strings against the submitted artefacts, and records review flags.

4. **Aggregation.** Final marks are computed from verified task totals and deterministic test outputs. The verified evidence trail is stored as JSON for later audit and repair.

Domain-triggered policies such as stub-inheritance, claim-vs-artefact mismatch, valid-extra answers, and incomplete-response flags are recorded as review flags rather than silently folded into the score.

C Implementation details and computational budget

Model and inference setup. All LLM-based components were run through API inference. The full two-stage pipeline uses one Stage 1 call to construct a response-specific sub-rubric and one Stage 2 call to apply the sub-rubric and emit the

Subcriterion (anon.)	Max	Mark	Evidence / rationale	Verifier outcome
S_a: drive uses local energy delta (Task 7 quality)	2.0	2.0	Cited code path `def delta_energy(self, i, j, s_new)` computes local ΔE from the perturbed spin and its four nearest neighbours instead of calling `system.energy()` on each iteration. Rubric Task 7 explicit bonus.	quote present in `driver.py`; not stub; R1 pass
S_b: README documents installation (Task 10)	2.5	0.5	*Submitted README is 154 chars, byte-identical to the instructor stub (assessment-brief link + integrity-form link). No installation/usage section added.*	stub-inherited; thin-README cap applied; R2 fired
S_c: notebook simulation produced visible skyrmion (Task 12)	4.0	1.5	*Notebook plot displayed as "final skyrmion" but the driver cell has `execution_count: null`; the displayed quiver field is the hand-constructed initial spin state, not simulation output.*	claim decoupled from code path; R3 flagged for human review
S_d: `System.dmi` matches rubric formula (Task 6 quality)	6.0	0.0	Cited code computes `np.cross(s[1:,...], s[:-1,...])[..., 0]` — the x-component of the cross product — for both row and column neighbour pairs. The rubric's stated DMI formula requires direction-dependent $\mathbf{D}_{ij}$ vectors with different components for x- and y-bonds. Unit tests pass because `test_dmi_non_uniform` asserts `abs(...)` and the chosen spin pattern yields identical absolute values for the correct and broken formulations.	semantic mismatch with rubric; mechanical tests passed; row marked for human review
S_e: agent-emitted subcriterion (Task 10 documentation)	0.5	0.0 (was 1.5)	Agent emitted a subcriterion with `mark = 1.5` against its own declared `max = 0.5`. The orchestrator detected *mark* $>$ *max*, clipped to the declared maximum, and flagged the agent's trail for inspection. Here the failure source is the agent's own arithmetic, not the student's work.	R4 clipped and flagged
S_f: deterministic driver tests pass (Task 7 tests)	7.0	7.0	Initial `pytest mcsim/tests/test_driver.py` run returned 0/3 (per-file 90 s timeout exceeded). Rerun with a 600 s budget passed all three tests in 155 s; the driver implementation is correct but unvectorised. Mechanical test mark restored; the rubric's separate Task 9 (optimisation) subcriterion records the runtime cost.	timeout-recovery path applied; mechanical mark restored; Task 9 docked separately

Table 6: Anonymised excerpt of the structured evidence trail produced during NPP marking. Each row carries a subcriterion label, the maximum mark, the awarded mark, the quoted source span or short rationale, and the verifier outcome. S_a–S_f are six different students drawn from the rigorously analysed subset (Section 6.2). Row 1: positive credit grounded in a quoted code span. Row 2: R2 stub-inheritance cap firing on inherited material. Row 3: R3 claim-vs-artefact policy flagging the row for human review. Row 4: the cohort-scale DMI test-suite gap, where the agent's evidence trail compares the implementation against the rubric's stated formula rather than against test pass/fail alone. Row 5: R4 arithmetic guard firing on the agent's own output, not the student's work — illustrating that the evidence trail audits the model as well as the submission. Row 6: hybrid mechanical+judgement layer recovering a deterministic test that the initial timeout had wrongly zeroed.

evidence-trail schema. Direct-scoring and rubric-conditioned single-pass baselines use one LLM call per response. No model training or fine-tuning was performed.

Unless otherwise stated, Stage 1 and Stage 2 used the same model family and the same decoding configuration. We used deterministic decoding for marking experiments, with temperature set to 0. The prompt templates were fixed before each reported evaluation run. Post-hoc aggregation fixes on ASAP-SAS reused fields already stored in the Stage 2 evidence JSON and did not require additional LLM calls.

Computational budget. The public benchmark experiments used 1,052 ASAP-SAS responses and 427 SAS-Bench held-out examples. The full two-stage public benchmark runs therefore required approximately 2,958 LLM calls, excluding the one-call baseline conditions. The NPP analysis reported in this paper used 61 submissions, a rigorously analysed subset of the wider course cohort. The framework incurred inference cost only; no GPU training budget was used.

NPP marking pipeline. Each submission is processed in three phases. *(i) Deterministic prep*: the student's private GitHub Classroom repository is cloned, installed into a shared Python venv, exer-

Field	Description
`label`	Subcriterion name derived from the Stage 1 sub-rubric
`max`	Maximum mark for this subcriterion
`mark`	Awarded mark ($\leq$ `max`; enforced by R4)
`source_span`	Verbatim quote or code span from the submitted artefact (verified by R1)
`rationale`	Free-text justification linking evidence to the mark decision
`flags`	Zero or more review flags (e.g. `stub-inherited`, `claim-not-in-artefact`, `valid-extra`, `dmi-symmetry-gap`)
`task_total`	Sum of `mark` across subcriteria (recomputed by orchestrator)
`task_max`	Sum of `max` across subcriteria
`feedback`	Student-facing summary generated from the evidence trail

Table 7: Per-task evidence-trail schema. Top rows are emitted per subcriterion; bottom rows are per task. The orchestrator independently recomputes `task_total` from the subcriterion marks as an arithmetic guard.

cised with `pytest` under a 90 s per-file budget, and parsed for commit history, filesystem inventory, and notebook execution metadata; the package is uninstalled before the next student. *(ii) Judgement marking*: an LLM call reads the source files (`mcsim/*.py`, README, `pyproject.toml`, research notebook) plus the prep JSON and emits a per-task evidence trail across the 13 marking tasks. *(iii) Aggregation*: the orchestrator applies the verifier checks, recomputes the row total from sub-criterion sums, and records review flags. The 61-submission subset produced approximately 800 task-level evidence records. The 7 slow-but-correct driver cases (§6.2) were re-run with a 600 s budget; recovered marks were propagated to the row total with a separate audit-log entry. Two further submissions could not be marked (repositories inaccessible at marking time); they are flagged for human review and excluded from the 61.

Implementation and package parameters. The orchestrator, verifier checks, aggregation rules, and metric computation were implemented in Python. Score-bound validation recomputes task totals from the structured evidence-trail fields and rejects marks exceeding declared maxima. Source-presence verification uses exact string matching for code and repository artefacts. For ASAP-SAS short-answer text, quote verification uses a lightweight bigram-overlap check to tolerate minor paraphrase-style span variation. Quadratic weighted kappa, accuracy, and mean absolute error were computed with standard Python evaluation utilities. The verifier layer is deterministic and does not use an additional trained classifier or learned retrieval model.

D Example evidence-trail excerpt

Table 6 shows six anonymised subcriterion rows drawn from the per-student evidence JSON produced by the marking agent on six different NPP submissions. Each row links a subcriterion decision to its quoted source evidence and the verifier outcome. The full trail for each student typically contains 30–60 such rows across the 13 marking tasks. The six rows are chosen to illustrate the distinct verifier and policy mechanisms the framework relies on: positive credit grounded in code (S_a), the two most frequent NPP policy firings R2 and R3 (S_b, S_c), the cohort-scale test-suite gap surfaced by semantic comparison against the rubric formula (S_d), the R4 arithmetic guard catching a malformed agent output (S_e), and the hybrid mechanical+judgement layer's timeout-recovery path (S_f).

E NPP rubric summary

The NPP coursework rubric covers 13 marking tasks summing to 100 marks: regular commits (5); test+quality pairs for seven implemented methods, `mean` (5), `__abs__` (6), `zeeman` (6), `anisotropy` (7), `exchange` (8), `dmi` (12), and `drive` (14); plot quality (10); cross-cutting optimisation (10); documentation (7); packaging (6); and a research notebook executed without changes (4).

F ASAP-SAS per-prompt details

Table 8 reports the full two-stage pipeline (C3) alongside human inter-rater reliability for each prompt.

Prompt	Range	n	C3	IRR	%IRR
P1	0–3	201	.718	.946	76
P2	0–3	202	.855	.936	91
P3	0–2	202	.683	.821	83
P4	0–2	245	.551	.829	66
P10	0–2	202	.719	.887	81
Mean			.705	.884	80

Table 8: Full framework (C3) QWK vs human Score1 and human inter-rater reliability (IRR). %IRR = C3 QWK as a percentage of human IRR. C3 reaches 66–91% of human agreement across five prompts.

G ASAP-SAS post-hoc corrections

During iterative development, two scoring-rule errors were diagnosed from the evidence trail and corrected at the aggregation layer without additional API calls. Table 9 reports the before/after QWK from the pipeline iteration in which each error was identified. Both errors arose from deterministic formulas applied to fields that Stage 2 had already correctly extracted; the corrections were carried forward into the final pipeline whose results appear in Table 5.

Prompt	Root cause	Before	After	Δ
P2	Raw sum over-counts ranking claims (B>D>C>A = 4 marks, should be 1 conclusion)	.739	.777	+.038
P4	Boolean gate requires definition AND article connection; collapses partial credit to 0	.492	.551	+.059

Table 9: Post-hoc scoring-rule corrections identified during development. Both fixes re-read fields already stored in the Stage 2 evidence JSON (`n_valid_conclusions`, `n_valid_improvements` for P2; sub-criterion totals for P4). No additional LLM calls were made. These corrections are incorporated in the final C3 results reported in Table 5.

H ASAP-SAS R1 quote-verification ablation

Table 10 shows the effect of removing the R1 bigram-overlap quote check on ASAP-SAS. The model rarely fabricates quotations on short-answer text: R1 fires on only 20 of 1052 responses. On P4, R1 penalises 14 paraphrase-style quotes that are semantically valid but fail exact overlap, slightly reducing QWK. On P10, R1 correctly zeroes 5 unverifiable sub-criteria, improving QWK by 0.026. This contrasts with NPP, where R1 acts preventatively on a larger scale by ensuring evidence is grounded in executable code rather than comments or docstrings.

Prompt	n	With R1	W/o R1	Δ	Affected
P1	201	.718	.718	.000	0
P2	202	.855	.855	.000	0
P3	202	.683	.688	+.005	1
P4	245	.551	.600	+.049	14
P10	202	.719	.694	−.026	5

Table 10: R1 quote-verification ablation on ASAP-SAS. Δ = QWK change when R1 is removed. The check has minimal aggregate impact on short-answer text but serves different functions across artefact types (Section 6.2).

I SAS-Bench per-bucket results

Bucket	Prompt	n	QWK	Acc.	MAE	R1 fail
Chinese gapfill	generic	26	0.975	0.96	0.04	8.3%
Physics choice	answer_match	22	0.953	0.91	0.18	0.0%
Biology gapfill	generic	48	0.948	0.50	0.88	0.4%
Math gapfill	answer_match	49	0.905	0.94	0.22	2.0%
Political short	generic	44	0.892	0.14	2.09	0.5%
Chinese short	generic	48	0.825	0.25	2.79	0.5%
History short	history	48	0.819	0.31	2.42	3.3%
Chemistry gapfill	generic	9	0.741	0.00	3.00	23.0%
Math short	generic	36	0.724	0.31	1.97	9.0%
Physics short	generic	22	0.723	0.27	1.59	7.0%
Geography short	geo-specific	27	0.614	0.22	2.19	0.0%
English gapfill	generic	48	0.603	0.29	2.17	0.0%

Table 11: Held-out SAS-Bench validation results by bucket. Prompts were frozen before evaluation.