Benchmarking LLM Judges for Soundness Flaw Detection in Open-ended Mathematical Modeling Reports

Anonymous ACL submission

Abstract

Open-ended mathematical modeling tasks, such as predicting the housing supply in a city for the next five years, require extensive information gathering and quantitative reasoning, and there could exist multiple valid approaches to address the same task. While Deep Research Agents (DRAs) demonstrate remarkable potential in generating reports for such tasks, our analysis found that they still produce non-negligible soundness flaws, i.e., errors occurring in the derivation of data, assumptions, and methodology to conclusions. Reliably detecting such flaws is a prerequisite for trusting and improving DRA outputs. In this work, we introduce OSJBENCH, a dataset for evaluating LLM judges, which consists of math modeling reports paired with annotations of soundness flaws. To improve LLM judges on this task, we systematically study a two-axis design space of *rubric guidance* and *input representation*. Experimental results show that instance-level rubrics yield the most pronounced gains in flaw detection, indicating that the principal bottleneck is knowing *what to check* for each problem; the graph representation provides a consistent additional improvement that is, however, partially substitutable with rubric specificity. Furthermore, supervised fine-tuning further improves precision, offering a viable path to enhance the capability of LLM-as-a-judge for open-ended MM reports.

1 Introduction

Mathematical modeling (MM) is to abstract an ambiguous real-world question into a tractable mathematical structure (Bliss et al., 2014). It is a foundational tool in multiple domains, including financial valuation (Merton, 1994), supply-chain planning (Lee et al., 2016), and public-policy decisions (Panovska-Griffiths et al., 2021). Such tasks have no unique correct answer: starting from different assumptions, one can construct entirely different yet equally defensible mathematical models. For instance, predicting a city's housing supply over the next five years can be reasonably approached by a logistic growth model anchored to a saturation capacity, or by a time-series model fitted to a monthly inventory series.

Deep Research Agents (DRAs) (OpenAI, 2025a) are now able to produce complete MM reports. Since these reports admit no single correct answer, evaluating them is naturally delegated to LLM-as-a-judge (Li et al., 2025; Du et al., 2025). However, most evaluate against a single golden answer (Mialon et al., 2023; Wei et al., 2025; Phan et al., 2025) or reference-based textual overlap (Li et al., 2025), which is not applicable when multiple modeling paths are equally valid. The few benchmarks designed for open-ended reports rely on LLM judges that return only an overall quality score (Sharma et al., 2025; Li et al., 2026), which is not all that downstream users need: an analyst deciding whether to trust a MM report needs to know *which* statements are not convincing, not just how high the score is. We therefore reframe the judge's task from scoring to *soundness error detection*, which is to locate spans of the report where the derivation from data and assumptions to conclusions breaks down, e.g., magic numbers without justification. Our paper aims to evaluate the ability of LLM judges in detecting soundness errors in open-ended MM reports.

To this end, we construct OSJBENCH (Open-ended Soundness Judge Benchmark), a benchmark dataset consisting of 2k reports with soundness-error labels for reference-free evaluation. The construction proceeds in two stages. In the first stage, we generate reports on Math Modeling Challenge (M3) problems (M3, 2026) using a diverse pool of DRAs and annotate each report. However, the number of reports collected in the first stage alone is insufficient as an evaluation benchmark because of the cost of expert annotation. The second stage scales up the benchmark by synthesizing reports

with controllable error placement. Using the taxonomy and error-type frequencies induced from the first stage as a prior, we inject soundness errors into clean or annotated reports, yielding reports whose (error type, sentence span) ground truth is fully known by construction.

We further explore how to improve LLM judges on soundness-error detection. A capable judge needs two abilities at once: *knowing what* to inspect, and *tracing how* data, methods, and conclusions depend on one another in the report. We accordingly characterize the judge's design space along two axes: (i) rubric guidance, which controls what the judge should check. We propose the *instance-level rubric*, in which an LLM first proposes the modeling paths a sound solution might take and derives a checklist of step-level requirements the report must satisfy; and (ii) input representation, which controls how the report is presented to the judge. We propose a *graph representation* that converts the report into a directed logical graph whose nodes are atomic claims (data, parameters, derivations, conclusions) and whose edges are their dependencies.

We explore the performance of LLM judges across these configurations. Experimental results show that rubric guidance has the most pronounced effect: the instance-level rubric yields significant F1 gains (over 10) across all judges, indicating that the principal bottleneck is identifying what should be checked for a given problem. The graph representation provides a positive contribution, but this advantage is offset by the instance-level rubric. Finally, building on this, supervised fine-tuning further improves precision without sacrificing recall. Together, our best proprietary and open-weight judges reach F1 scores of 94 and 82, respectively, making them practical tools both for assisting human analysts and for serving as reward signals to train future DRAs. We will release our benchmark along with the judges to the community.

2 Related Work

Benchmarks for DRA Evaluation Existing work on DRA evaluation differs in the form of ground truth it uses. Some benchmarks score agents against verifiable final answers, including GAIA (Mialon et al., 2023), HLE (Phan et al., 2025), and BrowseComp (Wei et al., 2025). For long-form outputs, ReportBench (Li et al., 2025) measures textual and citation overlap with published surveys. DeepResearch Bench (RACE) (Du et al., 2025) and DRB-II (Li et al., 2026) rate full reports on dimensions such as comprehensiveness, insight, instruction-following, and readability, while ResearchRubrics (Sharma et al., 2025) and DEER (Han et al., 2025) include a general rubric to include reasoning soundness and logical completeness. None of these benchmarks target open-ended MM reports, where the output is built from assumptions, variables, equations, and parameter calibrations and admits multiple equally defensible modeling paths. In addition, all of these judges output a scalar score or a Likert/binary rating per rubric item, rather than typed errors tied to specific sentences. We depart from this line on both points: we evaluate DRAs on open-ended MM reports, and we frame LLM-as-a-judge as a typed, sentence-localized soundness-error detector.

LLM-as-a-judge for Reports A separate line of work studies how to make LLM judges more reliable for long reports. On the rubric side, ResearchRubrics (Sharma et al., 2025) and DRB-II (Li et al., 2026) obtain instance rubrics with heavy expert curation. RACE (Du et al., 2025) designs the auto-generation task-specific criteria for comprehensiveness, insight, and readability. However, these instance-level rubrics aim at content coverage, instruction-following, or general writing quality, not at logical soundness in analysis. Our instance-level rubric exploration differs on exactly this point: given only the question, an LLM enumerates plausible modeling paths and, along each derivation chain, lists what every step must make explicit, such as the calibration source of each parameter, the year of an initial condition, and the justification for any proxy variable. On the input side, judges are usually prompted on the raw report or on response-level atomic facts (e.g., FActScore (Min et al., 2023)), where the unit being checked is a factual claim rather than a reasoning dependency. Our work explores how to extract a report into a logical graph of nodes and use it as the judge's input.

3 OSJBENCH

We start by characterizing how DRAs perform in generating reports for MM problems. In particular, we establish a taxonomy of soundness flaws that are commonly made by current DRAs by comparing their reports with the human references. The taxonomy then serves as the foundation for the construction of OSJBENCH.

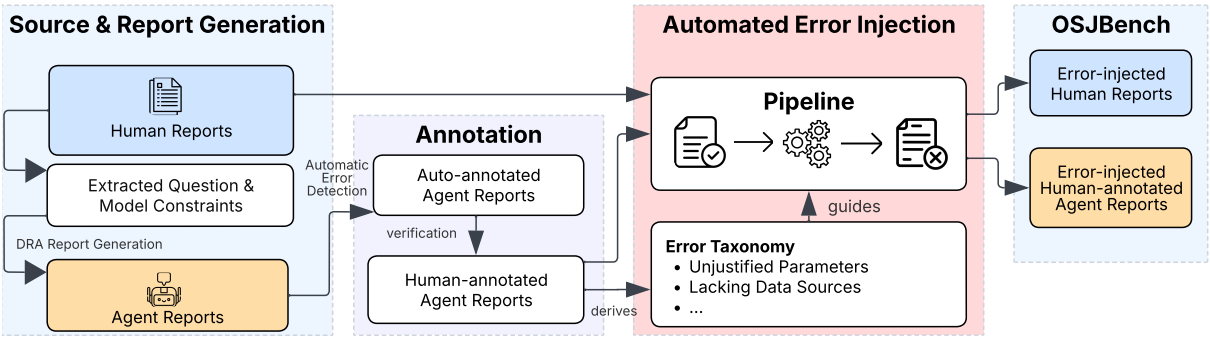

Figure 1: The soundness error identification and injection pipeline for OSJBENCH. A taxonomy of logical errors is derived from human annotations and subsequently guides the automated error injection process. The process corrupts reports to produce a high-quality set from human and DRAs, and is verified by annotators.

Source Data (M3 Challenge)	
# of math modeling questions	43
# of human award-winning reports	176
# of (question, constraint) prompts	176
# of DRA-generated reports	140
OSJBENCH	
# of reports	2,000
# built on human reports (no native errors)	1,000
# built on DRA reports (with native errors)	1,000
Avg. errors per report	4.2

Table 1: Statistics of the OSJBENCH.

3.1 Data Source

High-quality math modeling reports with reliable human evaluation are scarce, as producing and grading them requires substantial domain expertise. We build on the Math Modeling Challenge (M3),[1] a high school MM competition whose past problems span diverse domains from environmental science to economics (Appendix B) and come with award-winning student reports that serve as high-quality references. After filtering out writing-heavy tasks (e.g., policy recommendations), we retain 43 questions. Because open-ended modeling problems admit multiple valid approaches, random model choices of DRAs would confound the annotation. Therefore, we extract the core modeling approach from each human report and treat it as a generation constraint (e.g., *"Use logistic regression to predict housing supply"*), and additionally require the agent to follow a standard math-modeling report format (Bliss et al., 2014). This yields 176 unique (question, constraint) generation prompts, each pairing the DRA-generated report with its corresponding human reference for evaluation.

We generate reports using a diverse pool of DRAs: two direct API settings (o4-mini-deep-research (OpenAI, 2025), Tongyi-DeepResearch-30B (Alibaba Tongyi Lab, 2025)), and twelve open-framework settings combining three agent harnesses (DeerFlow (ByteDance, 2025), ms-agent (ModelScope, 2025), LangChain Open Deep Research (LangChain, 2025)) with four backend LLMs (GPT-5.4-nano, Qwen3.5-35B-A3B, Gemma4-26B-A4B-IT, DeepSeek-V4-Flash). Open-framework settings use the Tavily search engine. Sampling 10 reports per setup yields 140 DRA reports in total; full statistics are in Table 1.

3.2 Flaw Identification and Error Taxonomy

Systematically identifying soundness errors in DRA reports through human audit is challenging: long-form reports are costly to audit from scratch. We therefore adopt an LLM-assisted candidate-discovery procedure. We first use GPT-5.5 to compare each DRA report with the corresponding award-winning student report and propose candidate soundness errors under a deliberately relaxed, recall-oriented criterion, and then run a cross-model verification pass with Claude Opus-4.7 (Anthropic, 2026) to validate and supplement these candidates. Human annotators subsequently verify each candidate, revise its error type when necessary, and add errors missed by the models (Appendix C).

Building on these human-verified annotations, we manually cluster the identified errors into a taxonomy grounded in real DRA failure modes rather than hypothesized ones (Table 2). The most frequent error type is *Unjustified Quantitative Result*, where reports provide insufficient explanation for numerical outputs; other common errors include introducing external data without proper citation and presenting parameters without explanation. This taxonomy, together with its empirical frequency

[1] https://m3challenge.siam.org/resources/

Category	Definition	Example	Freq.
Unjustified Quantitative Result	Presenting numerical results without enough intermediate calculations or reproducible derivation.	The report lists city-level forecasts such as "Seattle: 28.5%" but does not show the calculation that produces the number.	16.94%
Calculation Error	A numerical result is inconsistent with the report's own stated formula, inputs, or constraints.	The report defines $Y = \sum_i P_i R_i$, but the value reported in the table does not equal the weighted sum of the listed P_i and R_i.	13.97%
Unjustified Parameters/Hyper-Parameters	Failing to justify how parameters, thresholds, weights, or scenario constants are selected.	The report sets the manual-labor remote-readiness rate to 0.05 and the corporate rate to 0.35 without citation, calibration, or derivation.	13.32%
Insufficient Model Justification	Choosing a model, objective, constraint, or structural assumption without explaining why it fits the problem.	The report uses a linear trend model for long-term adoption but gives no reason why linear growth is appropriate instead of logistic or saturation behavior.	11.45%
Variable Definitions	Variables, indices, units, or latent quantities are undefined, ambiguous, or inconsistent across the report.	The report introduces a vulnerability score S_i but does not clearly define whether larger values mean higher risk, lower resilience, or both.	11.00%
Sensitivity and Validation	Making validation, robustness, uncertainty, or diagnostic claims without actually supporting them.	The report states that changing key assumptions alters results by less than 1%, but provides no sensitivity table, parameter range, or recomputation.	9.66%
Lacking Data Sources	Using data, empirical estimates, or real-world inputs without traceable sources or extraction details.	The report says it uses historical city employment data from 2019–2023 but does not provide the dataset, table, URL, or extraction procedure.	8.02%
Incorrect Handling of Data Limitations	Ignoring or mishandling limitations such as missing data, small samples, extrapolation, outliers, or synthetic data.	The report extrapolates from only four years of city data to a long-term forecast without discussing sample size, uncertainty, or structural breaks.	6.14%
Unit Inconsistency	Mixing incompatible units, time scales, or conversions in equations or interpretations.	The report treats a monthly growth rate as if it were an annual rate when projecting multi-year outcomes.	4.51%
Internal Inconsistency	Different parts of the report contradict each other in assumptions, equations, tables, or conclusions.	The report first assumes "All necessary electric-semi infrastructure is already in place," but later states "Remote areas with limited grid infrastructure may not support widespread electric truck adoption."	3.57%

Table 2: Soundness error taxonomy induced from DRA-generated mathematical modeling reports. Frequency is computed over 2019 retained soundness errors from 140 annotated DRA reports.

distribution, guides the construction of controlled error reports in Section 3.3.

3.3 OSJBENCH Construction

OSJBENCH has 2k reports with span-level soundness-error labels to evaluate LLM judges in a *reference-free* setting, where the judge sees only the report under test, reflecting real-world judging scenarios. Dataset statistics are shown in Table 1; the construction pipeline is illustrated in Figure 1.

Because post-hoc human annotation does not scale (only 316 verified reports), we construct the remaining reports via taxonomy-guided error injection. The pipeline proceeds in three steps: (i) sample target error types from the empirical distribution in Table 2 (e.g., *Unjustified Quantitative Result* is sampled with higher probability); (ii) an LLM acts as a *locator* to identify sentence spans semantically aligned with each target error (e.g., a parameter citation to remove); and (iii) the LLM then acts as a *rewriter* to introduce the logical degradation while preserving document structure and fluency (examples in Appendix D). Importantly, the synthetic portion is not constructed from arbitrary corruptions. We use the taxonomy and empirical frequency distribution induced from human-verified native DRA errors as an injection prior. This prior controls both which error types are injected and how often they appear, making the injected subset reflect the observed failure profile of current DRAs at the error-type level. Human validation then checks instance-level validity, namely whether each injected flaw is logically valid and naturally integrated into the surrounding report. Human validation on a sampled subset of pipeline outputs yields an injection success rate of 0.97 (Appendix C).

Each report's ground-truth labels are the union of (i) any pre-existing soundness errors annotated in Section 3 (only for DRA reports; award-winning reports are treated as error-free) and (ii) the errors injected in this stage. We evaluate each judge at the report level using Precision, Recall, and F1. A detected error counts as correct only if both its error type and sentence span match an annotation.

4 Soundness Error Detection

Detecting soundness errors in open-ended MM reports requires the judge to do two things: decide *what to check* for a given problem, and trace *long-range reasoning dependencies*. We accordingly characterize the judge's design space along two independent axes: (i) **rubric design**, controlling *what* the judge should check, and (ii) **input representation**, controlling *how* the report is presented.

4.1 Rubric Design

We consider two levels of rubric guidance, ordered by the amount of task-specific information injected into the judge prompt.

General Rubric. The judge is shown the full error taxonomy with category names, judgment criteria, definitions, and examples for each type. This rubric is *task-agnostic*: every report is judged against the same standards, encoding general requirements for logical rigor rather than domain knowledge about the specific question. This reflects a realistic deployment scenario in which per-question rubrics cannot be prepared in advance.

Instance-level Rubric. The judge is given a question-specific rubric that instantiates the general error taxonomy on the question at hand. Because open-ended questions admit multiple valid modeling paths, we obtain the instance-level rubric in two steps: from the question alone, an LLM judge first enumerates the common modeling approaches and predicts how each error type in the taxonomy is most likely to manifest under that approach. For instance, for a logistic housing-growth model, the rubric anticipates that *Unjustified Parameters* will likely concern the carrying capacity K and growth rate r (which calibration data and estimation procedure produced them), and that *Missing Data Sources* will likely surface around the initial condition (which anchor year it is taken from). The rubric is derived from the question alone and never observes the report, providing task-relevant standards without information leakage.

4.2 Input Representation

The most direct representation is the natural-language (NL) report, which preserves all detail but forces the judge to track reasoning dependencies across long, redundant text that LLM judges may lose focus (Shi et al., 2025). To mitigate this, we additionally extract the report into a directed *logical graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that exposes its reasoning structure on a single audit surface. Nodes are of three types: *Claim* (c_i): any assertion made in the report (model choices, parameter values, derived quantities, final conclusions); *Evidence* (e_j): external grounding such as data points or citations; *Assumption* (a_k): explicit conditions or simplifications. Edges are also of three types—*Generate* ($e/c_{\text{in}} \rightarrow c_{\text{out}}$): a calculation producing an output from its inputs; *Support* ($e \rightarrow c$): direct grounding of a claim by evidence; and *Condition* ($a/c_{\text{in}} \rightarrow c_{\text{out}}$): a claim holding under an assumption or prior claim.

We construct $\mathcal{G}$ by prompting an LLM to parse the report into nodes and edges following the schema, as shown in Figure 2. The judge then inspects $\mathcal{G}$ in two passes: first checking intrinsic node properties by type (e.g., whether *Claim* or *Assumption* nodes violate physical common sense, or whether quantitative nodes are missing units), then validating inter-node connections (e.g., a parameter node must have at least one incoming *Evidence* or *Claim* edge to be considered grounded). The inspection results feed the final judging stage that outputs error types and sentence-level localizations. We manually verify that the extractor preserves the reasoning gaps actually present in the report and does not hallucinate compensating links that would mask the original flaws (Appendix F).

5 Experiment

5.1 Experiment Setup

Judge Models and Scoring Protocol We benchmark three LLM judges in detecting soundness errors from open-ended MM reports: GPT-5.5 (OpenAI, 2026), GPT-5.4-nano (OpenAI, 2025b), and Qwen3.5-35B-A3B (Yang et al., 2025). To keep each judge's pipeline self-contained, the instance-level rubric and the logical graph are produced by the same LLM that serves as the judge. The final soundness error prediction is formatted as a set of error types and error sentences, and we utilize GPT-5.4-mini to compare the predicted and ground-truth errors to tolerate slight lexical variations (details

Judge	Representation	General Rubric			Instance-level Rubric		
		P	R	F1	P	R	F1
GPT-5.4-nano	NL	60.1	63.3	61.6	74.2	**77.5**	75.8
	Graph	**70.1**	**64.7**	**67.3**	**75.7**	76.3	**76.0**
Qwen3.5-35B-A3B	NL	71.2	65.4	68.2	82.9	**75.9**	79.2
	Graph	**76.6**	**74.8**	**75.7**	**84.3**	75.5	**79.7**
GPT-5.5	NL	85.3	87.1	86.2	**93.2**	**95.1**	**94.1**
	Graph	**89.5**	**92.2**	**90.8**	92.4	94.3	93.3

Table 3: Soundness error detection performance (Precision, Recall, F1) under the rubric guidance $\times$ input representation design space. Bold marks the better representation within each (judge, rubric) cell.

Category	GN	GG	IN	IG
Unjustified Quant. Result	61.8	69.4 $\uparrow$7.6	77.6 $\uparrow$15.8	78.5 $\uparrow$16.7
Calculation Error	81.3	82.1 $\uparrow$0.8	87.5 $\uparrow$6.2	86.2 $\uparrow$4.9
Unjustified Parameters	58.2	62.4 $\uparrow$4.2	68.9 $\uparrow$10.7	69.6 $\uparrow$11.4
Insufficient Model Just.	55.7	65.3 $\uparrow$9.6	73.8 $\uparrow$18.1	75.6 $\uparrow$19.9
Variable Definitions	56.4	64.7 $\uparrow$8.3	71.8 $\uparrow$15.4	73.6 $\uparrow$17.2
Sensitivity & Validation	47.9	55.5 $\uparrow$7.6	58.1 $\uparrow$10.2	58.7 $\uparrow$10.8
Lacking Data Sources	57.6	66.8 $\uparrow$9.2	74.7 $\uparrow$17.1	76.4 $\uparrow$18.8
Incorrect Data Limitations	61.4	67.6 $\uparrow$6.2	75.5 $\uparrow$14.1	76.8 $\uparrow$15.4
Unit Inconsistency	57.8	59.8 $\uparrow$2.0	93.0 $\uparrow$35.2	92.4 $\uparrow$34.6
Internal Inconsistency	70.3	75.8 $\uparrow$5.5	78.4 $\uparrow$8.1	79.6 $\uparrow$9.3

Table 4: Per-category F1 of GPT-5.4-nano under general and instance-level rubrics. Arrows show absolute F1 improvement over General NL within the same category.

are in Appendix C). All prompts used in our experiments are in Appendix G.

5.2 Main Results

Table 3 reports judge performance across four settings: General Rubric with Natural Language (GN), General Rubric with Graph (GG), Instance-level Rubric with Natural Language (IN), and Instance-level Rubric with Graph (IG). We report three findings.

Finding 1: Under the General Rubric, Graph compensates for NL by exposing dependency structure. In the General Rubric columns of Table 3, Graph yields stable gains over NL: GPT-5.4-nano improves from 61.6 to 67.3, Qwen from 68.2 to 75.7, and GPT-5.5 from 86.2 to 90.8. Table 4 shows that these gains concentrate on error types that hinge on structural relations, such as *Insufficient Model Justification* (+9.6), *Lacking Data Sources* (+9.2), *Variable Definitions* (+8.3), *Sensitivity & Validation* (+7.6), and *Unjustified Quantitative Result* (+7.6). The corresponding rows of Table 5 reveal the underlying intuition: such errors appear in NL as sentences scattered across paragraphs, and the judge struggles to reconstruct the dependencies on their own. Graph instead organizes them into an explicit node–edge structure, for instance, treating the unsupported 0.58 weight as a *parameter* $\rightarrow$ *score* link, exposing a missing *unit/normalization edge* between square miles and percentages, or treating the inserted 2024/2025 entries as fitting-input nodes without a source. Graph supplies the structural cues that NL alone does not.

Finding 2: Holding representation fixed, the Instance Rubric resolves a grounding problem and covers checks that the General Rubric cannot express. Table 3 shows that the Instance Rubric yields consistent gains over the General Rubric across every judge and representation: GPT-5.4-nano improves from 61.6 to 75.8 under NL, Qwen from 68.2 to 79.2, and the strongest judge, GPT-5.5, follows the same trend. Table 4 further shows that these gains are substantial across all categories. This indicates that the principal bottleneck in soundness error detection is not raw model capability, but a *grounding* problem: the abstract error categories in the General Rubric are not grounded in the specific objects of the problem at hand. The Instance Rubric instantiates the taxonomy into question-specific checkpoints, grounding each error category onto concrete targets in the report. In the *Sensitivity and Validation* row of Table 5, for example, the forecast band 128 to 146 is reported without any uncertainty model or interval derivation; it is a well-formed claim node in the graph with no missing incoming edges, so both GN and GG miss it. Only after the Instance Rubric writes out the obligation that "any forecast band must be backed by residual variance, scenario perturbations, or sampling-based intervals" do IN and IG catch the error.

Error Type	Error Description	GN	GG	IN	IG	Instance-level Rubric	Interpretation
Unjustified Parameters	The report assigns a leading scoring weight of 0.58 before comparing alternatives, but gives no source, calibration, or derivation for this value.	✗	✓	✓	✓	For this corridor-ranking task, check whether weights assigned to route length, community motivation, cost, anticipated usage, and infrastructure readiness are justified before ranking the five corridors.	Graph exposes an unsupported parameter-to-score dependency; the rubric names the task-specific scoring weights that must be justified.
Unit Inconsistency	The report combines an area quantity measured in square miles with a percentage quantity as if both were already on the same dimensionless scoring scale.	✗	✓	✓	✓	For the corridor score, verify that geographic quantities, percentage-based community support, cost measures, and usage indicators are normalized to a common dimensionless scale before being added.	Graph exposes incompatible inputs feeding the same score; the rubric states which corridor-ranking quantities require normalization.
Lacking Data Sources	The report adds 2024 and 2025 values to a historical input series before fitting the model, but does not explain how those added values are obtained.	✗	✓	✓	✓	For any fitted forecast in this task, check whether added 2024/2025 entries are observed, imputed, or extrapolated, and whether the construction rule is stated before fitting.	Graph identifies the added years as constructed fitting inputs; the rubric asks how those specific added entries were produced.
Sensitivity and Validation	The report presents a forecast band from 128 to 146 units, but does not provide an uncertainty model, sampling procedure, or interval derivation.	✗	✗	✓	✓	For this forecast/ranking task, check whether forecast bands or stable-ranking claims are backed by residual variance, scenario perturbations, bootstrap/Monte Carlo sampling, or recomputed rankings.	General settings miss the unsupported interval; the rubric makes the task-specific uncertainty evidence requirement explicit.
Unjustified Quantitative Result	The report lists fitted parameter values, such as $\theta = 2.61$ and intercept 14.2, but provides no fitting output or derivation that would reproduce them.	✗	✓	✓	✓	For fitted growth or adoption models in this task, check whether parameters such as θ, intercepts, growth rates, and carrying capacities are tied to the data, objective, and optimization output that produced them.	Graph treats these as fitted quantitative outputs; the rubric names the task-specific fitted values that require reproducible support.

Table 5: Case study illustrating how graph input and instance-level rubrics improve soundness-error detection with GPT-5.4-nano as the judge. GN, GG, IN, and IG denote General-Rubric NL, General-Rubric Graph, Instance-Rubric NL, and Instance-Rubric Graph, respectively. ✗ indicates missing detection and ✓ indicates correct detection.

Finding 3: Under the Instance Rubric, Graph and NL converge because the rubric textualizes Graph's structural cues. Table 3 shows that under the Instance Rubric, Graph's gain over NL shrinks substantially: GPT-5.4-nano (75.8 to 76.0) and Qwen (79.2 to 79.7). Table 4 shows that this compression is uniform across categories, with IN and IG tracking each other closely. The reason is that Graph and the Instance Rubric overlap heavily in the structural cues they provide: the dependencies that Graph exposes through its node–edge structure are already written out in textual form by the Instance Rubric. In the *Unjustified Parameters* row of Table 5, Graph flags the 0.58 weight via its parameter-to-score link, while the rubric directly requires task-specific weights to be justified; in the *Unit Inconsistency* row, Graph detects the issue via unit/normalization dependencies, while the rubric directly requires the relevant inputs to be normalized to a common dimensionless scale. The two paths reach the same verdict, so Graph's independent contribution is absorbed by the rubric, and the NL–Graph gap narrows as the rubric becomes stronger.

5.3 Fine-tuning LLM Judges

To further enhance the performance of the LLM judge, we also explore supervised fine-tuning (SFT). Specifically, we curated a training dataset using 231 reports prior to 2020 and the remaining 85 reports as the test set. By re-applying our error injection pipeline, we generate 3k new "report-error" pairs. The input is the report and instance-level rubric, and we leveraged GPT-5.5 to generate high-quality soundness error reasons based on the golden soundness errors as the reference. More details are in Appendix E.

In our experiment, SFT further improves overall judge performance, primarily by raising precision. Compared to the untrained instance-level rubric prompting baseline, SFT increases precision from 82.9 to 92.4 while leaving recall largely unchanged, yielding an overall F1 improvement from 79.2 to 82.3. This indicates that the fine-tuned model be-

comes more confident in its identification of genuine soundness errors and produces fewer false positives, i.e., it is less likely to flag content that does not constitute a clear error. In other words, SFT helps the model learn which inspection findings should actually be reported as errors, thereby improving the precision of error detection.

5.4 Out-of-taxonomy Generalization

A practical concern for deploying LLM judges is whether they can still detect soundness errors of types that fall outside the taxonomy. To evaluate this, we sample 500 reports from OSJBENCH and re-evaluate GPT-5.5 as a judge under the Instance Rubric-NL (IN) setting. For each report, we randomly select one target error type from its ground-truth annotations and remove that type from the rubric, so that the masked type varies across reports rather than concentrating on a few fixed categories. We additionally include an explicit note in the prompt stating that the listed error categories may not cover all errors in the report, signaling to the judge that out-of-taxonomy errors can be predicted. Since the masked type name is no longer available in the prompt, we relax the matching criterion for this type: a prediction counts as correct as long as GPT-5.4-mini (our benchmark scorer) judges it to refer to the same erroneous claim as the ground truth and to be semantically related, regardless of how the judge names the error type.

In our experiment, we find that GPT-5.5's detection rate on the masked category drops from 94.7% to 75.9%. This indicates that the taxonomy plays a concrete grounding role: it tells the judge what to inspect, and removing a category substantially reduces coverage of the corresponding error mode. At the same time, the post-masking detection rate remains substantial, showing that the judge retains meaningful out-of-taxonomy generalization. A large fraction of errors is still localized even without an explicit category definition. Together, these results suggest that a complete taxonomy is the primary lever for reliable detection, while out-of-taxonomy capability provides a useful fallback for unlisted failure modes.

5.5 Ranking DRAs using the Best Judge

Our earlier experiments show that GPT-5.5 under the Instance Rubric-NL setting reaches an F1 of 94.1 on soundness error detection, making it a reasonably reliable judge. We therefore turn the setup around and use this judge to evaluate the report quality of different LLMs serving as DRA backbones. Specifically, we plug seven LLMs into the LangChain Open Deep Research harness (LangChain, 2025), including GPT-5.5, Claude Opus 4.7, GPT-5.4-mini, Qwen3.5-35B-A3B, DeepSeek-v4-flash, GPT-5.4-nano, and Mistral-large-2512 (Mistral AI, 2025). Each DRA generates 10 reports based on questions randomly sampled from the 43 initial questions from the M3 challenge.

Model	Avg. Error
GPT-5.5	2.51
Claude Opus 4.7	2.97
GPT-5.4-mini	5.42
Qwen3.5-35B-A3B	5.73
DeepSeek-v4-flash	5.83
GPT-5.4-nano	5.98
Mistral-large-2512	7.73

Table 6: Average number of detected soundness errors per report across reports generated by different LLMs.

Table 6 shows the average errors of different LLMs. GPT-5.5 and Claude Opus 4.7 produce the cleanest reports, with only 2.51 and 2.97 detected errors per report, respectively. Other LLMs exhibit higher error density. Beyond the overall ranking, different model families tend to fail in different ways. The OpenAI models more often leave quantitative parameters, validation claims, or intermediate calculations insufficiently justified, while Qwen and DeepSeek more frequently produce arithmetic or internal- consistency problems in multi-step derivations. This suggests that report soundness is not only a matter of overall model strength: different backend families have distinct failure modes, so a useful judge should identify both the number of errors and the types of errors each model tends to introduce.

6 Conclusion

In this paper, we identified the limitations of current LLMs-as-judges in evaluating open-ended MM reports, particularly in detecting flaws impairing the report's soundness. To systematically assess LLM judges in this task, we constructed the OSJBENCH and also explored two axes of judge improvement. Future work can extend this evaluation paradigm to qualitative domains, such as legal or policy analysis. Additionally, the robust error detection capabilities demonstrated by our method offer a promising avenue for developing reliable reward models to train next-generation Deep Research Agents.

7 Limitation

Our work has two main limitations. First, the questions in OSJBENCH are drawn from the M3 Challenge, which provides high-quality and verifiable open-ended modeling problems but does not directly cover other analytical domains that do not require explicit math modeling, such as qualitative case studies. Second, while the instance-level rubric and the graph method improve the performance, they increase the additional overhead in computing, and it's possible to distill this ability into the NL LLM judge. We view these two aspects as important follow-up works.

References

Alibaba Tongyi Lab. 2025. Tongyi deepresearch. https://github.com/Alibaba-NLP/DeepResearch. Accessed: 2026-05-22.

Anthropic. 2026. Claude Opus 4.7. https://www.anthropic.com/news/claude-opus-4-7. Large language model. Model ID: claude-opus-4-7. Accessed: May 25, 2026.

Karen M Bliss, Kathleen R Kavanagh, and Benjamin J Galluzzo. 2014. *Math modeling: Getting started and getting solutions*. SIAM.

ByteDance. 2025. DeerFlow: Deep research framework. https://github.com/bytedance/deer-flow. Accessed: 2026-05-22.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. 2025. Deepresearch bench: A comprehensive benchmark for deep research agents. *arXiv preprint arXiv:2506.11763*.

Janghoon Han, Heegyu Kim, Changho Lee, Dahm Lee, Min Hyung Park, Hosung Song, Stanley Jungkyu Choi, Moontae Lee, and Honglak Lee. 2025. Deer: A comprehensive and reliable benchmark for deep-research expert reports. *arXiv preprint arXiv:2512.17776*.

LangChain. 2025. open_deep_research: A configurable open source deep research agent. https://github.com/langchain-ai/open_deep_research. Accessed: 2026-01-23.

Young Hae Lee, Paulina Golinska-Dawson, and Jei-Zheng Wu. 2016. Mathematical models for supply chain management. *Mathematical Problems in Engineering*, 2016.

Minghao Li, Ying Zeng, Zhihao Cheng, Cong Ma, and Kai Jia. 2025. Reportbench: Evaluating deep research agents via academic survey tasks. *arXiv preprint arXiv:2508.15804*.

Ruizhe Li, Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. 2026. Deepresearch bench ii: Diagnosing deep research agents via rubrics from expert report. *arXiv preprint arXiv:2601.08536*.

M3. 2026. M3 challenge: Past problems. https://m3challenge.siam.org/resources/past-problems/. Accessed: Jan. 27, 2026.

Robert C Merton. 1994. Influence of mathematical models in finance on practice: past, present and future. *Philosophical Transactions of the Royal Society of London. Series A: Physical and Engineering Sciences*, 347(1684):451–463.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12076–12100, Singapore. Association for Computational Linguistics.

Mistral AI. 2025. Mistral Large 3 675B Instruct 2512. https://huggingface.co/mistralai/Mistral-Large-3-675B-Instruct-2512. Large language model. Model ID: mistralai/Mistral-Large-3-675B-Instruct-2512. Accessed: May 25, 2026.

ModelScope. 2025. ms-agent. https://github.com/modelscope/ms-agent. Accessed: 2026-05-22.

OpenAI. 2025a. Deep research system card.

OpenAI. 2025b. Gpt-5.1 instant and gpt-5.1 thinking system card addendum. Accessed: 2026-01-26.

OpenAI. 2025. o4-mini deep research. https://platform.openai.com/docs/models/o4-mini-deep-research. Accessed: 2026-05-22.

OpenAI. 2026. GPT-5.5. https://chatgpt.com/. Large language model.

J Panovska-Griffiths, CC Kerr, W Waites, and RM Stuart. 2021. Mathematical modeling as a tool for policy decision making: Applications to the covid-19 pandemic. In *Handbook of Statistics*, volume 44, pages 291–326. Elsevier.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, and 1 others. 2025. Humanity's last exam. *arXiv preprint arXiv:2501.14249*.

Manasi Sharma, Chen Bo Calvin Zhang, Chaithanya Bandi, Clinton Wang, Ankit Aich, Huy Nghiem, Tahseen Rabbani, Ye Htet, Brian Jang, Sumana Basu, and 1 others. 2025. Researchrubrics: A benchmark of prompts and rubrics for evaluating deep research agents. *arXiv preprint arXiv:2511.07685*.

Lin Shi, Chiyu Ma, Wenhua Liang, Xingjian Diao, Weicheng Ma, and Soroush Vosoughi. 2025. Judging the judges: A systematic study of position bias in llm-as-a-judge. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 292–314.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. 2025. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

A Licensing of Data and Software

M3 Challenge data. The Math Modeling Challenge problems and award-winning reports are publicly available on the M3 Challenge website for educational and research purposes. Our use of these reports for benchmark construction is consistent with this intended use.

Agent frameworks. DeerFlow, ms-agent, and LangChain Open Deep Research are released under their respective open-source licenses (MIT/Apache-2.0); we use them through their public APIs in accordance with these terms.

LLMs. All proprietary LLMs are accessed through their official APIs in accordance with provider terms of service. Open-weight LLMs are used under their published model licenses.

B Domain Coverage of M3 Challenge Source Questions

The M3 Challenge problems span a broad range of real-world analytical domains. Table 7 summarizes the yearly source problems used in our data collection and their primary application domains. Although each annual challenge typically consists of multiple sub-questions, we group them by year-level problem theme to illustrate the breadth of the source corpus. The topics cover public finance, energy policy, demographic analysis, transportation planning, environmental sustainability, education economics, public health, digital infrastructure, labor markets, housing policy, and climate resilience. This diversity supports our goal of evaluating generalizable analytical reasoning in open-ended mathematical modeling reports, rather than performance on a single domain-specific report genre.

C Human Annotation

C.1 Error Detection

The annotation was performed by two CS PhD students. We provide standardized annotation instructions for annotators to focus on soundness errors. Each instance is shown to the annotator together with: (i) the original question, (ii) the corresponding human reference report, (iii) the LLM-generated report, and (iv) the automatically detected candidate error (its error type, the specific sentence containing the error, and the system's reasoning). The annotator is asked to perform the following steps:

1. **Verify the candidate error.** Decide whether the candidate error is a genuine soundness flaw in the LLM-generated report, judged against the reference report and the error taxonomy. The annotator marks each candidate as `valid`, `invalid`, or `partially valid`.

2. **Revise the error type.** If the candidate is genuine but its predicted error type is incorrect, the annotator selects a more appropriate type from the taxonomy in Table 2.

3. **Adjust the sentence span.** If the candidate's localized sentence span is incomplete or misaligned, the annotator corrects the span to point to the exact sentence(s) where the error occurs.

4. **Supplement missing errors.** The annotator scans the report for soundness errors that were not proposed by the system; for each newly found error, the annotator records its type and sentence span.

Annotators are instructed to focus exclusively on *soundness* errors (e.g., unjustified parameters, missing data sources, derivation inconsistencies) and not to flag stylistic, grammatical, or formatting issues. Definitions and illustrative examples of each error type from Table 2 are made available to the annotator throughout the task. The task does not involve any sensitive or personally identifying content; all data come from publicly available competition reports and DRA-generated outputs.

The inter-annotator agreement reached 0.92. We only keep the valid errors and missing errors that both annotators agree on.

C.2 Error Injection

To ensure the quality and validity of the synthetic flaws introduced in our dataset, we conducted a human validation study on the error injection process. Two graduate students were recruited to annotate a subset of the injected reports based on two key criteria: (1) **Injection Success**, which evaluates whether the error was logically integrated into the report without introducing unintended artifacts, and (2) **Detection Difficulty**, which assesses the subtlety of the error and the effort required for a human expert to identify it.

The annotation results yielded an exceptional inter-annotator agreement of 0.98, reflecting near-perfect consensus on the nature of the injected flaws. Crucially, the error injection success rate was found to be 0.97, indicating that the pipeline is highly reliable in generating valid logical discrepancies. These results demonstrate that the vast majority of our injected errors are both technically sound and challenging, providing a robust foundation for evaluating automated detection systems.

During the construction of the Golden Set, human annotators rated the perceptibility of each injected error on a three-tier scale: *Obvious*, *Moderate*, and *Subtle*, where a higher tier indicates easier detection. The distribution of the annotated dataset reveals that 50.8% of errors were classified as Obvious, 31.0% as Moderate, and 18.2% as Subtle. This distribution suggests that while the dataset contains challenging instances, the majority of logical errors remain clearly distinguishable to human experts, confirming that the task is well-posed and solvable by human standards.

C.3 Using GPT-5-mini as the Scoring Model

We use GPT-5-mini as the scoring model to determine whether a judge-predicted error matches a ground-truth annotation. To verify that this automatic scoring is reliable, we conduct a human evaluation on a randomly sampled subset of 50 scoring decisions produced by GPT-5-mini. Each sampled instance consists of a judge-predicted error (its predicted error type and sentence span) paired with a ground-truth annotation, together with the binary match/no-match decision produced by GPT-5-mini. A human annotator independently re-evaluates each decision, and we find a 100% agreement rate between GPT-5-mini and the human annotator.

This high level of agreement is consistent with the nature of the scoring task. Rather than judging the correctness of an analytical claim, the scoring model only needs to determine whether a predicted error matches a ground-truth annotation along two well-defined dimensions: the error type (drawn from a fixed taxonomy) and the sentence span it refers to. Both dimensions are concrete and largely surface-level, which makes the matching task substantially easier than the soundness-flaw detection task itself.

We choose LLM-based scoring rather than exact string matching because predicted and ground-truth spans often differ in superficial ways that exact matching cannot tolerate—e.g., differences in LaTeX formatting (such as `$\theta$` vs. `theta`, or `\times` vs. `x`), in whitespace, in punctuation, or

in minor paraphrasing of the same sentence. Exact matching would penalize such cosmetic mismatches and underestimate true judge performance. GPT-5-mini, in contrast, tolerates these surface variations while still requiring the predicted error to refer to the same content and the same error type as the annotation. Combined with the 100% human-agreement rate above, this makes GPT-5-mini a robust and practical choice for automatic scoring on OSJBENCH.

D Example of Error Injection

In this section, we present three examples comparing the original human-authored text with the error-injected version. We highlight the modifications in **bold**. These examples demonstrate how the pipeline introduces logical flaws while maintaining the original sentence structure and professional tone.

D.1 Error Type: Data Sourcing and Credibility

Original Text: Childcare cost estimates are derived from **Care.com Editorial Staff (2021)**, and productivity parameters are based on **Bloom et al. (2015)**.
Injected Text: Childcare cost estimates are derived from **publicly available online summaries of typical U.S. rates**, and productivity parameters are based on **prior remote-work research findings reported in the broader literature**.
Analysis: The specific, verifiable citations are replaced with vague, unverifiable descriptors ("online summaries", "broader literature"), creating a data sourcing error.

D.2 Error Type: Unjustified Parameters

Original Text: To account for seasonality and trend, we employ an ARIMA(1,1,1) model. **By fitting the model to the monthly housing inventory data from January 2015 to December 2023, we determine the autoregressive coefficient $\phi = 0.72$ and the moving average parameter $\theta = -0.45$.**
Injected Text: To account for seasonality and trend, we employ an ARIMA(1,1,1) model. **We set the autoregressive coefficient $\phi = 0.72$ and the moving average parameter $\theta = -0.45$.**
Analysis: The original text specifies the exact dataset and time range used to fit the ARIMA parameters. The injected text presents the same specific coefficients but removes the estimation process.

D.3 Error Type: Unjustified Quantitative Result

Original Text: Using the $\widehat{H}(t)$ values from the earlier forecast and the cost parameters derived above, we calculate the incremental capital cost to be approximately $273,900,000.
Injected Text: Based on the planning requirements for the initial phase of the city's expansion project, we calculate the incremental capital cost to be approximately $273,900,000.
Analysis: The derivation link ("Using H(t)... and parameters") is removed. The number $273M now appears as a "magic number" without a traceable calculation path, despite the sentence remaining fluent.

E SFT Training Details

We fine-tuned Qwen3.5-35B-A3B with full-parameter supervised fine-tuning. The training data were constructed from the controlled full-report benchmark. Each example uses the complete mathematical modeling report as input, together with the active soundness-error taxonomy and the corresponding instance-level rubric. The target output is a JSON list of gold soundness errors, where each error contains its type, an exact quoted span from the report, and a short explanation.

Training was run on $4\times$A100 80GB GPUs. We use bf16 precision, a maximum sequence length of 6144, global batch size 16, and per-GPU micro-batch size 4. We train for 4 epochs with AdamW and learning rate 5×10^{-6}. To fit the 35B model, we enable gradient checkpointing, parameter offloading, and optimizer offloading. We disable AdamW for each of the operations to avoid the extra optimizer-step memory spike observed in preliminary runs. For evaluation, we serve the fine-tuned checkpoint with SGLang and keep Qwen thinking enabled.

F Graph Examples and Quality Evaluation

F.1 Definition of Logical Graph

Formally, given a raw report $\mathcal{R}$, we construct a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The graph consists of nodes representing reasoning units and edges representing operations.

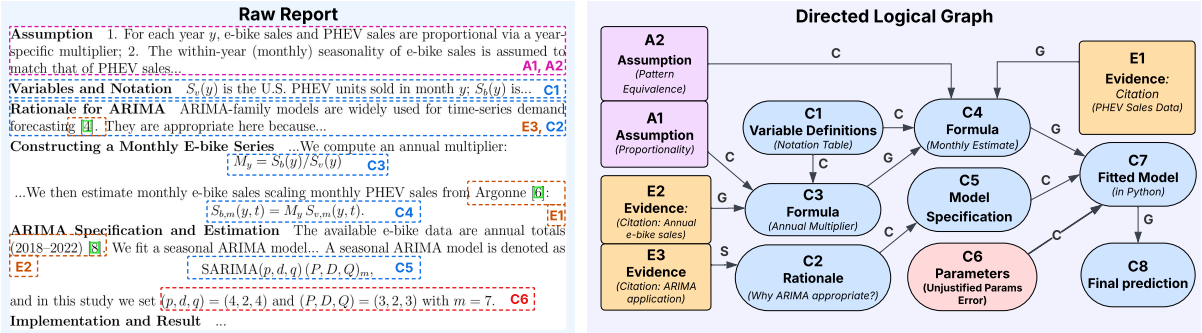

Figure 2: An example of a Directed Logical Graph derived from raw report. The graph consists of three node types (`Assumption` (A), `Evidence` (E), and `Claim` (C)) connected by three edge types (`Generate` (G), `Support` (S), and `Condition` (C)).

Nodes ($\mathcal{V}$). The node set consists of three distinct types of units that capture the content of the report. First, a Claim ($\mathbf{c}_i$) encompasses assertions and conclusions generated in the report, including specific model decisions (e.g., choosing a Logistic Growth Model), parameter values (e.g., growth rate $r = 0.5$), derived quantitative results, and final conclusions. Second, Evidence ($\mathbf{e}_j$) represents the external inputs that ground the analysis, such as external data points and bibliographic citations. Third, an Assumption ($\mathbf{a}_k$) represents the explicit conditions or simplifications proposed in the report (e.g., *"Assuming the population growth is constant"*).

Edges ($\mathcal{E}$). The edges are **Operations** that show the connection between each node. The edge set $\mathcal{E}$ encodes three specific logical flows: `Generate` ($\mathbf{e}/\mathbf{c}_{in} \rightarrow \mathbf{c}_{out}$), which represents the calculation or derivation process where a quantitative result is produced;

`Support` ($\mathbf{e} \rightarrow \mathbf{c}$), which represents direct grounding where an evidence node validates a claim (e.g., a hyper-parameter cited from a source); and `Condition` ($\mathbf{a}/\mathbf{c}_{in} \rightarrow \mathbf{c}_{out}$), which is the most common edge and represents the claims or assumptions applied to be the condition of the continual claim.

F.2 Logical Graph Extraction and Judging

LogicGraphJudge functions through a two-stage process: extracting the logical graph from the textual report, followed by checking the graph for logical issues. Both stages are implemented by prompting an LLM.

First, we prompt the LLM to extract the logical graph from the text with the graph definition. The model parses the text to identify nodes and relationships to form edges.

For instance, as shown in Figure 2, the LLM identifies the textual assumption of proportionality as node a_1 and the cited "2018-2022 Annual e-bike sales" data as evidence node e_2. It then constructs the reasoning structure by linking both the data source (e_2) and the theoretical constraint (a_1) to the derived "Annual Multiplier" formula (c_3) via `Generate` and `Condition` edges, respectively. Besides, we conduct a manual inspection to confirm that current LLMs can faithfully preserve reasoning gaps during extraction without hallucinating corrective links, ensuring the graph accurately reflects the text's logical flaws.

Once the graph is constructed, the LLM is prompted to inspect it to identify logical errors. First, the judge inspects the intrinsic properties of each node based on type-specific rules. For example, for *Claim* and *Assumption* nodes, it verifies whether the content violates physical common sense; for nodes containing quantitative variables, it checks for missing physical units.

Second, with an explicit definition of standards, such as a hyper-parameter node must possess an incoming edge from an *Evidence* or other *Claim* node to be considered valid, the LLM validates the connections between nodes.

The model audits the current graph against these criteria, and the inspection results are recorded as structured metadata appended directly to the corresponding nodes or edges within the graph.

F.3 Graph Quality Annotation

A potential concern with the graph representation is that the extraction step itself might silently "repair" flawed reasoning in the original report (for example, by inferring a missing dependency or filling in an unstated assumption), which would mask the very errors we aim to detect and overstate the effectiveness of the graph-based judge. To verify that

this does not happen, we manually inspect 30 sampled extracted graphs and check, for each ground-truth soundness error in the corresponding report, whether the original flaw is still observable in the graph. We find no notable evidence of such "repair": the original reasoning gaps are consistently preserved in the extracted structure rather than being silently fixed by the extractor. This indicates that the graph representation faithfully reflects the report's actual reasoning, and the improvements attributed to the graph stem from making existing dependencies easier to inspect rather than from the extractor introducing missing structure.

G Prompts in Experiments

G.1 Agent Report Generation Prompt

Mathematical Modeling Report Prompt

You are an expert mathematical modeler and academic writer tasked with producing a complete formal solution report for the following problem.

Problem
{question}

Modeling constraint, if any
{modeling_constraint}

Search and evidence requirements

1. Use actual web-search evidence when external data, statistics, or real-world context are needed.
2. Cite the external sources used for empirical data, parameter calibration, historical trends, or domain facts.
3. If exact data are incomplete, make defensible estimates from the searched evidence, state the approximation clearly, and still complete the report.

Report requirements

1. The response must be a complete mathematical modeling report, not an outline.
2. Include all of the following sections: Introduction, Model Assumptions, Variables and Parameters, Model Development, Model Application and Results, Validation/Sensitivity/Uncertainty, Discussion, Conclusion, and References.
3. Define variables, parameters, units, and data sources before using them in formulas.
4. Present equations, tables, and calculations clearly enough for an external reader to audit the reasoning.
5. Justify model choices, important parameter choices, and any extrapolation or data-processing step.
6. Discuss validation, sensitivity, uncertainty, and limitations.
7. Provide a complete References section.

Output format

- Return only the final report.
- Use Markdown sections with LaTeX equations, or a clean LaTeX report body.
- Do not include meta commentary, code fences, placeholders, or statements that the task cannot be completed.

G.2 Report Comparison Prompt

Report Comparison Prompt

You are an expert judge for a mathematical modeling challenge. You are given a reference human award-winning report and a generated report. Your task is to compare their reasoning quality and identify soundness flaws in the generated report.

Problem
{question}

Reference human report
{human_report}

Generated report
{generated_report}

Active soundness-error taxonomy
{taxonomy}

Instructions

1. Use the human report only as a reference for what a sound reasoning path may look like; do not require the generated report to use the same model.
2. Identify concrete soundness errors in the generated report under the active taxonomy.
3. Do not flag ordinary style issues, missing exposition, or debatable modeling choices unless they create a local soundness error.
4. Quote the exact generated-report span that exhibits each error.
5. Do not invent error categories outside the active taxonomy.

Output format
Return a JSON object with a top-level `errors` list. Each item in `errors` contains:

- `error_id`: a local identifier such as `E1`;
- `error_type_id`: one active taxonomy id;
- `error_type`: the canonical type name;
- `quote`: an exact span from the generated report;
- `why_error`: the specific reason this span is a soundness error;
- `confidence`: a numeric confidence score.

G.3 Error Injection Prompt

Error Injection Prompt

Create controlled soundness-error injection edits for one mathematical modeling report.

You will receive the problem, requested error types, and candidate snippets copied exactly from the report. Make small replacement edits only. The final benchmark report will be created by replacing each `original_span` with `replacement_span` in the original report.

Problem
{question}

Modeling constraint
{modeling_constraint}

Requested error types, definitions, and injection guidance
{target_error_types_with_definitions_and_tips}

Candidate report snippets
{candidate_snippets}

Rules

1. Output exactly one JSON object, no Markdown.
2. Prefer one edit per requested error type. If a requested type is not feasible from the snippets, omit it and list it in `omitted_type_ids`.
3. Each edit must copy `original_span` exactly from a snippet, including LaTeX when present.
4. `replacement_span` should be a local replacement, not a full-report rewrite.
5. `replacement_span` must create a clear, locally checkable soundness error of the requested type.
6. Do not introduce task-compliance, formatting, style, or model-constraint errors.
7. Do not critique assumptions merely because they are debatable. The injected error must be concrete and auditable from the altered report.
8. Avoid tiny rounding changes. If using arithmetic, make the contradiction material enough to affect a result, ranking, sign, bound, or conclusion.
9. Do not use the same `original_span` twice.
10. Do not add labels such as “error”, “incorrect”, “injected”, or other meta-comments.

Required JSON schema
Return a JSON object with:

- `status`: `ok`;
- `edits`: a list of local replacement edits;
- `omitted_type_ids`: requested type ids that could not be injected;
- `notes`: optional implementation notes.

Each item in `edits` contains:

- `type_id`: one requested type id;
- `snippet_id`: the id of the selected snippet, such as `S1`;
- `original_span`: exact text copied from a snippet;
- `replacement_span`: replacement text to insert;
- `raw_sentence`: the smallest exact altered span that reveals the injected error;
- `why_error`: a brief reason why this is a soundness error;
- `severity`: one of `minor`, `major`, or `critical`.

G.4 Graph Construction Prompt

Logical Graph Extraction Prompt

You are a **Logical Graph Engineer**. Your task is to extract the **high-level reasoning skeleton** of the provided analytical report into a Directed Logical Graph.

 # 1. NODE ONTOLOGY Classify every key element into one of these Types/Subtypes:

- **CLAIM** (The Reasoning Steps):
  - `VARIABLE_DEF`: A group of variable definitions or notation tables.
  - `RATIONALE`: Text explaining *why* a method was chosen.
  - `FORMULA`: Mathematical equations or calculation logic.
  - `MODEL_REASON`: The specific model configuration or code implementation.
  - `PARAMETER`: Key hyperparameters or constants.
  - `CONCLUSION`: The **final** answers to the prompt's main questions.
- **EVIDENCE** (The Grounding):
  - `CITATION`: References to external papers, websites, or datasets.
- **ASSUMPTION** (The Constraints):
  - `SIMPLIFICATION`: Deliberate simplifications.

 # 2. EDGE ONTOLOGY Identify relationships using these three specific flows:

- **GENERATE** (Derivation Numerically):
  - Logic: Input → Formula → Model → Conclusion. The output must be a numeric value.
- **SUPPORT** (Grounding):
  - Logic: Evidence → Rationale; Evidence → Parameter. The input is a supporting evidence for the target node.
- **CONDITION**:
  - Logic: Assumption → Formula/Model; Variable_Def → Formula; Formula → Formula, etc.

OUTPUT FORMAT
Return **ONLY** a valid JSON object.

```
{
  "nodes": [...],
  "edges": [...]
}
```

REPORT TEXT: {report text}

G.5 Graph Inspection Prompt

Forensic Logic Audit Prompt

You are a **Logical Graph Inspection**. You have a Logical Graph representing the reasoning structure of a report. Your task is to inspect every Node and Edge against the **Standards of a Rigorous Reasoning Process**.

 # INSTRUCTIONS Output the exact same JSON graph, but add a `checking_result` field to every Node and Edge.

- **If the element meets the standard**: Write "PASS - [Brief reason why it passes]".
- **If the element fails**: Write "FLAG - [Describe what is missing to make the reasoning complete]".

 # STANDARDS OF A RIGOROUS REASONING PROCESS

A. Node Integrity Standards (Intrinsic Validity) *Inspect the intrinsic properties of each node based on its type.*

- **Physical Common Sense:** For any `CLAIM` (e.g., Results, Parameters) or `ASSUMPTION` node, the content must not violate basic physical laws or domain logic (e.g., "Negative mass", "Probability > 1").
- **Unit Completeness:** For any node containing quantitative variables or numerical results, explicit physical units must be recorded.
- **Source Verifiability:** For any `EVIDENCE` node, the description must be specific enough to be verified (e.g., citing a specific year, author, or dataset name).
- **Model Justification:** For any node representing a Model Specification (`MODEL_SPEC`) or Rationale (`RATIONALE`), the text must contain substantive explanation for why the method was chosen.

B. Connectivity Consistency Standards (Logical Flow) *Inspect the connections between nodes to ensure a coherent reasoning chain.*

- **Parameter Dependency:** If a `CLAIM` node represents a Hyper-parameter Choice or a Constant, it must possess at least one incoming edge from an `EVIDENCE` node (citation) or another `CLAIM` node (derivation).

- **Result Derivation:** If a node represents a Final Conclusion or Intermediate Result, it requires an incoming `GENERATE` edge specifically from a `FORMULA` or `MODEL_SPEC` node.
- **Formula-Variable Grounding:** Any node representing a Mathematical Formula should ideally have an incoming `CONDITION` edge from a `VARIABLE_DEF` node to ensure all symbols are defined.
- **Constraint Consistency:** For any `CONDITION` edge, the source node (Assumption or Prior Conclusion) must not logically contradict the target node.
- **Evidence Relevance:** For any `SUPPORT` edge, the content of the Evidence node must logically support the target Claim.

 # INPUT GRAPH
{json_graph}

 # OUTPUT FORMAT
Return **ONLY** the valid JSON object with the added `checking_result` fields.

G.6 Instance-level Rubric Prompt

Instance-Level Rubric Generation Prompt

You are generating an instance-level soundness rubric for a judge that will later inspect generated mathematical modeling reports.

Important leakage rule
You cannot see any generated report, human report, golden errors, injected errors, or previous annotations. Use only the problem, modeling constraint, and taxonomy.

Problem
{question}

Modeling constraint, if any
{modeling_constraint}

Active soundness-error taxonomy
{taxonomy}

Rubric design requirements

1. Create concrete checks tailored to this problem's likely data, variables, model outputs, parameters, validation, sensitivity, uncertainty, and recommendation structure.
2. Checks should help detect obvious or moderately clear soundness errors, not demand a perfect solution.
3. Do not require a particular modeling approach unless the problem or modeling constraint explicitly requires it.
4. Produce 8 to 12 checks.
5. Keep each field concise: one sentence, at most 30 words.
6. Do not include examples, report-specific spans, golden labels, or chain-of-thought.
7. Prioritize likely local checks for the active type ids.
8. Do not include checks for any type id outside the active list.

Output format
Return only valid JSON with:

- `rubric_generator`: the generator name;
- `rubric_scope`: what the rubric is designed to check;
- `checks`: a list of instance-level checks.

Each item in `checks` contains:

- `check_id`: a local id such as `R1`;
- `target_error_type_ids`: one or more active type ids;
- `what_to_check`: a specific instance-level check;
- `evidence_needed`: what a sound report should provide;
- `flag_when`: when the later judge should flag an error;
- `do_not_flag`: what should not be counted as an error.

G.7 LLM-as-a-Judge Prompts

Natural-Language Judge Prompt

You are evaluating soundness errors in a mathematical modeling report.

Problem
{question}

Modeling constraint, if any
{modeling_constraint}

Active soundness-error taxonomy
{taxonomy}

Report
{report_text}

Instructions

1. Find concrete soundness errors in the report using only the active taxonomy ids.
2. Quote exact report spans; the quote must appear in the report.
3. Systematically check formula-result consistency, data provenance, extrapolation, manual and fitted parameters, objective/constraint implementation, validation, sensitivity, uncertainty, direct contradictions, and recommendations.

4. Do not output ordinary debatable assumptions, model family critiques, writing issues, or formatting issues.

5. Return every distinct local error that is directly checkable, up to 12 errors. Do not fill a quota.

Output format
Return a JSON object with a top-level `errors` list. Each item in `errors` contains:

- `error_id`: a local identifier such as `E1`;
- `error_type_id`: one active type id;
- `error_type`: the canonical type name;
- `quote`: an exact report span;
- `why_error`: the specific reason this span is a soundness error;
- `confidence`: a numeric confidence score.

Graph-Guided Judge Prompt

You are evaluating soundness errors in a mathematical modeling report using an internal directed logical-graph audit.

Problem
{question}

Active soundness-error taxonomy
{taxonomy}

Graph
{Extracted_graph}

Report
{report_text}

Internal graph-audit procedure

1. Identify nodes for DATA/SOURCE, DATA_PROCESSING, VARIABLE_DEF, PARAMETER, FITTED_STATISTIC, FORMULA, MODEL_SPEC, RESULT_TABLE, VALIDATION/SENSITIVITY/UNCERTAINTY, and CONCLUSION/RECOMMENDATION.

2. Identify missing or invalid `SUPPORT`, `GENERATE`, and `CONDITION` links.

3. First check direct formula-result arithmetic, unit/scale, bound/capacity, table-to-table, and statement-to-table contradictions.

4. Then check unsupported data processing, parameters/fitted statistics, validation/sensitivity/uncertainty, model-mechanism/objective implementation, and recommendations.

5. Convert only concrete graph failures into active taxonomy errors. Every emitted error must quote an exact report span.

Output format
Use the same JSON schema as the Natural-Language Judge Prompt.

Instance-Rubric Judge Prompt

You are evaluating soundness errors in a mathematical modeling report using an instance-level rubric.

Problem
{question}

Active soundness-error taxonomy
{taxonomy}

Instance-level rubric
{instance_level_rubric}

Report
{report_text}

Instructions

1. Apply every rubric check systematically, but output only real active-taxonomy soundness errors.

2. Quote exact report spans from the report, not from your explanation.

3. For each rubric check, search the report for a local span involving the specific object in `what_to_check`.

4. If the span states a concrete value, fitted estimate, constructed input, validation result, sensitivity result, uncertainty band, ranking, or score but the report does not provide the `evidence_needed`, emit that span using the check's target type.

5. Do not require generic trigger words such as "without evidence"; a report can violate a rubric check by presenting the object/value/claim without the required support.

6. Then check direct formula-result, arithmetic, unit, bound, table, and contradiction errors.

7. Return every distinct directly checkable error, up to 12 errors.

Output format
Use the same JSON schema as the Natural-Language Judge Prompt.

Instance-Rubric Graph Judge Prompt

You are evaluating soundness errors in a mathematical modeling report using both an instance-level rubric and an internal directed logical-graph audit.

Problem
{question}

Active soundness-error taxonomy
{taxonomy}

Instance-level rubric
{instance_level_rubric}

Graph
{Extracted_graph}

Report
{report_text}

Internal graph-rubric audit

1. Use the rubric checks as targeted probes over those nodes.
2. For each rubric check, build a small graph around the object in `what_to_check`. If a report span creates a relevant node but lacks the check's required `SUPPORT`, `GENERATE`, or `CONDITION` evidence, emit that span using the check's target type.
3. Do not require the span to explicitly say "unsupported" or "without evidence".
4. Then identify direct formula-result, unit/scale, bound/capacity, table-to-table, and statement-to-table contradictions.
5. Output only concrete active-taxonomy errors with exact report spans.

Output format
Use the same JSON schema as the Natural-Language Judge Prompt.

G.8 Judge Result Evaluation

Error Detection Evaluation Prompt

You are a professional quality auditor tasked with evaluating the accuracy of an error detection system.

Input Data

- **Golden Error (Ground Truth):** {golden_error}
- **Detected Error (System Output):** {detected_error}

Matching Rules (Strictly Follow)

1. **Location/Sentence Priority (Crucial):** A detected error matches a golden error if the text in its `error_sentences` list significantly overlaps with or is semantically identical to the golden `error_sentence`.
2. **Type Consistency:**
   - If the locations match, verify if the `error_type` is semantically consistent.
   - **Synonyms/Sub-categories are OK:** e.g., "Unjustified Parameters" and "No Model Justification" match if they describe the same flaw in the same sentence.
   - **Fundamental Mismatch:** If the types describe entirely different issues (e.g., "Data Sourcing" vs. "Unit Consistency"), it is NOT a match even if the sentence is the same.
3. **Structural Unpacking:** The `detected_error` groups multiple sentences under one type. You must evaluate each sentence in those lists individually against the items in the `golden_error` list.
4. **One-to-One Mapping:** Each golden error index can be matched at most once.

Output Requirements

1. **Reasoning:** For each Golden Error index (0, 1, 2...), explain why it was matched or not matched. Reference specific groups or sentences from the Detected Error.
2. **Final Line:** You **MUST** end your response with exactly one line in the following format: `detected error in golden error: [indices]`

Hard Constraints

- Indices in the final list must be unique and sorted in ascending order.

H Artifact Use Consistent With Intended Use

Our use of external artifacts is consistent with their intended educational, research, and software-development purposes. The M3 Challenge problems and award-winning reports are publicly available for educational and research use, and we use them only as source materials for constructing and evaluating a benchmark for open-ended mathematical modeling reports. We do not redistribute the original reports as a standalone replacement for the M3 Challenge resources. Instead, we derive benchmark instances, annotations, and controlled error-injected variants for research on soundness-flaw detection.

The open-source agent frameworks used in this work, including DeerFlow, ms-agent, and LangChain Open Deep Research, are used as research software to instantiate deep research agent pipelines. We follow their public documentation and use them under their respective open-source licenses. The proprietary LLMs used in this work are accessed through official APIs, and open-weight LLMs are used under their published model licenses. Search tools and APIs are used only to support report generation in the intended manner of web-assisted research agents.

The resulting benchmark and judging artifacts are intended for research use in evaluating and improving the ability of LLM judges to detect soundness flaws in open-ended analytical reports. They are not intended to serve as an authoritative grading system for real-world mathematical modeling competitions, student assessment, or high-stakes professional decision making without human over-

sight. In particular, the benchmark is designed to measure typed, sentence-localized soundness-error detection under controlled experimental conditions, rather than to certify the overall quality or correctness of arbitrary real-world reports.

I Parameters For Packages

Most software artifacts used in this work are agent frameworks, API clients, or inference and training packages rather than statistical packages with tunable scientific parameters. For the open-source deep research frameworks, we use their default tool-calling and report-generation settings unless otherwise specified in the experiment description. The main experimental variation comes from the choice of backend LLM, rubric type, and input representation, rather than from framework-specific hyperparameters.

For report generation, we use two direct API settings and twelve open-framework settings. The direct API settings are o4-mini-deep-research and Tongyi-DeepResearch-30B. The open-framework settings combine three agent harnesses, DeerFlow, ms-agent, and LangChain Open Deep Research, with four backend LLMs: GPT-5.4-nano, Qwen3.5-35B-A3B, Gemma4-26B-A4B-IT, and DeepSeek-V4-Flash. Open-framework settings use the Tavily search engine. We sample 10 reports per setup, yielding 140 DRA-generated reports in total.

For judge evaluation, each judge is evaluated under the same scoring protocol. The input to the judge is either the natural-language report or the extracted logical graph, combined with either the general rubric or the instance-level rubric. The instance-level rubric and logical graph are generated by the same LLM that serves as the judge, so that each judging pipeline remains self-contained. The final prediction is formatted as a set of typed soundness errors with sentence-level localizations.

For supervised fine-tuning, we fine-tune Qwen3.5-35B-A3B with full-parameter supervised fine-tuning. Training is conducted on 4×A100 80GB GPUs with bf16 precision, maximum sequence length 6144, global batch size 16, per-GPU micro-batch size 4, 4 training epochs, AdamW optimization, and learning rate 5×10^{-6}. To fit the 35B model, we enable gradient checkpointing, parameter offloading, and optimizer offloading. For evaluation, we serve the fine-tuned checkpoint with SGLang and keep Qwen thinking enabled.

J AI Assistants In Research

AI assistants were used as research tools in several controlled parts of the project. First, LLMs were used to generate mathematical modeling reports under predefined prompts and model constraints, which forms part of the object of study in this work. Second, LLMs were used in an annotation-assistance pipeline: GPT-5.5 was used to propose candidate soundness errors by comparing DRA-generated reports with corresponding human reference reports, and Claude Opus-4.7 was used in a cross-model verification pass to validate and supplement these candidates. Human annotators subsequently verified the candidate errors, revised error types when necessary, corrected sentence spans, and added missing errors.

AI assistants were also used to support the construction of synthetic benchmark instances. Specifically, LLMs were used as locator and rewriter components in the controlled error-injection pipeline. The locator identifies sentence spans semantically aligned with a target error type, and the rewriter introduces a soundness degradation while preserving the surrounding document structure and fluency. These injected errors were guided by a taxonomy and empirical frequency distribution induced from human-verified native DRA errors, and their quality was further checked by human validation.

In the evaluation pipeline, LLMs were used to generate instance-level rubrics, extract logical graphs, perform soundness-error detection, and score whether predicted errors matched ground-truth annotations. These uses are part of the experimental design rather than hidden assistance. We report the relevant prompts, model choices, and validation procedures in the appendix to support reproducibility.

AI assistants were also used for limited writing support, including grammar polishing, improving clarity, and formatting LaTeX text. They were not used to fabricate experimental results, alter reported numerical findings, or replace human judgment in the final validation of benchmark annotations. All scientific claims, experimental designs, and final reported results were reviewed and approved by the authors.

Year	M3 Challenge Topic	Primary Domain	Representative Analytical Focus
2006	Solving the Social Security Stalemate	Public finance and social policy	Long-term solvency, demographic change, retirement-system sustainability
2007	Beat the Street!	Finance and investment	Portfolio selection, risk-return trade-offs, market performance modeling
2008	Energy Independence Meets the Law of Unintended Consequences	Energy, agriculture, and environmental policy	Corn ethanol production, fuel substitution, unintended policy impacts
2009	$787 Billion: Will the Stimulus Act Stimulate the U.S. Economy?	Macroeconomics and public policy	Economic stimulus allocation, employment effects, policy effectiveness
2010	Making Sense of the 2010 Census	Demographics and political representation	Census undercount adjustment, congressional apportionment, districting fairness
2011	Colorado River Water: Good to the Last Acre-Foot	Water resources and regional planning	Drought impact, hydrologic forecasting, water and power sustainability
2012	All Aboard: Can High Speed Rail Get Back on Track?	Transportation and infrastructure	High-speed rail placement, implementation cost, energy-dependence effects
2013	Waste Not, Want Not: Putting Recyclables in Their Place	Waste management and environmental sustainability	Plastic waste quantification, recycling strategy, national recycling standards
2014	Lunch Crunch: Can Nutritious Be Affordable and Delicious?	Nutrition, education, and public health	Student caloric needs, affordable meal planning, nutritional constraints
2015	Stem Sells: What Is Higher Education Really Worth?	Education economics and labor market analysis	College cost, expected earnings, career outcomes, lifestyle trade-offs
2016	Share and (Car) Share Alike: Modeling New Approaches to Mobility	Urban mobility and transportation economics	Car-sharing adoption, driver behavior, emerging transportation technologies
2017	From Sea to Shining Sea: Looking Ahead with the National Park Service	Environmental management and tourism policy	National park demand, climate-related risks, resource planning
2018	Better Ate Than Never: Reducing Food Waste	Food systems and sustainability	Food waste reduction, cost-benefit analysis, community-level intervention planning
2019	One is Too Many and A Thousand Not Enough: Substance Use and Abuse	Public health and social cost analysis	Substance-use prevalence, societal cost, intervention prioritization
2020	Keep On Trucking: U.S. Big Rigs Turnover From Diesel to Electric	Freight transportation and energy infrastructure	Electric semi-truck adoption, charging-station placement, route prioritization
2021	Defeating the Digital Divide: Internet Costs, Needs, and Optimal Planning	Digital infrastructure and equity	Broadband access, cost modeling, rural and low-income service planning
2022	Remote Work: Fad or Future	Labor markets, real estate, and environmental impacts	Remote-work adoption, sectoral impacts, population and economic shifts
2023	Ride Like the Wind Without Getting Winded: The Growth of E-Bike Use	Sustainable transportation and urban planning	E-bike adoption, substitution for car travel, energy and mobility impacts
2024	A Tale of Two Crises: The Housing Shortage and Homelessness	Housing policy and social welfare	Housing supply, affordability, homelessness, long-term urban planning
2025	Hot Button Issue: Staying Cool as the World Heats Up	Climate resilience, energy demand, and emergency planning	Heat-wave impacts, indoor temperature modeling, grid demand and resource allocation

Table 7: Domain coverage of the M3 Challenge source problems. The annual problems span diverse real-world domains, supporting our benchmark's focus on generalizable analytical reasoning across open-ended mathematical modeling reports rather than a single specialized application area.